



UNIVERSITÉ DE STRASBOURG
UFR DE MATH-INFO
IRMA STRASBOURG



Project report

To validate the fourth semester of This Master's program

MASTER

in Applied Mathematics

With the topic of

Learning Green's functions using neural operators

Presented by
M. PAWLUS Jérémy

Under the supervision of
Dr. FRANCK Emmanuel
Dr. MICHEL-DANSAC Victor
Dr. DELEFORGE Antoine

Academic Year 2023-2024

Abstract

This Master’s thesis presents a novel approach to solving the Helmholtz equation using Deep Green Neural Operators. The primary objective of this research is to develop a reduced model capable of handling general geometries and efficiently solving inverse problems. Initially, the model is constructed for simple cases and progressively enriched for more complex scenarios.

The research begins with a comprehensive exploration of functional analysis, focusing on PDEs and Green’s functions. This foundation supports the development of the Deep Green Neural Operators framework, which integrates scientific machine learning principles, particularly physics-informed neural networks (PINNs). The core innovation lies in approximating Green functions using neural networks, enabling rapid and accurate solutions across various discretizations and media properties.

Data sampling and implementation are critical components of this work. This Master’s thesis introduces Scimba, a PyTorch-based library offering classes related to scientific machine learning (SciML) for numerical analysis. Procedures for Gaussian process sampling and finite differences methods are detailed. The evaluation focuses on Multi-Layer Perceptrons (MLPs) and Fourier Nets, the latter proving especially relevant for oscillatory test cases like the Helmholtz equation. The mathematical formulation of the problem, including criteria and loss functions, is elaborated, and the evolution of the implementation is documented, highlighting challenges, alternatives, and solutions.

The numerical results section showcases the application of the Deep Green Neural Operators approach to several test cases: starting with a simple case (Laplace equation, homogeneous boundary conditions, no parameters, no oscillations), progressing to more complex cases (Advection-diffusion equation with inhomogeneous boundary conditions, without parameters), and culminating with the Helmholtz equation (homogeneous boundary conditions, with and without oscillations and parameters). For each case, the exact Green function is determined and used as reference where possible, and the predicted solutions are compared to the true targets, with visualizations provided for both the Green functions and the homogeneous solutions.

This Master’s thesis concludes with a synoptic and objective overview of the research, a self-appreciation of the internship experience, and perspectives on future work. Significant difficulty was encountered in learning parameterized oscillatory PDEs, especially for the Helmholtz equation when the training range for the parameters was set near eigenfrequencies. Highlighting the relevancy of neural networks, future work will focus on leveraging their capabilities to detect anomalous vicinities that complicate the learning process. This extensive computational study in the one-dimensional setting demonstrates the potential of Deep Green Neural Operators to significantly and positively impact the solving of PDEs, offering significant advancements in computational efficiency and accuracy.

Contents

1	Introduction	5
1.1	Preliminary information	5
1.1.1	Acknowledgements	5
1.1.2	Preliminary remarks	5
1.2	Context of the project	6
1.2.1	General context	6
1.2.2	Specific context	6
1.3	Objectives of the project	7
1.3.1	Academical objectives	7
1.3.2	Scientific objectives	7
1.4	Estimation of the Time Needed	8
2	Preliminary mathematical notions	11
2.1	Basic notions of partial differential equations (PDEs)	11
2.1.1	Introduction to Differential Equations	11
2.1.2	Boundary and Initial Conditions	11
2.1.3	About elliptic PDEs	12
2.1.4	The equations involved in the internship	13
2.1.5	Preliminary concepts on Green's functions	14
2.2	Generalities regarding Green's Functions	15
2.3	Green's Function in Boundary Value Problems (BVPs)	15
2.3.1	Finding the expression for the Green's function using the Method of Variation of Parameters	15
2.3.2	Formulation the solution of a BVP using the Green's function	16
2.4	Properties of the Green's Function	16
3	The scientific machine learning framework	18
3.1	Introduction	18
3.2	Preliminary aspects of machine-learning	18
3.2.1	Definition	18
3.2.2	Examples of Machine Learning Problems	18
3.3	Machine Learning Approaches	19
3.4	Notions of Supervised Learning	19
3.4.1	Training with Labeled Data	19
3.4.2	Model Evaluation	20
3.5	Deep Learning and neural networks	20
3.5.1	About the Multilayer Perceptron	20
3.5.2	Activation Function σ of a Neuron	22

3.6	Physics-Informed Machine Learning	23
3.6.1	Introduction	23
3.6.2	Physics-Informed Neural Networks (PINNs)	23
3.6.3	Detailed Overview of the Physical Loss in PINNs	23
3.6.4	Training PINNs	25
3.6.5	Advantages of PINNs	25
3.6.6	Challenges with PINNs	25
3.7	Neural Operators	25
3.8	Deep Green Operators and Their Problem Setting	26
3.8.1	Posing the Problem	26
3.8.2	Objective	26
3.8.3	Kernel Formulation and Neural Operator Theory	27
3.9	Conclusion	28
4	Data sampling, architecture and implementation	29
4.1	Introduction	29
4.2	The Scimba library	29
4.3	Data sampling	31
4.3.1	The numerical solvers used to obtain the true target solutions	31
4.4	The architecture of the neural network	32
4.4.1	The preliminary notion of rational neural networks	32
4.4.2	The architecture of the neural network used in the implementation	33
4.5	Data generation and training	34
4.6	The loss function of the architecture	35
4.6.1	Loss Function for the Green Kernel	35
4.6.2	Loss Function for the Homogeneous Solution	35
4.6.3	Overall Loss Function	36
4.7	History of the implementation throughout the internship	36
4.7.1	Introduction	36
4.7.2	Initial Phase: Understanding the Problem and Setting Up the Environment	36
4.7.3	Experimentation and Optimization of Neural Operators	37
4.7.4	Transition to a Data-Driven Approach	38
4.7.5	Redesign of the training data by relying on Gaussian Processes	38
4.7.6	Advancing the Training with Full Data and Addressing Early Convergence Issues	39
4.7.7	Transition to More Difficult Problems	40
4.7.8	Final Refinements and Report Writing	41
4.8	Conclusion	42
5	Final Numerical Results and Analysis	43
5.1	Introduction	43
5.2	Highlighting the theoretical expectations of the Monte Carlo convolution	43
5.2.1	Introduction	43
5.2.2	The test case used	44
5.2.3	Results and Analysis	44
5.3	Plots of Gaussian Processes and Solutions	46
5.3.1	Introduction	46
5.3.2	The configuration and the test cases involved	46

5.3.3	Results and Analysis	46
5.4	Order of Convergence of the Finite Difference Solver	48
5.4.1	Introduction	48
5.4.2	The test case used	49
5.4.3	Results and Analysis	49
5.5	Final Results for the Laplace Equation	51
5.5.1	Introduction	51
5.5.2	The configuration and the test cases involved	51
5.5.3	Results and Analysis	52
5.5.4	Getting the predicted solution on several source terms	52
5.5.5	The convergence of the loss function	53
5.6	Final Results for the Advection-Diffusion Equation	54
5.6.1	Introduction	54
5.6.2	The configuration and the test cases involved	55
5.6.3	Results and Analysis	55
5.6.4	The homogeneous solution	56
5.6.5	Getting the predicted solution on several source terms	60
5.6.6	The convergence of the loss function	61
5.7	Final Results for the Helmholtz Equation with a fixed wavenumber	62
5.7.1	Introduction	62
5.7.2	The configuration and the test cases involved	62
5.7.3	Results and Analysis	62
5.7.4	Getting the predicted solution on several source terms	63
5.8	Final results for the Helmholtz equation with a varying wavenumber	65
5.8.1	Introduction	65
5.8.2	The configuration and the test cases involved	66
5.8.3	Results and Analysis	66
5.8.4	Getting the predicted solution on several source terms	67
5.9	Performance Comparison: Neural Network vs Finite Differences	69
5.9.1	Introduction	69
5.9.2	Neural Network Approach	70
5.9.3	Finite Differences	70
5.9.4	Results and Analysis	70
5.10	Unsatisfactory Results for the Parametric Helmholtz Equation in the Vicinity of the Eigenmodes	71
5.10.1	The configuration and the test cases involved	71
5.10.2	Results and Analysis	71
6	Conclusion	74
A	The theoretical framework for Boundary Value Problems	77
A.1	Introduction	77
A.2	Sturm-Liouville Operator	77
B	About Gaussian Processes	79
B.1	Introduction to Gaussian Processes	79
B.2	Mathematical Definitions related to Gaussian Processes	79
B.3	RBF Kernel	80

C	Exact Green's Functions	81
C.1	Finding the Expression for the Green's Function Using the Method of Variation of Parameters	81
C.2	Green's Function for the Laplace Equation	81
C.2.1	Step 1: Finding u_1 and u_2	81
C.2.2	Step 2: Sturm-Liouville Equation and $p(x)$	81
C.2.3	Step 3: Finding the Wronskian	82
C.2.4	Step 4: Applying the Green's Function Formula	82
C.2.5	Step 5: Verifying the Equation	82
C.2.6	Step 6: Checking the Boundary Conditions	83
C.3	Green's Function for the Helmholtz Equation	83
C.3.1	Step 1: Finding u_1 and u_2	83
C.3.2	Step 2: Sturm-Liouville Equation and $p(x)$	83
C.3.3	Step 3: Finding the Wronskian	84
C.3.4	Step 4: Applying the Green's Function Formula	84
C.3.5	Step 5: Verifying the Equation	84
C.3.6	Step 6: Checking the Boundary Conditions	85
D	Table of the several source terms used for the numerical results	86
E	Other numerical results	87
E.1	Some miscallenous results for the Laplace equation	87
E.1.1	Getting the predicted solution on a specific source term	87
E.2	Some miscallenous results for the Advection-Diffusion equation	88
E.2.1	About the test case used	88
E.2.2	The homogeneous solution	88
E.2.3	Getting the predicted solution on a specific source term	92
E.3	Some miscallenous results for the Helmholtz equation with $k = 8$	93
E.3.1	Getting the predicted solution on a specific source term	93
E.4	Some miscallenous results for the parameterized Helmholtz equation	94
E.4.1	Getting the predicted solution on a specific source term	94

Chapter 1

Introduction

1.1 Preliminary information

1.1.1 Acknowledgements

I am deeply grateful to my supervisors, Dr. Emmanuel Franck, Dr. Victor Michel-Dansac, and Dr. Antoine Deleforge, for their invaluable guidance, support, and teachings throughout this project. Their profound knowledge and continuous encouragement have been essential in navigating the complexities of this research. I sincerely appreciate their patience and unwavering availability, which have been crucial in overcoming numerous challenges. This opportunity to work on a project that addresses key questions about my future career has been immensely rewarding, and I am thankful for their role in making it possible.

1.1.2 Preliminary remarks

About the report This report details my Master’s thesis project conducted in the Master of Applied Mathematics and Computational Sciences (Master CSMI) program at the University of Strasbourg. The project ran from February 19, 2024, to July 31, 2024. This thesis expands on the research conducted during my third-semester project, which allowed me to become acquainted with scientific machine learning (SciML) and Scimba. Many elements in the subsequent sections are taken from this prior research.

About the internship Building on my project from the third semester, this thesis delves deeper into scientific machine learning, with a specific emphasis on the Helmholtz partial differential equation. The aim is to develop a streamlined method for different shapes and reverse issues using the Green Neural Operator (Green Neural Operators) platform. The research incorporates physics-informed neural networks (PINNs) to predict Green functions. A new library called Scimba, based on PyTorch and aimed at scientific machine learning, is still under development by the Inria research team Macaron, which will be introduced below. It focuses on utilizing Multi-Layer Perceptrons (MLPs) and Fourier Nets in oscillatory test cases. The results show both the success and difficulties of the Green Neural Operators technique for addressing parameterized oscillatory PDEs.

Presentation of the research center

The research center: *Inria* Established in 1967, Inria, officially known as the National Institute for Research in Computer Science and Automation, is a public research institute in France.

Inria is dedicated to advancing the fields of computer science and applied mathematics. With a primary goal of driving progress in digital sciences and technologies, Inria employs over 3,000 researchers and engineers across numerous research centers and laboratories throughout France. Their work significantly contributes to the development of innovative solutions and the advancement of knowledge in these critical areas.

My team and my supervisor Researcher Emmanuel Franck supervised my internship, while researchers Victor Michel-Dansac, and Antoine Deleforge were my co-supervisors. All three are members of the MACARON team at Inria. They are interested in relying on deep-learning technique to numerically solve PDEs (especially of nonlinear variety) with theoretical guarantees.

1.2 Context of the project

1.2.1 General context

The field of scientific machine learning is expanding at the intersection of applied mathematics, physics, scientific computing, and machine learning. The goal is to create customized machine learning techniques for scientific issues, directed by physical expertise like physics-based modeling and simulation. This includes the creation of loss functions based on physics, the incorporation of physics-based structures in selecting particular neural network architectures, and attempts to blend physics and machine learning to improve the comprehension of opaque models.

This area is especially intriguing because it provides more cost-effective alternatives for solving PDEs compared to traditional numerical techniques like finite differences, finite elements, and finite volumes. It can also be applied for reverse-engineering PDEs, determining the PDE parameters based on observed data. This method is helpful for addressing inverse problems, which are frequently ill-posed, indicating they lack a singular solution or their solution is not reliable in relation to the data. Furthermore, scientific machine learning holds potential in identifying the governing equations of unformalized dynamical systems.

Neural operators act as surrogate models by approximating complex, computationally expensive simulations through machine learning techniques. Once trained on data from the original, high-fidelity model, they can rapidly predict outcomes for new inputs, offering a balance between computational efficiency and accuracy. This potentially makes them particularly valuable in tasks like optimization, sensitivity analysis, or any scenario requiring numerous simulations, where they provide a practical solution for fast and repeated evaluations without the need for costly computations.

1.2.2 Specific context

My research centered on the utilization of neural operators, specifically Green Neural Operators (Green Neural Operatorss), to understand the Green function. This function acts as the convolution kernel for acquiring solutions to provided force terms in different PDEs. One of the main equations examined was the Helmholtz equation, a time-independent linear PDE obtained from the Wave equation using the separation-of-variable technique. Alternatively, this equation can also be obtained by applying the Fourier Transform on the Wave equation. This equation holds importance in various areas of physics, such as electromagnetism and acoustics.

1.3 Objectives of the project

1.3.1 Academical objectives

The University of Strasbourg's Master CSMI program involves various required courses that students must complete in order to graduate. One of these opportunities is a major internship project usually completed in the last semester.

The primary objective of this internship is to enable students to apply their expertise in a professional setting, particularly in engineering and mathematical research. This internship offers hands-on experience in addressing real-world issues and complements the academic studies.

The internship involves working with at least one supervisor to help students solve problems that encompass various mathematical and computational components. The internship requirements include a final report, a final presentation, and a portfolio that outlines all the completed tasks.

The final report needs to provide comprehensive details about the project and its context, an analysis of the data used, and a theoretical explanation of the mathematical and computational models employed. It should also present the findings from the experiments conducted during the project, including measurements, uncertainties, and any observed effects. The report concludes with a summary and a forward-looking perspective on potential future enhancements.

1.3.2 Scientific objectives

The primary goals of my master's internship, which centers on utilizing scientific machine learning (SciML) to investigate the Helmholtz equation, are outlined below:

1. **Exploring the Foundational Requirements for Theoretical Study:** Acquire a comprehensive understanding of SciML, Green functions, and deep learning techniques pertinent to the project. A solid grasp of these foundational concepts is crucial for effectively applying them to the Helmholtz equation.
2. **Developing the Neural Network for the Kernel:** Create and implement a neural network specifically tailored for the kernel of the Green function. This involves designing the architecture, training the network, and validating its performance in approximating the Green function.
3. **Creating the Monte Carlo (MC) Sublayer:** Develop a Monte Carlo sublayer for accurately estimating integrals within the neural operator framework. Ensure the MC sublayer is well-integrated into the neural network structure and operates efficiently within the solver.
4. **Implementing a Solver for Various PDE Test Cases Using Neural Operators:**
 - (a) **Basic Test Scenario:** Develop a solver for a one-dimensional Laplace equation with homogeneous boundary conditions, no parameters, and no oscillations in the spatial domain of $[0,1]$. This serves as a fundamental test case to validate the basic functionality of the solver.
 - (b) **Second Test Case:** Implement a solver for an Advection-Diffusion equation with inhomogeneous boundary conditions in the spatial domain of $[0,1]$. This involves using a neural network to model the homogeneous solution while accounting for the inhomogeneous boundary conditions (local sublayer).

- (c) **Third Test Case:** Enhance the solver for the Advection-Diffusion equation to accommodate varying parameters within the $[0,1]$ spatial domain.
- (d) **Fourth Test Case:** Develop a solver for the one-dimensional Helmholtz equation with homogeneous boundary conditions and oscillatory characteristics within the $[0,1]$ spatial domain, keeping the parameters constant.
- (e) **Fifth Test Case:** Conclude with a comprehensive solver for the one-dimensional Helmholtz equation with homogeneous boundary conditions, oscillatory behavior, and varying parameters in the $[0,1]$ spatial domain.

These goals aim to leverage advanced machine learning methods to develop efficient solvers for the Helmholtz equation and other PDEs, thereby contributing to the theoretical understanding of solving PDEs and its practical applications. Overall, the scientific objectives of this project are to deepen our knowledge of scientific machine learning and deep learning, and to explore cost-effective approaches to studying the Helmholtz equation compared to traditional numerical methods.

Specific (technical and investigative) objectives

The more specific goals are about implementing a Python code relying on the Scimba library to solve the Helmholtz equation. The code should be able to solve the Helmholtz equation on a given domain, with a given source term and a given set of physical parameters. The code should also be able to compute the errors between the exact solution and the numerical solution, and to plot the results of the learning process.

Besides those aspects, the specific objectives of my project will be to optimize and debug the code; and learning solutions to improve the overall comprehension and utilization of the Scimba library.

My code will be implemented using Python files.

1.4 Estimation of the Time Needed

1. Analyzing the Theoretical Basis of the Project:

- (a) Estimated: 4 weeks
- (b) Actual: 2.5 weeks
- (c) During this phase, I delved into the fundamental concepts of scientific machine learning (SciML), Green functions, and deep learning techniques. This also included learning about the Scimba library and its various dependencies.

2. Installing and Mastering the Scimba Library:

- (a) Estimated: 1 week
- (b) Actual: 1 week
- (c) Setting up the Scimba library was relatively straightforward. However, I needed to ensure I could use it effectively, familiarize myself with the Python virtual environment, and install the necessary dependencies.

3. Developing the Neural Network Code for the Kernel:

- (a) Estimated: 2 weeks

- (b) Actual: 2 weeks
- (c) Creating and implementing a neural network for the core of the Green function involved designing the architecture, training the model, and validating its performance. This task required significant attention and precision.

4. **Creating the Monte Carlo (MC) Sublayer:**

- (a) Estimated: 3 weeks
- (b) Actual: 2 weeks
- (c) Developing a Monte Carlo sublayer to accurately approximate integrals within the neural operator framework proved challenging due to the complexity of the code and the limited time available to fully understand neural operators.

5. **Developing Neural Operator-Based Solvers for Various PDE Test Cases:**

- (a) **Basic Test Scenario (one-dimensional Laplace equation with homogeneous boundary conditions, no parameters, no oscillations):**
 - i. Estimated: 4 weeks
 - ii. Actual: 6 weeks
 - iii. This foundational test case was crucial for verifying the basic functionality of the solver. Ensuring everything worked correctly and dealing with the initial complexity of Scimba made it particularly challenging. This stage uncovered several intricate issues that needed to be resolved. It was the most complex aspect of my project.
- (b) **Second Test Scenario (Advection-Diffusion equation with inhomogeneous boundary conditions):**
 - i. Estimated: 2 weeks
 - ii. Actual: 2 weeks
 - iii. This involved using a neural network to model the homogeneous solution while considering the inhomogeneous boundary conditions.
- (c) **Third Test Scenario (Advection-Diffusion equation with varying parameters):**
 - i. Estimated: 2 weeks
 - ii. Actual: 2 weeks
 - iii. Enhancing the solver to handle varying parameters within the spatial domain was a critical step forward.
- (d) **Fourth Test Scenario (one-dimensional Helmholtz equation with homogeneous boundary conditions, oscillatory nature):**
 - i. Estimated: 2 weeks
 - ii. Actual: 2 weeks
 - iii. Developing a solver for the Helmholtz equation required addressing its oscillatory characteristics.
- (e) **Fifth Test Scenario (one-dimensional Helmholtz equation with homogeneous boundary conditions, oscillatory nature, varying parameters):**
 - i. Estimated: 2 weeks

- ii. Actual: 2 weeks
- iii. Completing an all-encompassing solver for the Helmholtz equation guaranteed its ability to smoothly manage a variety of boundary conditions and parameters.

Chapter 2

Preliminary mathematical notions

2.1 Basic notions of partial differential equations (PDEs)

2.1.1 Introduction to Differential Equations

Differential equations [1] express how a function evolves in relation to its derivatives, illustrating the function's response to changes in one or multiple variables. Ordinary differential equations (ODE) refer to the case when the function depends on a single variable. Conversely, when the function depends on multiple independent variables, it is described by a partial differential equation (PDE). The highest derivative contained within a differential equation determines its order, indicating the intricacy and extent of the relationship it represents.

With their wide range of physical phenomena and theoretical constructs, differential equations are essential tools in both applied and pure mathematics. Distinguishing between ODEs and PDEs is essential for guiding solution methods and result interpretation in different scientific and engineering situations.

Example 1.

$$\frac{d^4u}{dx^4} + \frac{d^2u}{dx^2} + u^2 = \cos x$$

is a fourth-order ODE, while:

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} - u$$

is a second-order PDE.

In my internship test case, all the test cases were stationary one-dimensional PDEs, which means they can be comprehended as ODEs.

2.1.2 Boundary and Initial Conditions

Solutions to PDEs are often specified by initial conditions (for dynamical systems) or boundary conditions (for equilibrium problems).

In my internship test cases, boundary conditions were involved and more especially the Dirichlet boundary conditions. The homogeneous Dirichlet boundary conditions were used for the Laplace and Helmholtz equations on the interval $[0, 1]$, while heterogeneous Dirichlet boundary conditions were applied to the Advection-Diffusion equation on the same domain as before.

Definition 1 (Dirichlet Boundary Condition). *The Dirichlet boundary condition specifies the value of the function at the boundary for a one-dimensional ordinary differential equation (ODE). For a function u defined on the interval $[a, b]$, the Dirichlet boundary condition is given by:*

$$u(a) = \alpha, \quad u(b) = \beta \quad (2.1.1)$$

where α and β are known constants.

A special case of the Dirichlet boundary condition is the homogeneous Dirichlet boundary condition, where the function is set to zero at the boundaries:

$$u(a) = 0, \quad u(b) = 0 \quad (2.1.2)$$

2.1.3 About elliptic PDEs

We will exclusively focus on one-dimensional elliptic PDEs throughout this internship.

They are a sub-category of second-order order linear PDEs, which means that they permit the application of powerful analytical tools such as the Superposition Principle and Green's functions.

Elliptic PDEs can also be subjected to the Lax-Milgram theorem, which guarantees the existence and uniqueness of solutions to the associated variational problem. Such a theorem will help formulate the problem related to the problem setting of the Green neural operator, in section 3.8.1.

Prerequisites on linear PDEs

Definition 2. *A linear PDE can be expressed in the form $\mathcal{L}[u] = f$, where $\mathcal{L}$ is a linear differential operator, u is the unknown function, and f is a given source term. The source term f represents external influences or inputs to the system described by the PDE.*

Example 2. *Consider a linear differential operator $\mathcal{L}$ expressed as a linear combination of $\frac{d^2}{dx^2}$, $\frac{d}{dx}$, and the identity operator Id .*

During my internship, all the studied equations were second-order linear ordinary differential equations (ODEs), which can be represented using this operator. For example:

$$\mathcal{L} = a_2 \frac{d^2}{dx^2} + a_1 \frac{d}{dx} + a_0 Id$$

where a_2, a_1 , and a_0 are constants. For any sufficiently smooth function u , the operator $\mathcal{L}$ acts as:

$$\mathcal{L}[u] = a_2 \frac{d^2 u}{dx^2} + a_1 \frac{du}{dx} + a_0 u$$

A specific variant of this operator is used in the Helmholtz equation. For the Helmholtz equation, the operator $\mathcal{L}$ is given by:

$$\mathcal{L} = \frac{d^2}{dx^2} + k^2 Id$$

where k is a constant representing the wave number. Acting on a sufficiently smooth function u , the operator $\mathcal{L}$ becomes:

$$\mathcal{L}[u] = \frac{d^2 u}{dx^2} + k^2 u.$$

Proposition 1. *The superposition principle asserts that if u_1 and u_2 are solutions to the linear partial differential equation (PDE) $\mathcal{L}[u] = f$ with corresponding source terms f_1 and f_2 , respectively, then any linear combination $c_1 u_1 + c_2 u_2$, where c_1 and c_2 are constants, is also a solution to the PDE with the source term $c_1 f_1 + c_2 f_2$. Mathematically, if $\mathcal{L}[u_1] = f_1$ and $\mathcal{L}[u_2] = f_2$, then $\mathcal{L}[c_1 u_1 + c_2 u_2] = c_1 f_1 + c_2 f_2$. This principle is pivotal in the analysis of linear systems, as it facilitates the decomposition of complex problems into simpler, more tractable components.*

Elliptic PDEs

Definition 3. A second-order linear partial differential equation (PDE) can generally be written as:

$$\sum_{i,j=1}^n a_{ij}(\mathbf{x}) \frac{\partial^2 u}{\partial x_i \partial x_j} + \sum_{i=1}^n b_i(\mathbf{x}) \frac{\partial u}{\partial x_i} + c(\mathbf{x})u = f(\mathbf{x}), \quad \mathbf{x} \in U \subset \mathbb{R}^n,$$

where $u : U \subset \mathbb{R}^n \rightarrow \mathbb{R}$ is the unknown function, $a_{ij}(\mathbf{x})$, $b_i(\mathbf{x})$, and $c(\mathbf{x})$ are given coefficient functions, and $f(\mathbf{x})$ is a source term.

The PDE is classified as elliptic [2] at a point $\mathbf{x} \in U$ if the symmetric matrix $\mathbf{A}(\mathbf{x}) = (a_{ij}(\mathbf{x}))_{1 \leq i,j \leq n}$, formed by the coefficients of the second-order derivatives, is positive definite. This means that all eigenvalues of $\mathbf{A}(\mathbf{x})$ are positive (or all negative), ensuring the matrix has eigenvalues of the same sign. Mathematically, this condition can be stated as:

$$\forall \xi \in \mathbb{R}^n \setminus \{0\}, \quad \xi^\top \mathbf{A}(\mathbf{x}) \xi > 0.$$

The Lax-Milgram Theorem

Theorem 2.1.1 (Lax-Milgram). Let H be a Hilbert space with an inner product $\langle \cdot, \cdot \rangle_H$ and associated norm $\| \cdot \|_H$. Let $a(\cdot, \cdot)$ be a bilinear form on $H \times H$ that satisfies the following conditions:

1. **Bilinearity:** The form $a(u, v)$ is linear in both arguments, i.e., for all $u, v, w \in H$ and scalars $\alpha, \beta \in \mathbb{R}$ (or $\mathbb{C}$ if H is complex):

$$a(\alpha u + \beta v, w) = \alpha a(u, w) + \beta a(v, w),$$

and

$$a(u, \alpha v + \beta w) = \alpha a(u, v) + \beta a(u, w).$$

2. **Coercivity:** There exists a constant $\alpha > 0$ such that for all $u \in H$:

$$a(u, u) \geq \alpha \|u\|_H^2.$$

3. **Continuity:** There exists a constant $\beta > 0$ such that for all $u, v \in H$:

$$|a(u, v)| \leq \beta \|u\|_H \|v\|_H.$$

Let f be a continuous linear functional on H , i.e., $f \in H'$, where H' is the dual space of H . Then, there exists a unique element $u \in H$ such that:

$$a(u, v) = f(v) \quad \text{for all } v \in H.$$

2.1.4 The equations involved in the internship

We will examine three essential linear second-order PDEs throughout this internship.

1. The Laplace (or Poisson) equation, which models potential fields in mechanical and thermal equilibria. The equation is given by:

$$\Delta u = f \quad \text{in } [0, 1]$$

with boundary conditions:

$$u(0) = 0, \quad u(1) = 0$$

2. The Advection-Diffusion equation, which models processes involving the transport and diffusion of substances. The equation is given by:

$$Du'' + 2u' + u = f \quad \text{in } [0, 1]$$

with boundary conditions:

$$u(0) = 1, \quad u(1) = -2$$

3. The Helmholtz equation, which arises in problems of wave propagation and vibration analysis. The equation is given by:

$$\Delta u + k^2 u = f \quad \text{in } [0, 1]$$

with boundary conditions:

$$u(0) = 0, \quad u(1) = 0$$

2.1.5 Preliminary concepts on Green's functions

The Delta Function

Definition 4. The Dirac delta function, $\delta_\xi(x)$, is a generalized function defined by the following properties.

Firstly, the **localization** property states that:

$$\delta_\xi(x) = 0 \quad \text{for } x \neq \xi.$$

Secondly, the **normalization** property is given by:

$$\int_a^b \delta_\xi(x) dx = 1, \quad \text{for } a < \xi < b.$$

Lastly, the **sifting property** asserts that for any continuous function $f(x)$:

$$\int_a^b f(x) \delta_\xi(x) dx = f(\xi), \quad \text{for } a < \xi < b.$$

Paving the Way for Green's Functions Understanding and applying these properties will be instrumental in our study of PDEs through the method of Green's functions.

Generalized functions are crucial in the formulation of Green's functions, which are used to solve inhomogeneous linear PDEs. The Green's function $G(x, \xi)$ for a differential operator $\mathcal{L}$ is defined such that:

$$\mathcal{L}[G(x, \xi)] = \delta_0(x - \xi), \tag{2.1.3}$$

where $\delta_0(x - \xi)$ is the Dirac delta function centered at ξ .

This relationship allows the solution of the PDE $\mathcal{L}[u] = f$ to be expressed as an integral involving the Green's function and the source term f :

$$u(x) = \int_{\Omega} G(x, \xi) f(\xi) d\xi \tag{2.1.4}$$

Thus, generalized functions provide a powerful tool for addressing boundary value problems by enabling the construction of Green's functions, which in turn facilitate the solution of PDEs. More details on Green's functions will be provided in subsequent sections.

2.2 Generalities regarding Green's Functions

In this section, we delve into the solution of nonhomogeneous differential equations by employing Green's functions [5]. Our objective is to resolve the nonhomogeneous differential equation

$$\mathcal{L}[u] = f, \quad (2.2.1)$$

where $\mathcal{L}$ is a differential operator. The solution can be formally expressed as

$$u = \mathcal{L}^{-1}[f] \quad (2.2.2)$$

The inverse of a differential operator manifests as an integral operator, which we aim to represent in the form

$$u = \int G(x, \xi) f(\xi) d\xi. \quad (2.2.3)$$

The function $G(x, \xi)$, known as the Green's function, serves as the kernel of the integral operator.

Another way of introducing the Green's function is by considering the solution of a given linear differential equation with a specified source term. Formally, the Green's function $G(x, \xi)$ is defined as the solution to the differential equation with a Dirac delta function $\delta(x - \xi)$ as the source term:

As explained above (see paragraph 2.1.5), the Green's function can also be defined as the solution to the differential equation with a Dirac delta function as the source term. This is expressed as:

$$\mathcal{L}[G(x, \xi)] = \delta_0(x - \xi),$$

where $\mathcal{L}$ is the linear differential operator. This formulation is pivotal in solving inhomogeneous differential equations, as it allows the solution $u(x)$ to be expressed as a convolution of the Green's function $G(x, \xi)$ with the source term $f(\xi)$, as written above.

This approach leverages the properties of the Dirac delta function to decompose the problem into more manageable parts, facilitating the solution of complex differential equations.

2.3 Green's Function in Boundary Value Problems (BVPs)

2.3.1 Finding the expression for the Green's function using the Method of Variation of Parameters

Assuming that the boundary value problem (BVP) can be formulated using the Sturm-Liouville operator (as introduced in section A.2), we now turn our attention to boundary value problems, specifically focusing on the following problem:

$$\frac{d}{dx} \left(p(x) \frac{du}{dx} \right) + q(x)u(x) = f(x), \quad a < x < b,$$

subject to the boundary conditions:

$$u(a) = 0, \quad u(b) = 0.$$

The solution takes the form:

$$u(x) = \int_a^b G(x, \xi) f(\xi) d\xi,$$

where $G(x, \xi)$ is the Green's function for the differential operator $\mathcal{L}$ associated with the BVP. The Green's function is defined as:

For such boundary value problems, the Green function can be expressed as:

$$G(x, \xi) = \begin{cases} \frac{u_1(\xi)u_2(x)}{p(\xi)W(\xi)}, & a \leq \xi \leq x, \\ \frac{u_1(x)u_2(\xi)}{p(\xi)W(\xi)}, & x \leq \xi \leq b, \end{cases} \quad (2.3.1)$$

where $u_1(x)$ and $u_2(x)$ are solutions of the homogeneous problem satisfying $u_1(a) = 0, u_2(b) = 0$ and $u_1(b) \neq 0, u_2(a) \neq 0$.

and $W(\xi)$ is the Wronskian of $u_1(x)$ and $u_2(x)$, defined as:

$$W(u_1, u_2)(\xi) = \begin{vmatrix} u_1(\xi) & u_2(\xi) \\ u_1'(\xi) & u_2'(\xi) \end{vmatrix}. \quad (2.3.2)$$

2.3.2 Formulation the solution of a BVP using the Green's function

Given a boundary value problem with inhomogeneous boundary conditions, we can express the solution as the sum of the homogeneous solution (satisfying the inherent boundary conditions). The solution to the BVP is then given by the convolution of the Green's function with the source term.

Given the BVP

$$\begin{aligned} \mathcal{L}[u] &= f(x), & a < x < b, \\ u(a) &= \alpha, \\ u(b) &= \beta \end{aligned}$$

where $\mathcal{L}$ is a linear differential operator.

Provided that $u_0(x)$ is the solution to the homogeneous problem with the same boundary conditions, the solution to the BVP can be expressed as:

$$u(x) = u_0(x) + \int_a^b G(x, \xi) f(\xi) d\xi \quad (2.3.3)$$

where $G(x, \xi)$ is the Green's function for the differential operator $\mathcal{L}$ associated with the following BVP:

$$\begin{aligned} \mathcal{L}[G(x, \xi)] &= \delta(x - \xi), & a < x, \xi < b, \\ G(a, \xi) &= G(b, \xi) = 0 \end{aligned}$$

2.4 Properties of the Green's Function

In this section, we will discuss the properties and solutions of the differential equation.

$$\frac{\partial}{\partial x} \left(p(x) \frac{\partial G(x, \xi)}{\partial x} \right) + q(x) G(x, \xi) = 0, \quad x \neq \xi \quad (2.4.1)$$

For $x < \xi$, we are on the second branch and $G(x, \xi)$ is proportional to $y_1(x)$. Hence, since $y_1(x)$ is a solution of the homogeneous equation, so is $G(x, \xi)$. For $x > \xi$, we are on the first branch and $G(x, \xi)$ is proportional to $y_2(x)$. Thus, $G(x, \xi)$ is a solution of the homogeneous problem.

Proposition 2 (Differential Equation). *The boundary value Green's function satisfies the differential equation:*

$$\frac{\partial}{\partial x} \left(p(x) \frac{\partial G(x, \xi)}{\partial x} \right) + q(x) G(x, \xi) = 0, \quad x \neq \xi.$$

Proof. This can be established by noting that for $x < \xi$, $G(x, \xi)$ is proportional to $y_1(x)$, which is a solution of the homogeneous equation. Similarly, for $x > \xi$, $G(x, \xi)$ is proportional to $y_2(x)$, also a solution of the homogeneous problem. $\square$

Proposition 3 (Boundary Conditions). *The Green's function satisfies the boundary conditions:*

$$G(a, \xi) = 0 \quad \text{and} \quad G(b, \xi) = 0.$$

Proof. Indeed, at $x = a$, $G(x, \xi)$ is proportional to $y_1(x)$, and since $y_1(x)$ satisfies the boundary condition, so does $G(x, \xi)$. The same argument applies for $x = b$. $\square$

Proposition 4 (Symmetry or Reciprocity). *The Green's function satisfies the reciprocity property:*

$$G(x, \xi) = G(\xi, x).$$

Proof. The symmetry of the Green's function simply stems from the integral representation of the Green's function. $\square$

Proposition 5 (Continuity of G at $x = \xi$). *The Green's function is continuous at $x = \xi$, i.e.,*

$$G(\xi^+, \xi) = G(\xi^-, \xi),$$

where ξ^+ and ξ^- are defined as:

$$G(\xi^+, x) = \lim_{x \downarrow \xi} G(x, \xi), \quad x > \xi,$$

$$G(\xi^-, x) = \lim_{x \uparrow \xi} G(x, \xi), \quad x < \xi.$$

Proof. Since the Wronskian W is continuous, the continuity of $G(x, \xi)$ between the two branches at $x = \xi$ is established. $\square$

Proposition 6 (Jump Discontinuity of $\frac{\partial G}{\partial x}$ at $x = \xi$). *The derivative of the Green's function with respect to x has a jump discontinuity at $x = \xi$ given by:*

$$\frac{\partial G(\xi^+, \xi)}{\partial x} - \frac{\partial G(\xi^-, \xi)}{\partial x} = \frac{1}{p(\xi)}.$$

Proof. This result can be derived by calculating the derivatives from each branch and subtracting them:

$$\frac{\partial G(\xi^+, \xi)}{\partial x} - \frac{\partial G(\xi^-, \xi)}{\partial x} = -\frac{1}{pW} (y_1'(\xi)y_2(\xi) - y_1(\xi)y_2'(\xi)) = \frac{1}{p(\xi)}.$$

$\square$

Chapter 3

The scientific machine learning framework

The previous sections have detailed the mathematical foundations and properties of Green's functions, which are essential tools for solving differential equations. These traditional methods provide a rigorous framework for understanding and solving physical problems. However, with the advent of modern computational techniques, there is an increasing interest in leveraging data-driven approaches to complement and enhance these classical methods. This brings us to the field of scientific machine learning (SciML).

3.1 Introduction

Scientific machine learning (SciML) combines scientific computing with machine learning to address complex challenges in physics, engineering, and other domains. By enhancing traditional methods, such as solving PDEs to obtain Green's functions (that were indeed evoked above, in section 2.2), SciML aspires to achieve greater efficiency and accuracy. However, as a nascent and experimental field, its effectiveness remains under exploration, with varying outcomes. Continued research is crucial to fully realize and understand the potential applications of SciML.

3.2 Preliminary aspects of machine-learning

3.2.1 Definition

Machine learning (ML) [6] involves using mathematical and statistical methods in artificial intelligence to allow computers to enhance their performance in tasks by learning from data, rather than being specifically programmed for each task. In a general sense, it includes creating, examining, refining, building, and putting into action these techniques.

3.2.2 Examples of Machine Learning Problems

Examples of machine learning problems include identifying groups of newspaper articles on the same topic without any prior examples or known topics. Such a problem is typically about unsupervised learning. In contrast, automatically classifying new images as either "dog" or "cat" based on a set of labeled images is a straightforward instance of supervised learning. Similarly, predicting the heating bill of an apartment using features such as size, neighborhood, floor, price,

construction year, and number of occupants is another case of supervised learning. These examples illustrate the diverse applications of machine learning techniques in handling different types of data and tasks.

It is crucial to emphasize the role of data in these examples. For instance, while predicting the heating bill of an apartment could theoretically be approached using pure PDE models (such as heat equations), the machine learning approach leverages large datasets to identify patterns and make predictions without explicitly modeling the underlying physical processes. This data-driven approach can often provide more flexible and scalable solutions, especially when dealing with complex and high-dimensional data. By utilizing vast amounts of data, machine learning models can uncover intricate relationships and dependencies that might be challenging to capture with traditional methods alone.

3.3 Machine Learning Approaches

Machine learning approaches include supervised learning, unsupervised learning, and reinforcement learning.

Definition 5 (Supervised Learning). *This approach involves training a model on a labeled dataset, where the input data is paired with the correct output. The model learns to predict the output from the input data.*

Definition 6 (Unsupervised Learning). *This approach deals with unlabeled data. The model tries to identify patterns and relationships within the data without any explicit instructions on what to look for.*

Definition 7 (Reinforcement Learning). *This approach involves training a model to make a sequence of decisions by rewarding it for good decisions and penalizing it for bad ones. The model learns to maximize the cumulative reward over time.*

In our case, we only relied on supervised learning to train the neural operator to learn the Green’s function.

3.4 Notions of Supervised Learning

3.4.1 Training with Labeled Data

Definition 8 (Training with Labeled Data). *The essence of supervised learning lies in its use of labeled data. During the training phase, the algorithm is fed a dataset where each input is associated with a known output. This process allows the model to learn and map the inputs to their respective outputs accurately.*

Mathematically, given a training dataset $\{(x_i, y_i)\}_{i=1}^n$, where $x_i \in \mathbb{R}^d$ represents the input features and $y_i \in \mathbb{R}$ or $\mathbb{Z}$ represents the corresponding label, the goal is to learn a function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ or $f : \mathbb{R}^d \rightarrow \mathbb{Z}$ that accurately maps inputs to outputs.

Label data has been crucial in the training of the neural operator to learn the Green’s function. In that case, the input were samples of the input space (the space domain and the parameter space) and samples of the source terms (using Gaussian processes, which will be explained later) and the output were the corresponding solutions of the differential equations obtained with a boundary value problem solver.

3.4.2 Model Evaluation

Definition 9 (Mean Squared Error (MSE)). *The Mean Squared Error (MSE) is a metric that quantifies the average squared difference between the predicted values and the actual values. It provides a measure of the model's accuracy in predicting continuous values. Formally, the MSE is defined as:*

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

where y_i represents the actual values, $\hat{y}_i$ denotes the predicted values, and n is the number of observations.

In our problem, especially in the data-driven approach, the MSE was used to formulate the training loss of the neural operator. The neural operator was trained to minimize the MSE between the predicted solution and the true solution of the differential equation.

3.5 Deep Learning and neural networks

Deep learning is a subset of machine learning that focuses on neural networks with many layers (deep neural networks). These models are capable of learning hierarchical representations of data, which makes them particularly effective for tasks such as image recognition, natural language processing, and complex game playing.

Neural networks are computational models inspired by the human brain. They consist of layers of interconnected nodes (neurons), where each connection has an associated weight. Neural networks learn by adjusting these weights based on the error of their predictions, typically using a method called backpropagation. This iterative process involves propagating the error backward through the network and updating the weights to minimize this error.

To delve deeper into the architecture of neural networks, we begin with one of the most fundamental types: the Multilayer Perceptron (MLP). The MLP serves as the building block for more complex neural network structures and provides a clear illustration of how deep learning models operate. Such an architecture was the basis for the neural operator used in my internship to learn the Green's function.

3.5.1 About the Multilayer Perceptron

Definition 10. *A Multilayer Perceptron (MLP) is a feedforward artificial neural network without loops, allowing unidirectional information flow from the input to the output layer. The intermediate layers are called hidden layers, each consisting of neurons that are not interconnected within the same layer.*

Formally, an MLP processes input vectors from $\mathbb{R}^d$. The input layer has d neurons, each corresponding to a component of the input vector. These neurons transmit information to the first hidden layer. Each neuron in a hidden layer computes a weighted sum of its inputs and applies an activation function before passing the result to the next layer.

Consider a neuron j in layer k . The input to this neuron is:

$$a_j = \sum_i w_{ji} z_i + b_j$$

where:

1. The sum is over all neurons i in the previous layer $k - 1$.
2. w_{ji} is the weight of the connection between neuron i in layer $k - 1$ and neuron j .
3. b_j is the bias term for neuron j .
4. z_i is the output from neuron i in layer $k - 1$.

The neuron j then emits the signal $\sigma(a_j)$ to the next layer, where σ is an activation function. The final layer emits the network's output, which can be a vector or scalar, depending on the task (classification or regression).

Fourier Feature Mapping and Fourier MLP

The Fourier feature mapping [7] transforms the input vector $x \in \mathbb{R}^d$ into a higher-dimensional space using a set of random frequencies. This is defined as:

$$\gamma(x) = [\cos(2\pi Bx), \sin(2\pi Bx)]$$

where $B \in \mathbb{R}^{m \times d}$ is a matrix of random frequencies, and m is the number of Fourier features. The entries of B are typically drawn from a Gaussian distribution with mean zero and standard deviation σ . The transformed input $\gamma(x)$ is then fed into the MLP.

By incorporating Fourier features, the Fourier MLP can better capture high-frequency components in the data, leading to improved performance in tasks requiring detailed and complex pattern recognition.

Fourier MLP have been showing promising results when it comes to learning Green's function in oscillatory problems such as the Helmholtz equation. The Fourier feature mapping allows the neural operator to capture high-frequency components in the data, which is essential for accurately learning the Green's function in such problems. This is why I relied on Fourier MLP in my internship to learn the Green's function in the Helmholtz equation.

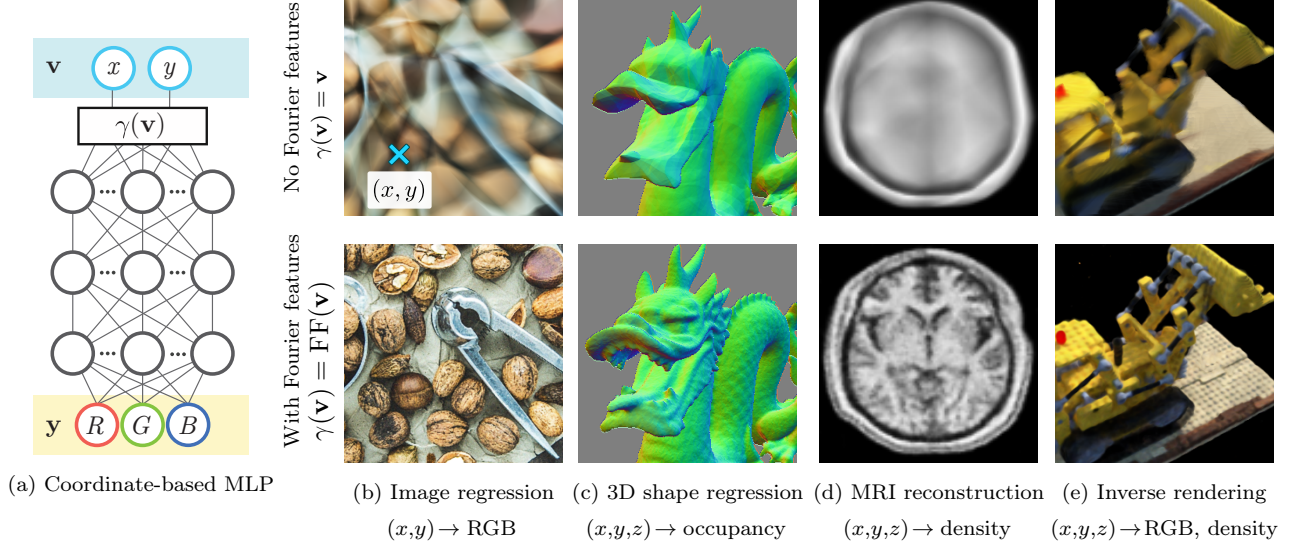


Figure 3.1: Fourier features enhance the performance of coordinate-based MLPs on high-frequency, low-dimensional regression tasks, whether under direct (b, c) or indirect (d, e) supervision. In an image regression example (b), where the network inputs pixel coordinates and outputs pixel colors, directly feeding coordinates into the network (a, top) results in blurry images. However, using a Fourier feature mapping (a, bottom) allows the MLP to capture higher frequency details, producing sharper images. The figure also comes from [7].

3.5.2 Activation Function σ of a Neuron

Definition 11. The output signal z_j of neuron j is defined as:

$$z_j = \sigma(a_j) = \sigma \left(\sum_i w_{ji} z_i + b_j \right)$$

where the sum is over all neurons i in the previous layer connected to neuron j , each with an output z_i , connection weights w_{ji} , and bias b_j .

It is crucial to use non-linear activation functions. If the activation functions were linear, the network would merely compute linear combinations of the inputs, which would significantly limit its representational power. Non-linear activation functions enable the network to approximate complex, non-linear mappings.

Example 3. Hyperbolic tangent ($\tanh$): Similar to the sigmoid but outputs values between -1 and 1, often used in hidden layers for its zero-centered output, which can make optimization easier.

Example 4. Rational activation: A function used for its ability to learn the Green function on the diagonal, providing flexibility in learning complex patterns. More details regarding such an activation function will be provided in the section 4.4.1 related to rational neural networks.

Remark 3.5.1. Activation functions in neural networks are characterized by several key properties: continuity for smooth transitions, monotonicity to preserve input order, differentiability for gradient-based optimization, boundedness to prevent output explosion, and non-linearity to enable learning of complex functions.

3.6 Physics-Informed Machine Learning

3.6.1 Introduction

Machine learning (ML) has seen significant success across domains, but its application in physical sciences benefits from integrating physical laws, resulting in more robust, interpretable, and generalizable models. Physics-Informed Machine Learning (PIML) [8] combines data-driven approaches with physical principles, ensuring models respect underlying physics.

PIML incorporates physical laws, such as differential equations (PDEs, ODEs, SDEs), and symmetry constraints into ML models, enhancing accuracy and interpretability. This approach is valuable for solving inverse problems and improving robustness in computer vision and reinforcement learning tasks.

3.6.2 Physics-Informed Neural Networks (PINNs)

PINNs are central to Physics-Informed Machine Learning, incorporating physical laws like PDEs and ODEs into the neural network's loss function. This ensures that predictions align with governing physical equations.

PINNs address both forward and inverse problems by embedding differential equations into the loss function. Given a PDE $\mathcal{L}(u) = 0$ over a domain Ω with boundary conditions $B(u) = 0$, the neural network u_θ approximates the solution u . The loss function combines data-driven loss $\mathcal{L}_{\text{data}}(\theta)$ with physics-informed loss $\mathcal{L}_{\text{physics}}(\theta)$, ensuring the model is both physically consistent and data-accurate, leading to robust and interpretable results.

3.6.3 Detailed Overview of the Physical Loss in PINNs

The physical loss in Physics-Informed Neural Networks (PINNs) ensures neural network solutions satisfy governing physical laws, typically described by differential equations, and fit observational data.

Formulating the Physical Loss

Physical loss originates from the differential equations (PDEs/ODEs) governing the system, penalizing violations of these laws.

PINNs and PDEs Consider a PDE:

$$\mathcal{L}(u) = f, \quad x \in \Omega, \quad B(u) = g, \quad x \in \partial\Omega, \quad (3.6.1)$$

where $\mathcal{L}$ is a differential operator, u is the solution, f is a source term, Ω is the domain, and $\partial\Omega$ is the boundary. The neural network $u_\theta(x)$ approximates $u(x)$, ensuring it satisfies both the PDE and boundary conditions.

Example 5. Consider the Helmholtz equation:

$$\nabla^2 u + k^2 u = 0, \quad x \in \Omega,$$

with boundary conditions $u(x) = u_b(x)$ for $x \in \partial\Omega$. Here, k is the wave number, and $u_b(x)$ represents boundary values. The neural network $u_\theta(x)$ approximates $u(x)$ to satisfy the Helmholtz equation and boundary conditions.

PDE Residual

Definition 12. The PDE residual quantifies the deviation of the neural network approximation from the true solution governed by the differential operator and source term. It is defined as:

$$\mathcal{R}(x) = \mathcal{L}(u_\theta(x)) - f(x),$$

where $\mathcal{L}$ is the differential operator, $u_\theta(x)$ is the neural network approximation, and $f(x)$ is the source term.

The physical loss for the PDE residual is the mean squared error of the residuals at the collocation points within the domain Ω . It is given by:

$$\mathcal{L}_{PDE}(\theta) = \frac{1}{N_r} \sum_{i=1}^{N_r} |\mathcal{R}(x_i)|^2,$$

where N_r is the number of collocation points in Ω .

Boundary Condition Residual

Definition 13. The boundary condition residual quantifies the deviation of the neural network approximation from the true boundary conditions. It is defined as:

$$\mathcal{R}_{BC}(x) = B(u_\theta(x)) - g(x),$$

where B is the boundary operator, $u_\theta(x)$ is the neural network approximation, and $g(x)$ is the boundary term.

The physical loss for the boundary conditions is the mean squared error of the residuals at the boundary points. It is given by:

$$\mathcal{L}_{BC}(\theta) = \frac{1}{N_b} \sum_{j=1}^{N_b} |\mathcal{R}_{BC}(x_j)|^2,$$

where N_b is the number of boundary points.

Total Physical Loss

Definition 14. The total physical loss combines the PDE and boundary condition losses:

$$\mathcal{L}_{physics}(\theta) = \mathcal{L}_{PDE}(\theta) + \lambda_{BC} \mathcal{L}_{BC}(\theta),$$

where λ_{BC} is a weighting factor that balances the contribution of the boundary condition loss.

Integrating Physical Loss into PINNs

Definition 15. The overall loss function for training the Physics-Informed Neural Network (PINN) is:

$$\mathcal{L}(\theta) = \mathcal{L}_{data}(\theta) + \lambda \mathcal{L}_{physics}(\theta),$$

where:

1. $\mathcal{L}_{data}(\theta)$ aligns the network's predictions with observed data.
2. $\mathcal{L}_{physics}(\theta)$ enforces the governing physical laws.
3. λ is a weighting factor that balances the data-driven loss and the physical loss.

Data-Driven Loss The data-driven loss measures the discrepancy between predictions and observed values:

$$\mathcal{L}_{\text{data}}(\theta) = \frac{1}{N_d} \sum_{k=1}^{N_d} |u_{\theta}(x_k) - y_k|^2,$$

where N_d is the number of data points.

Details on specific loss functions for the Green’s function will be provided in subsequent sections.

3.6.4 Training PINNs

Training a PINN involves minimizing the combined loss function $\mathcal{L}(\theta)$ through gradient-based optimization. The network’s parameters θ are adjusted to ensure both data accuracy and adherence to physical laws.

3.6.5 Advantages of PINNs

PINNs offer several advantages, including efficient handling of high-dimensional problems and the integration of diverse data with physical laws, even in the presence of noise or incomplete datasets. Their adherence to physical principles ensures robust generalization to unseen data, making them reliable across various scenarios.

3.6.6 Challenges with PINNs

Challenges include complex training due to sophisticated loss functions, requiring careful hyperparameter tuning. Scalability remains a concern for large systems, and ensuring that learned representations are interpretable and physically meaningful is still an open problem.

In our case, using second-order ODEs required differentiating the physical loss two to three times (to compute the Laplacian in the residual) and performing gradient descent to minimize the loss function. The physics-informed approach therefore caused memory overload and systematic training crashes, as showcased in section 4.7.3. Consequently, we adopted a data-driven approach to learn the Green’s function, as also noted in Nicolas Boull  ’s thesis [9].

3.7 Neural Operators

Introduction Mapping between function spaces is crucial in numerous scientific and engineering applications, such as solving differential equations, which involves transforming a coefficient function into a solution function. Traditionally, this process entails discretizing these function spaces and employing standard learning models like neural networks. However, such methods often fail to generalize effectively to discretizations beyond the training data grid.

To overcome this limitation, a novel deep-learning framework known as neural operators [11] has been introduced. Neural operators are designed to map directly between function spaces on bounded domains, offering a significant advantage over standard neural networks. They can be discretized using various methods and at different resolutions without necessitating retraining, thereby achieving discretization invariance—a property that standard neural networks lack.

About the role of neural operators Neural operators achieve this by learning mappings between infinite-dimensional function spaces through multi-layer architectures. These architectures consist of operator layers combined with non-linear activations, ensuring the overall composition of the operator. By focusing on linear operator layers coupled with non-linear activations, neural operators can universally approximate continuous operators, providing a robust and flexible solution to the challenges posed by traditional methods.

Similarities and differences with PINNs Neural Operators and Physics-Informed Neural Networks (PINNs) both represent cutting-edge advancements in the application of deep learning to solve PDEs. They share a common goal of leveraging neural networks to model complex physical systems, thereby enhancing our ability to predict and understand phenomena governed by PDEs. Both frameworks integrate domain knowledge into the learning process—Neural Operators through their ability to map between function spaces and handle various discretizations, and PINNs by embedding physical laws directly into the loss function. This integration ensures that the models not only fit observed data but also adhere to underlying physical principles, providing robust and interpretable solutions.

3.8 Deep Green Operators and Their Problem Setting

3.8.1 Posing the Problem

Consider a general elliptic partial differential equation (PDE) of the form:

$$\begin{cases} L_\mu u(x) = \nabla \cdot (A(x, \mu) \nabla u(x)) + B(x, \mu) u(x) = f(x), & x \in \Omega, \mu \in \mathcal{M} \\ a \partial_x u + bu = g(x), & x \in \partial\Omega \end{cases}$$

where μ represents some parameters.

L_μ is a linear differential operator, $A(x, \mu)$ and $B(x, \mu)$ are coefficient functions storing information related to the physical properties of the system, and $f(x)$ is the source term.

The boundary conditions are given by $a \partial_x u + bu = g(x)$ on the boundary $\partial\Omega$, where a and b are constants, and $g(x)$ is the boundary source term.

Such boundary conditions are formulated in this manner to cover a wide range of boundary conditions, including Dirichlet, Neumann, and Robin boundary conditions.

In our case, we will focus on the Dirichlet boundary conditions, whether homogeneous or inhomogeneous.

Example 6. For instance, we can consider the Helmholtz equation:

$$\Delta u + k^2 u = f(x)$$

with $\mu = k$ being the wavenumber, which translates to $A(x, \mu) = 1$ and $b(x, \mu) = k^2$.

3.8.2 Objective

Our goal is to learn an operator $\mathcal{F}$ that approximates the true solution operator $\mathcal{F}_{\text{true}}$ using a finite set of observations of input-output pairs $\{\mu_j, u_j\}_{j=1}^N$, where each μ_j is a parameter in $\mathcal{M}$ and u_j are functions defined on Ω . In practice, the training data is obtained numerically or through experiments, meaning that μ_j and u_j are discretized. Let $P_K = \{x_1, \dots, x_K\} \subset \Omega$ be a K -point discretization of the domain Ω . We assume we have observations $\mu_j|_{P_K}$ and $u_j|_{P_K}$ for a

finite collection of input-output pairs indexed by j . Our goal is to learn a discretization-invariant mapping based on this discretized data.

3.8.3 Kernel Formulation and Neural Operator Theory

The neural operator theory proposes to compute a reduced model of the PDE by learning a network that maps the input of the problem, such as the right-hand side $f(x)$, to the solution. To obtain a good reduced model capable of generalizing to many discretizations, the aim of the neural operator (NO) theory is to propose architectures where the result does not depend on the resolution/discretization of the inputs. We refer to it as neural operator theory because creating this mapping independent of resolution is 'equivalent' to constructing a map between functions.

Here, we propose to work on an architecture based on the Green's theory [12]. The Green's function is a function $G(x, y)$ that satisfies:

$$L_\mu(G(x, y), \nabla G(x, y), \Delta G(x, y)) = \delta(x - y)$$

with a specific boundary condition. If we have the Green function, we can show that the solution of the PDE satisfies:

$$u(x) = \int_{\Omega} G(x, y) f(y) dy + \int_{\partial\Omega} (A(x, \mu) (u(x) \nabla_x G(x, y) - G(x, y) \nabla_x u(x))) \cdot n dy$$

In general, if Dirichlet boundary conditions are used, we choose a Green function with homogeneous boundary conditions. If Neumann boundary conditions are used, we choose the Green function with homogeneous Neumann boundary conditions. The Robin boundary case is not treated here. The focus will be on the construction of the neural operator based on the approximation to the Green function. We will differentiate between the three cases.

More practically, we aim to learn a neural network N_κ with a kernel κ that approximates the Green's function. The neural network N_κ is trained to minimize the mean squared error between the predicted and true Green's function, enabling a discretization-invariant mapping for efficient PDE solutions.

The neural network N_κ must fit the equation:

$$L_\mu(\kappa(x, y), \nabla \kappa(x, y), \Delta \kappa(x, y)) = \delta(x - y)$$

with appropriate boundary conditions.

The solution of the PDE provided by the neural operator is:

$$u_{\text{NO}}(x) = \int_{\Omega} \kappa(x, y) f(y) dy + u_{\text{hom, NO}}(x) \quad (3.8.1)$$

where $u_{\text{hom, NO}}(x)$ is the homogeneous solution satisfying the boundary conditions.

More details regarding the loss function and the training of the neural operator will be provided in the section 4.6 related the finally-chosen loss function that was used in my internship.

The Green function approach is crucial in the context of neural operator theory as it enables the learning of a function that maps between the inputs and outputs while being invariant to discretization. By modeling the kernel with a neural network κ , we aim to capture the essential properties of the Green function, allowing for accurate and efficient solutions to the PDEs across various applications.

3.9 Conclusion

The scientific machine learning framework combines traditional scientific computing with modern machine learning to solve complex problems in various domains. By integrating physical laws and domain-specific knowledge, it enables accurate and interpretable solutions. The Scimba library provides tools for data sampling, model training, and uncertainty quantification, empowering researchers to address diverse scientific challenges.

With this foundation, we now focus on *data sampling, architecture, and implementation*. The next chapter explores how these elements combine to create effective and reliable scientific machine learning models.

Chapter 4

Data sampling, architecture and implementation

4.1 Introduction

In this section, we delve into the practical aspects of implementing the neural operator to learn the Green’s function for the Helmholtz equation. We discuss the data sampling process, the architecture of the neural network, and the implementation details using the Scimba library. These components are essential for training the neural operator effectively and obtaining accurate solutions to the PDE.

4.2 The Scimba library

Introduction to the Scimba library

The Scimba library is a comprehensive and versatile toolkit designed for scientific machine learning, with a particular emphasis on classical PDEs. Created by researchers Emmanuel Franck and Victor Michel-Dansac from Inria, this library serves as a valuable resource for research endeavors within the field of Scientific Machine Learning (SciML). It offers an innovative approach by integrating classical numerical methods with modern machine learning techniques, bridging the gap between traditional computational science and contemporary AI-driven methods.

Scimba is particularly notable for its modular design, which allows users to build custom solutions for various SciML problems. The library supports Physics-Informed Neural Networks (PINNs) and Neural Operators, offering robust methods for general sampling, model reduction, and uncertainty quantification, all essential for solving PDEs with embedded physical laws.

A few details about the Scimba library regarding the sampling of points and the PINNs

The Scimba library, tailored for scientific machine learning, is particularly suited for projects involving classical PDEs within the SciML domain. The library is highly modular, with `scimba.sampling` and `scimba.pinns` being two key components that enable users to address various aspects of the PDE problem-solving process.

The `scimba.sampling` module is essential for implementing general sampling methods that are critical for Physics-Informed Neural Networks (PINNs) and Neural Operators. This module

provides tools for discretizing continuous domains and functions, which is a crucial step when training neural networks to handle continuous data.

Among these, the `UniformSampling` class provides a method for uniformly sampling points within a specified domain. This is particularly useful when a uniform distribution of points across the domain is required, ensuring that the neural network receives a consistent and representative set of data points for training.

Another key component is the `MuSampler` class, which is designed for sampling the parameter space of PDE or ODE models. It leverages the flexibility of the Scimba library’s sampling methods to generate samples across a specified domain, making it essential for tasks that involve varying parameters within a model.

The `XSampler` class is specialized for sampling the spatial domain, particularly in scenarios where different types of domains (such as square, disk, or more complex geometries) are involved. This class handles the complexities of domain-specific sampling, including boundary conditions, ensuring that the samples generated are appropriate for the specific PDE model being studied. The `XSampler` class also integrates various submodules to manage different parts of the domain, such as subdomains and holes, offering a comprehensive solution for spatial sampling challenges.

The `data_sampling_pde_x` class is a high-level utility that combines these sampling tools to generate data for training PDE models. It coordinates the sampling of both the domain and the boundary conditions, as well as the generation of corresponding function values. This class simplifies the process of setting up complex sampling schemes, allowing users to focus on the higher-level aspects of their scientific computing tasks.

The `scimba.pinns` module is crucial for creating and training Physics-Informed Neural Networks (PINNs), leveraging the networks and sampling methods from `scimba.sampling`. PINNs solve PDEs by incorporating physical laws directly into the neural network training process, ensuring solutions adhere to underlying physical principles.

Activation Functions in Scimba

The `scimba.nets` module provides various activation functions essential for defining neuron outputs in neural networks. These functions introduce non-linearity, enabling the model to capture complex patterns. Scimba includes common activation functions like `ReLU`, `Sigmoid`, and `Tanh`, as well as specialized functions for scientific computing.

The `Tanh` (hyperbolic tangent) function, with an output range of -1 to 1, is favored in scenarios involving physical systems requiring symmetrical output ranges. Unlike `Sigmoid`, which compresses inputs to $[0, 1]$, `Tanh` is suitable for tasks needing effective capture of negative values. This property is particularly useful in modeling phenomena with symmetrical behavior around zero.

Scimba also offers advanced functions like `RationalAct`, a rational approximation providing enhanced expressiveness over standard activations like `ReLU` and `Tanh`. Defined as a ratio of two polynomials, `RationalAct` models a wider variety of functions, especially where higher non-linearity is required. This flexibility makes it powerful for scientific machine learning, enabling accurate approximations of complex physical processes. In our case, `RationalAct` was employed to capture complex, non-linear relationships, yielding better results for the Green’s function $G(x, \xi)$, particularly when $x = \xi$.

Neural Operators in Scimba

Neural Operators represent one of the advanced features of the Scimba library, designed to address the challenges of learning mappings between function spaces, a common requirement in solving PDEs. Unlike traditional neural networks that map inputs to outputs in finite-dimensional spaces,

Neural Operators aim to learn operators that map functions to functions, making them ideal for problems where the solution involves complex, infinite-dimensional spaces.

The `scimba.neural_operators` module implements these operators, offering tools for constructing and training models that can generalize across different resolutions and geometries. This is particularly useful in scenarios where the PDE solutions need to be learned over varying domains or under different boundary conditions. The Neural Operators in Scimba leverage techniques such as kernel integral operators and Fourier neural operators, which allow for efficient learning and approximation of these complex mappings.

By using Neural Operators, Scimba enables researchers to tackle a broader range of SciML problems, from fluid dynamics to quantum mechanics, where traditional methods may struggle due to the high dimensionality and complexity of the solution space. The ability to generalize across different scales and domains makes Neural Operators a powerful tool in the Scimba library, expanding the possibilities for scientific discovery and engineering applications.

Conclusion

The Scimba library empowers researchers in scientific machine learning with tools ranging from activation functions and neural networks to advanced techniques like Neural Operators and the Neural Galerkin method. Its modular design, comprehensive documentation, and integration of classical methods with modern AI make it an invaluable resource for tackling PDE-related problems.

More detailed information about the Scimba library, along with tutorials, additional examples, and API references, can be found in the Scimba documentation.

4.3 Data sampling

4.3.1 The numerical solvers used to obtain the true target solutions

Introduction to the Finite Difference Method

Finite difference methods [14] approximate differential operators by replacing derivatives with differential quotients. The domain is partitioned in space and time, and solutions are computed at these points. The error between the numerical and exact solutions, known as discretization or truncation error, arises from using a finite part of a Taylor series for the approximation.

Centered Finite Difference Approximations

For a more accurate approximation, the centered finite difference method can be used. The centered finite difference approximation for the first derivative is given by:

$$u'(x) \approx \frac{u(x+h) - u(x-h)}{2h}$$

The centered finite difference approximation for the second derivative is given by:

$$u''(x) \approx \frac{u(x+h) - 2u(x) + u(x-h)}{h^2}$$

These centered approximations provide higher accuracy compared to the forward or backward finite difference methods, as they take into account the function values on both sides of the point x .

During my internship, I utilized these finite difference methods to solve various differential equations, demonstrating their practical applicability and effectiveness in numerical analysis. More details will be provided in the subsequent sections related to the history of my implementation.

The SciPy BVP Solver

Throughout my internship, I utilized the SciPy BVP solver [15] to efficiently address boundary value problems (BVPs) arising from differential equations. This tool provides high-accuracy numerical solutions using methods such as finite difference and collocation techniques.

The `scipy.integrate.solve_bvp` function is designed for solving BVPs in ordinary differential equations (ODEs), where solutions must satisfy conditions at the domain boundaries. Unlike initial value problems (IVPs), BVPs require solutions to meet specified boundary conditions at multiple points.

This solver employs a collocation method, fitting a piecewise polynomial to approximate the solution while ensuring it satisfies both the differential equations and boundary conditions. The user provides the system of ODEs, the boundary conditions, and an initial guess, which the solver iteratively refines to achieve the desired accuracy.

The `solve_bvp` function is versatile, capable of handling both linear and nonlinear boundary conditions, and even scenarios with fewer boundary conditions than equations. This flexibility makes it suitable for a wide range of scientific and engineering problems, particularly for stationary 1D PDEs that can be transformed into ODEs using a BVP approach.

Conclusion about the numerical solvers

Finally, after evaluating various numerical solvers, we ultimately used the Boundary Value Problem (BVP) solver for the final results, beginning in section 5.5. This choice was driven by its robustness and accuracy in handling the specific requirements of our problem, ensuring reliable and precise solutions.

4.4 The architecture of the neural network

4.4.1 The preliminary notion of rational neural networks

Definition 16. *A rational neural network is a type of neural network that uses rational functions as activation functions or as part of its layers. Rational functions of degree (m, n) are defined as the ratio of two polynomials:*

$$R(x) = \frac{P(x)}{Q(x)} = \frac{a_0 + a_1x + \dots + a_mx^m}{1 + b_1x + \dots + b_nx^n}$$

where $P(x)$ and $Q(x)$ are polynomials of degrees m and n , respectively.

Key Characteristics of Rational Neural Networks

Rational neural networks use **rational functions as activation mechanisms**, unlike traditional neural networks that use ReLU, sigmoid, or tanh. These functions can adapt to a wider variety of input-output relationships, enhancing model expressiveness and flexibility. **Rational functions can approximate a broad range of functions** more efficiently than polynomial or traditional activation functions, capturing both local and global behaviors. The parameters of the polynomials $P(x)$ and $Q(x)$ in the rational function are learned during training, allowing the network to

dynamically adjust its activation functions to better fit the data. This balance of **stability** and **flexibility** makes rational neural networks suitable for tasks with complex features, such as sharp transitions or oscillations, which are challenging for traditional activation functions.

4.4.2 The architecture of the neural network used in the implementation

In our deep learning framework, very inspired by N. Boullé's works [9], two distinct rational neural networks, N_G and N_{hom} , are employed to learn critical components of the system's response governed by the unknown linearized PDE.

The figure below, sourced from Nicolas Boullé's thesis [9], illustrates the architecture of the neural network used in the implementation:

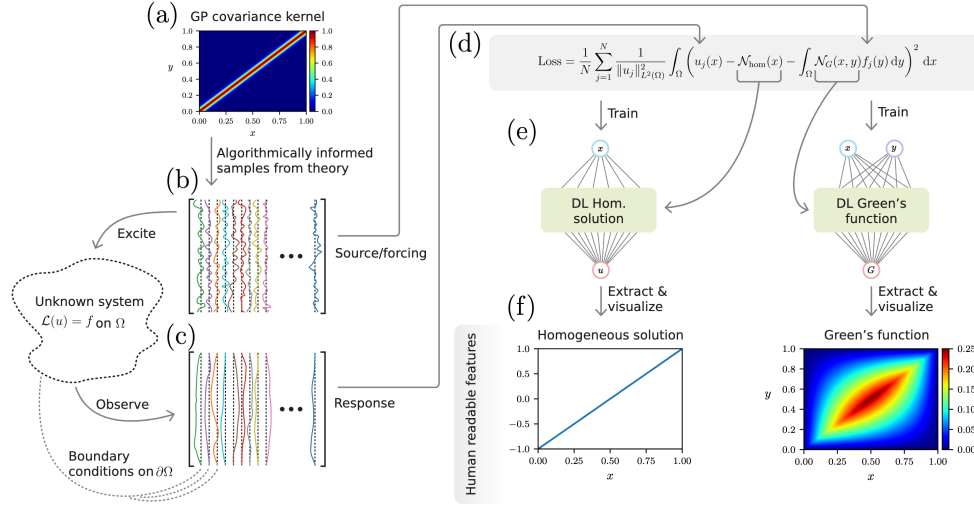


Figure 4.1: Architecture of the neural network used in the implementation

Role of N_G (Green's Function Network):

The network N_G is tasked with approximating the Green's function $G(x, y)$. The Green's function represents the influence that a point source at location y has on the solution at another point x . Essentially, N_G learns how the system propagates the effects of the forcing terms throughout the domain. The Green's function is fundamental in understanding the system's response to various excitations and is crucial for reconstructing the full solution to the PDE from the forcing terms.

In the implementation, N_G is implemented with two modules: `green_kernel_x` for the Green's kernel G in the convolution, and `kernel_sublayer_x` for the discretized Green's convolution itself.

Then, in the neural operator theory (evoked in section 3.8.3), the `green_kernel_x` part of N_G is the neural network N_{κ} that approximates the Green's function with the kernel $\kappa(x, y)$. On the other hand, the Scimba module `kernel_sublayer_x` is what performs the Monte-Carlo approximation of the convolution integral in equation 3.8.1.

More specific details about the architecture and implementation of N_G will be provided in the sections 4.7 and 5.2.

Role of N_{hom} (Homogeneous Solution Network):

The network N_{hom} is responsible for learning the homogeneous solution $u_{\text{hom}}(x)$, which satisfies the PDE without any external forcing (i.e., when the forcing term $f_j = 0$). The homogeneous

solution encapsulates the boundary conditions of the problem and any inherent dynamics of the system independent of external inputs. By learning this component, N_{hom} ensures that the final solution satisfies the necessary boundary conditions.

In our implementation, N_{hom} is implemented with the module `local_sublayer_x`.

In the neural operator theory, N_{hom} is the neural network that approximates the homogeneous solution $u_{\text{hom,NO}}(x)$ in equation 3.8.1.

Successful results of training N_{hom} will be detailed in the section E.2.2.

Conclusion about the architecture:

Together, these networks allow for a comprehensive representation of the system's response: N_G accounts for how external forces influence the system, while N_{hom} ensures that the boundary conditions are met. The training process involves jointly optimizing both networks to minimize the discrepancy between the predicted and actual system responses, effectively capturing the underlying PDE's characteristics through these two key functions. This dual-network approach is particularly powerful as it decouples the problem into manageable parts, allowing each network to specialize in a critical aspect of the solution.

To allow the training of both (N_G and N_{hom}) networks at the same time, I implemented Scimba module `deep_operator_x`, according to equation 2.3.3.

4.5 Data generation and training

Preliminary Given a one-dimensional domain Ω (for example, $[0, 1]$) and a certain parameter domain $\mathcal{M}$, we aim to generate a training dataset that captures the system's response to various forcing terms. This dataset will be used to train the neural networks N_G and N_{hom} to learn the Green's function and homogeneous solution, respectively.

Let us add that there is N_{simu} simulations (which means, N_{simu} parameters instances μ_j and $N_{\text{collocation}}$ collocation points $\{x_i\}_{i=1}^{N_{\text{collocation}}}$ for each simulation) and N_{sensor} sensors points $\{y_i\}_{i=1}^{N_{\text{sensor}}}$ for the Green convolution.

To generate the training dataset for our deep learning approach, we follow a systematic process involving three key steps:

1. First, we generate the forcing terms by sampling N_{simu} random functions $\{f_j\}_{j=1}^{N_{\text{simu}}}$ from a Gaussian process (in equation B.2.2) characterized by a carefully selected covariance kernel. These forcing terms serve as the excitations for the system and are promising when it comes to approximating functions and solving PDEs [10]. More detailed information about the Gaussian process used will be provided in the chapter B in the appendix.
2. Next, for each sampled forcing term f_j , we solve the governing PDE $Lu_j = f_j$ using numerical methods (in section 4.3.1) such as finite difference or boundary value problem (BVP) solvers. This yields the corresponding system responses $\{u_j\}_{j=1}^{N_{\text{simu}}}$, which represent the solutions to the PDE driven by the forcing terms.
3. Finally, the forcing terms and system responses are sampled at specific points within the domain Ω . The discretization points for the forcing terms are denoted as $\{y_i\}_{i=1}^{N_{\text{sensor}}}$, and for the responses as $\{x_i\}_{i=1}^{N_{\text{data}}}$. In our case, $N_{\text{data}} = N_{\text{sensor}}$. These points are chosen to align with the grid used during training, minimizing computational costs and improving training efficiency.

The Gaussian process used to generate forcing terms typically employs a radial basis function (RBF) kernel, $K_{RBF}(x, y)$, defined as introduced in equation B.3.1, where ℓ is the length-scale parameter. This parameter is crucial for determining the diversity and oscillatory nature of the sampled functions, which is essential for accurately capturing the various modes of the operator L .

For problems defined over an interval $\Omega = [a, b]$, we introduce a normalized length-scale parameter $\lambda = \ell/(b - a)$ to ensure that the method remains invariant to the domain's size. The length-scale ℓ is chosen to be larger than the discretization size $(b - a)/N_f$ to ensure adequate resolution of the sampled functions.

In cases where the PDE represents a boundary-value problem, we relied on the two numerical solvers mentioned in section 4.3.1.

This methodical approach ensures that the training data is well-suited for the neural networks to learn the underlying Green's function and homogeneous solution accurately, thus capturing the essential characteristics of the PDE.

4.6 The loss function of the architecture

The loss function used in the architecture is a combination of two key components: the loss function associated to the Green kernel and the loss function associated to the homogeneous solution. These components are combined to form the overall loss function, which guides the training process and ensures that the neural networks learn the system's response effectively.

The formulation of the loss function stems from the formulation of the solution of the BVP in equation 2.3.3.

4.6.1 Loss Function for the Green Kernel

The loss function for the Green kernel aims to minimize the discrepancy between the predicted Green's function $\kappa(x, y)$ and the true Green's function $G(x, y)$. This is achieved by comparing the output of the neural network N_G with the exact Green's function, computed using the numerical solvers mentioned in section 4.3.1.

The loss function for the Green kernel is defined as the mean squared error (MSE) between the true targets used for the training (as highlighted just above, in section 4.5) and the convolution of the predicted Green kernel $\kappa(x, y)$ with the source terms.

It can be written as follows:

$$\begin{aligned} \mathcal{J}_G(x) &= \frac{1}{N_{\text{simu}}} \sum_{j=1}^{N_{\text{simu}}} \left(\int_{\Omega} \kappa(x, y; \mu_j) f_j(y) dy - u_j(x) \right)^2 dx \\ &\approx \frac{1}{N_{\text{simu}}} \sum_{j=1}^{N_{\text{simu}}} \left(\frac{\text{vol}(\Omega)}{N_{\text{sensor}}} \sum_{i=1}^{N_{\text{sensor}}} \kappa(x, y_i; \mu_j) f_j(y_i) - u_j(x) \right)^2 dx, \end{aligned}$$

where N_{simu} is the number of simulations, N_{sensor} is the number of sensors, and $\text{vol}(\Omega)$ is the volume of the domain Ω .

In our case, since all the case are defined over the interval $\Omega = [0, 1]$, it implies that $\text{vol}(\Omega) = 1$.

4.6.2 Loss Function for the Homogeneous Solution

The loss function for the homogeneous solution focuses on minimizing the discrepancy between the predicted homogeneous solution $u_{\text{hom}}(x)$ and the true homogeneous solution. This component

ensures that the neural network N_{hom} accurately captures the system’s behavior in the absence of external forcing terms, satisfying the boundary conditions and inherent dynamics of the system.

The loss function for the homogeneous solution is defined as the mean squared error (MSE) between the predicted solution and the true solution obtained from the numerical solvers. It is given by:

$$\mathcal{J}_{\text{hom}}(x) = \frac{1}{N_{\text{simu}}} \sum_{j=1}^{N_{\text{simu}}} (u_{\text{hom}}(x; \mu_j) - u_{j_{\text{hom}}}(x))^2,$$

where $u_{j_{\text{hom}}}(x; \mu_j)$ represents the true homogeneous solution obtained from the numerical solvers corresponding to the source term f_j .

4.6.3 Overall Loss Function

The overall loss function for the architecture is the sum of the loss functions for the Green kernel and the homogeneous solution. By jointly optimizing these components, the neural networks N_G and N_{hom} learn to accurately model the system’s response to external forcing terms and satisfy the boundary conditions.

4.7 History of the implementation throughout the internship

4.7.1 Introduction

This section details the progress made over the course of the internship, including the challenges encountered, the solutions implemented, and the outcomes of various experiments.

4.7.2 Initial Phase: Understanding the Problem and Setting Up the Environment

From February 19th to March 1st, 2024

At the beginning of the internship, I focused on understanding the theoretical background of neural operators, particularly DeepONets and their application in PDE learning. The initial discussions and documents provided by mentors and colleagues helped clarify key concepts about deep learning, Green’s functions, and their role in solving PDEs. This foundational knowledge was essential for grasping the intricacies of the project and formulating a coherent strategy for implementation.

To implement and test these concepts, I began setting up the necessary environment, including installing libraries and understanding the codebase of Scimba, a framework used for training neural operators.

Challenges: Understanding the intricate details of the mathematical formulations behind neural operators and their implementation in code was a significant challenge. I delved deeply into the theoretical framework of Green’s functions and neural operators, which provided a solid foundation for my work. Additionally, configuring the environment posed its own set of difficulties, particularly with setting up SSH access to the Gaya and V100 machines necessary for running large-scale computations. Overcoming these hurdles was essential for progressing with the project.

Solutions: I successfully resolved the environment configuration issues by coordinating with the IT support team and setting up the SSH access correctly. This allowed me to utilize powerful GPUs for training models, crucial for handling large datasets and complex computations. Additionally, many notional confusions about deep learning, Physics-Informed ML, and neural operators were resolved through exchanges with Emmanuel Franck and Victor Michel Dansac.

4.7.3 Experimentation and Optimization of Neural Operators

From March 4th to April 12th, 2024

With the environment set up, I moved on to experimenting with different neural operator architectures, particularly focusing on Green’s functions. The first set of experiments involved training a Linear Green Operator with an MLP kernel. I configured the training parameters, including the number of epochs, points of collocation, and sensors, and observed the model’s performance.

I also had to implement the module (in the sense of a PyTorch module) of the Green kernel and the module related to discrete convolution to be able to use the Green kernel in the neural network. The discrete convolution was performed using Monte-Carlo integration methods with the kernel PyTorch tensor provided by the first module and the tensor contained the values of the source Terms to the source terms.

Key Experiments: I implemented the Monte-Carlo integration method to compute the convolution of the Green’s kernel with the source terms. This involved generating random samples and evaluating the kernel function at these points to approximate the convolution integral. To verify the accuracy of the Monte-Carlo integration, I conducted tests using predefined kernel and source terms. These tests aimed to check whether the integration was properly approximated and whether the results aligned with the theoretical expectations of using Monte-Carlo integration methods.

To improve computational efficiency, I vectorized the implementation of the kernel functions, specifically for the modules related to the Green kernel module and the discrete Monte Carlo Green convolution using the Green kernel from the kernel module. Benchmarking results showed significant performance gains, particularly when increasing the number of points of collocation and sensors. Additionally, this vectorization also improved the performance of the Green kernel module itself. However, the models often crashed due to excessive memory usage, especially on the V100 GPU. To address this, I experimented with the `.detach()` method in PyTorch to prevent unnecessary gradient calculations. Initially, this seemed promising in reducing memory usage, but further testing revealed that the improvement was minimal.

The physics-informed approach was unfruitful both in precision and performance, frequently resulting in memory crashes due to the Laplacian computations.

Outcomes: The Monte-Carlo convolution module fit the theoretical expectations of Monte-Carlo integration, providing accurate approximations of the convolution integral. The vectorization efforts led to significant reductions in training time, allowing for more extensive experimentation. However, while the `.detach()` method provided some memory savings, it was not sufficient to overcome the GPU memory limitations for more complex models. This led to exploring alternative strategies, such as using a full data-driven approach instead of physics-informed losses.

4.7.4 Transition to a Data-Driven Approach

From April 15th to May 10th, 2024

Given the limitations encountered with the physics-informed loss functions, especially in handling the Laplacian operator, I shifted focus to a purely data-driven approach. This involved modifying the loss function to directly compare the predicted and reference solutions, bypassing the need for calculating second-order derivatives using methods that suited Physics-Informed Neural Networks (PINNs).

Key Modifications: I modified the trainer and PDE classes to generate and utilize data points for training, rather than relying solely on physical laws. This approach was inspired by the work of Nicolas Boullé and his codebase, which I studied in detail to replicate similar strategies. For this data-driven approach, I used a manufactured true target solution and source term. The true target was defined as $u_{\text{ref}}(x; \mu) = \sin(2\pi x) \cdot \exp\left(-\frac{(x-1)^2}{0.05\mu}\right)$, where μ is a parameter. I generated 100 different values of μ and computed the corresponding target and source terms. The source term $f_{\text{source}}(x; \mu)$ was defined as $-u_{\text{ref}}''(x; \mu)$.

I tested the data-driven approach on both MLP and Fourier kernel-based operators. The Fourier kernel, with its ability to capture more complex features, showed better performance in terms of fitting the training data and extrapolating to unseen parameter ranges.

Outcomes: The data-driven approach yielded more stable training and allowed for a larger number of collocation points, previously infeasible due to memory constraints. The Fourier kernel, with more features, provided better predictions despite higher computational costs.

To introduce variability and stochasticity, we used Gaussian process samples with an RBF kernel as source terms. Their numerical solutions, computed using a finite difference method and the SciPy BVP solver, served as true target solutions. This approach generated diverse training examples, enhancing the model's robustness and generalization.

4.7.5 Redesign of the training data by relying on Gaussian Processes

From May 20th to May 31st, 2024

During this period, our primary focus was on debugging and improving the data sampling process for the neural operator, particularly for solving PDEs using Gaussian processes. Initially, we realized that the dataset used for training the neural network lacked variability and stochasticity, as it was generated only once before the epochs began. This led to limited data diversity and overfitting. To address this, we considered resampling data points at each epoch to introduce more variability and improve generalization. However, this idea was ultimately discarded as it proved irrelevant to the problem at hand. Instead, we focused on generating a diverse dataset from the outset, which led to more stable and effective training.

The problem rather lied in the use of $u_{\text{ref}}(x; \mu) = \sin(2\pi x) \cdot \exp\left(-\frac{(x-1)^2}{0.05\mu}\right)$, where μ is a parameter. I generated 100 different values of μ and computed the corresponding target and source terms. The source term $f_{\text{source}}(x; \mu)$ was defined as $-u_{\text{ref}}''(x; \mu)$, as a reminder. Instead of generating data at each epoch, we decided to rely on Gaussian processes, as said above.

Key Modifications: To enhance the training process and ensure consistent performance, we implemented data resampling at each epoch. We tested the model with more data points and a higher number of epochs to refine the learning process. Initially, we tested without any parameters

to achieve an accurate approximation of the problem. Additionally, we considered developing a physics-informed loss function without second derivatives if memory issues arose, such as using energy-based methods. But because of other priorities and available resources, we decided to solely focus on the data-driven approach.

As the project advanced, we shifted our attention to refining the training process of the neural network and ensuring that the implementation was free of errors. A significant part of this phase involved verifying that the data generated by the `make_data` function was consistent with the expectations and aligned with the theoretical framework.

Outcomes: We observed that the network’s performance was heavily dependent on the training data. To address this, we compared outputs from different data generation methods to ensure consistency. By resampling data points and using methods like Gaussian process regression (GPR) with the RBF kernel, we provided the model with sufficient and diverse training data. Discussions during this period focused on fine-tuning the neural network architecture, including kernel choice, boundary condition handling, and overall model structure. Boundary condition challenges were addressed by incorporating them into the solver using a finite difference method with numpy, then translating the solution into a torch tensor for further processing.

4.7.6 Advancing the Training with Full Data and Addressing Early Convergence Issues

From June 3rd to June 11th, 2024 From early June, our focus shifted to training the neural network with full data, without incorporating physics-informed loss. On June 4th, we conducted training with various configurations, including 20,000 collocation points and up to 1,000 epochs. Initially, the results were unsatisfactory, leading us to suspect potential errors in the training data or network implementation. We discovered that a postprocessing function, specifically a polynomial $P_{\text{post-proc.}}(x) = x^2(1 - x)^2$, was being forcibly applied by multiplying the prediction output with it. After removing this multiplication, the issue was resolved.

The figure showcasing the results of such training (contrasting to the successful instance provided in section 5.6) lies below:

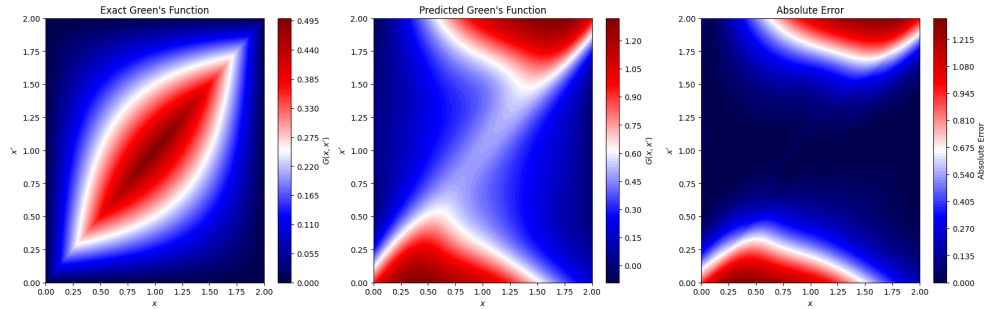


Figure 4.2: Results of the erroneous training with 20,000 collocation points and 1,000 epochs

Even though we did not encounter significant difficulties related to the convergence of the training process, particularly with the tanh activation function, we decided to further improve the convergence of the prediction by relying on a rational activation function, as suggested by Boullé’s work. Specifically, we used a rational activation function with a numerator of degree 3 and a

denominator of degree 2, with the following initial coefficients:

$$\begin{aligned} p_0 &= 1.1915, \\ p_1 &= 1.5957, \\ p_2 &= 0.5, \\ p_3 &= 0.0218, \\ q_0 &= 2.3830, \\ q_1 &= 0.0, \\ q_2 &= 1.0. \end{aligned}$$

These coefficients were implemented as torch parameters, meaning they change throughout the training process. This setting was chosen as the optimal initial configuration according to Boullé’s work. These rational activation functions proved to be more effective in reducing the loss rapidly compared to traditional activation functions. To further improve the training process, we decided to use the LBFGS optimizer alongside Adam. LBFGS is particularly effective for problems with a large number of parameters, as it can handle the curvature information more efficiently. This combination of Adam and LBFGS contributed to the first wave of satisfying results, although some imperfections remained.

Key Experiments and Issues: To address oscillations and potential overfitting, we explored different strategies, including adjusting the number of simulations and sensors, and experimenting with the inclusion of Gaussian noise during training. Despite these efforts, the results suggested that a more systematic approach to hyperparameter tuning was necessary.

Outcomes: A major breakthrough occurred when the network, equipped with rational activations, showed significantly improved performance. More epochs were needed for convergence due to solution complexity. Successfully learning the Green’s function for the Laplace equation paved the way for other test cases, underscoring the importance of appropriate activation functions and systematic hyperparameter tuning.

4.7.7 Transition to More Difficult Problems

From June 15th to July 15th, 2024 As the project progressed, we began addressing parametric problems, focusing on the advection-diffusion equation with a variable diffusion coefficient (D). This step was critical in extending the applicability of the neural operator to a broader class of PDEs. The equation we considered was $Du'' + 2u' + u = f$, where u is the unknown function and f is the source term. The training process was carried out with a varying diffusion coefficient (D), ranging from 0.5 to 1.5, on the space domain $[0,1]$, with promising results. The network was trained for 1,100 epochs, leveraging a dependency on the parameter (μ) within the local sublayer.

Key Experiments and Issues: In addition to the advection-diffusion equation, we revisited the Helmholtz equation, this time with a fixed wave number ($k = 8$). The training involved varying the standard deviation (std) of the Fourier features used in the network, which significantly influenced the convergence speed and accuracy. After extensive experimentation, it was concluded that a higher std value was more effective for capturing solutions with higher frequencies.

The figure below illustrate the unfruitful training for the Helmholtz equation with a fixed wave number of 8 and a standard deviation of 0.1:

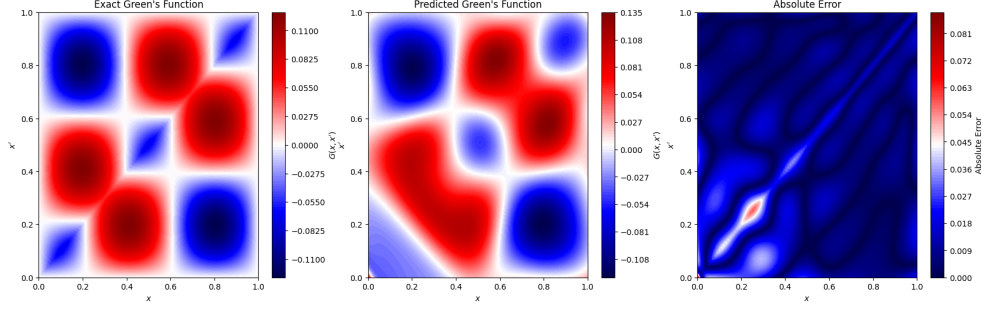


Figure 4.3: Unfruitful training for the Helmholtz equation with a fixed wave number of 8 and a standard deviation of 0.1

The training was parametered in the same way as the successful training for the Helmholtz (using a standard deviation of 2.0), as showcased in figure 5.14.

Outcomes: Training with a varying diffusion coefficient and the Helmholtz equation yielded promising results, improving the network's performance in capturing parametric problem solutions. Initially, training for the Helmholtz equation was unsatisfactory until we increased the standard deviation (std) of the Fourier features. Extensive experimentation revealed that a higher std value improved convergence speed and accuracy for higher frequency solutions.

4.7.8 Final Refinements and Report Writing

From July 16th to August 6th, 2024: By early August, the project entered its final phase, focusing on refining the neural network architecture and preparing the final report. The architecture was optimized by transitioning from a "butterfly" shape with layers [64, 32, 16, 32, 64] to a more uniform "sausage" shape with layers [50, 50, 50, 50].

Key Experiments and Issues: This change significantly improved training performance. The results from this optimized architecture were compared with those from the previous one, highlighting the "sausage" shape's effectiveness in achieving lower error rates and more accurate approximations of the Green's function. During this period, we implemented several key adjustments based on discussions with Antoine. We added a method to calculate the Mean Squared Error (MSE) between the exact and predicted Green's functions at each epoch, providing an evaluation metric. Additionally, we increased the number of layers from 4 to 5 (all with 50 nodes) to better compare with the previous "butterfly" architecture: [50, 50, 50, 50, 50] versus [64, 32, 16, 32, 64]. We also opted for a batch size of up to 1000/2000, provided the GPU memory did not saturate (previously it was 20, which could have induced excess stochasticity in training). Furthermore, I tested an alternative solver to the finite difference method, the `solve_bvp` function from the SciPy library, which provided significantly better results for our 1D PDEs defined by boundary conditions. I also cleaned and committed the code for `green_kernel_x`, `local_sublayer_x`, `kernel_sublayer_x`, and `deep_operator_x`.

Outcomes: The optimized "sausage" architecture demonstrated superior performance, achieving lower error rates and more accurate approximations. The implementation of MSE calculations, increased layer count, and adjusted batch size contributed to these improvements. The use of the `solve_bvp` function further enhanced the results. This period marked the final refinement of our neural network, culminating in the preparation of the final report.

4.8 Conclusion

In this chapter, we presented the architecture used to learn the Green's functions and we detailed the data-sampling process and the implementation history throughout the internship. The implementation history was divided into several phases, each focusing on different aspects of the project, from setting up the environment to refining the neural network architecture.

Moving forward, we will now analyze the performance of the Monte Carlo convolution and Gaussian processes, as well as evaluate the order of convergence of the finite difference solver. These analyses will provide further insights into the capabilities and limitations of the neural network approach in solving PDEs.

Chapter 5

Final Numerical Results and Analysis

5.1 Introduction

This chapter presents the numerical results obtained from training the neural network to learn the Green's function for various PDEs. The results showcase the network's ability to accurately predict the Green's function, enabling the solution of the underlying PDEs with high precision. The experiments conducted cover a range of PDEs, including the Laplace equation, advection-diffusion equation, and Helmholtz equation, demonstrating the versatility and effectiveness of the neural operator approach.

Additionally, we provide results regarding the Monte Carlo convolution, which align with theoretical expectations. We also analyze the performance of Gaussian processes and their corresponding solutions. Furthermore, we evaluate the order of convergence of the finite difference solver. Finally, we present the final results for each test case, highlighting the accuracy and precision achieved by the neural network.

Remark 5.1.1. *All the benchmark-related results were performed on my personal computer, equipped with an Intel Core i5-6600K CPU and an NVIDIA GeForce GTX 1660 Super GPU. The RAM was 16 GB DDR4, and the operating system was Ubuntu 22.04.*

5.2 Highlighting the theoretical expectations of the Monte Carlo convolution

5.2.1 Introduction

As mentioned earlier, I implemented a class called `kernel_sublayer` that computes the convolution of the Green's kernel with the source terms using Monte Carlo integration. This method approximates the convolution integral by generating random samples, denoted as x , from the space domain, named collocation, on which the solution u is evaluated. Additionally, other points, denoted as y , are sampled from the same domain. The corresponding source terms, denoted as $f(y)$, and the kernel provided by the module named `green_kernel_x`, denoted as $K(x, y)$, are used to perform the Monte Carlo approximation of the convolution.

The mathematical notation the Monte-Carlo approximation of the convolution is given by:

$$\begin{aligned}
u_{\text{pred}}(x; \mu) &= \int_{\Omega} K(x, y) f(y; \mu) dy \\
&\approx \frac{\text{vol}(\Omega)}{N} \sum_{i=1}^N K(x, y_i) f(y_i; \mu),
\end{aligned}$$

where N is the number of sensor points, x is a given collocation point, y_i are the sensor points, and $\text{vol}(\Omega)$ is the volume of the domain Ω .

Moreover, the theoretical convergence rate for the Monte Carlo approximation of the convolution is expected to be $\mathcal{O}(N^{-1/2})$, where N is the number of sensor points.

5.2.2 The test case used

Preliminary data

The number of sensor points start from 10 and increase incrementally, initially by 5s and 10s, then by 50s, 100s, 500s, and eventually by larger increments up to 100,000.

The source term and kernel used

The space domain is defined as $\Omega = [0, 2]$, and the source term is defined by $f(y; \mu) = y \exp(y)$.

The kernel is given by $K(x, y) = x \cdot y$, which represents the dot product of the vector containing the collocation points x_i and the sensor points y_i .

The sampler used for the Monte Carlo integration

We tested the method on both the PyTorch uniform sampler and the Scimba uniform sampler. The PyTorch uniform sampler is used to generate random samples from a uniform distribution, while the Scimba uniform sampler is used to generate random samples from a uniform distribution using numpy.

I ran one simulation for each sampler to compare the results.

5.2.3 Results and Analysis

The results of the Monte Carlo convolution approximation are presented in the following figure.

The figure consists of three subplots that provide a detailed comparison between two sampling methods: Scimba sampling and Torch sampling.

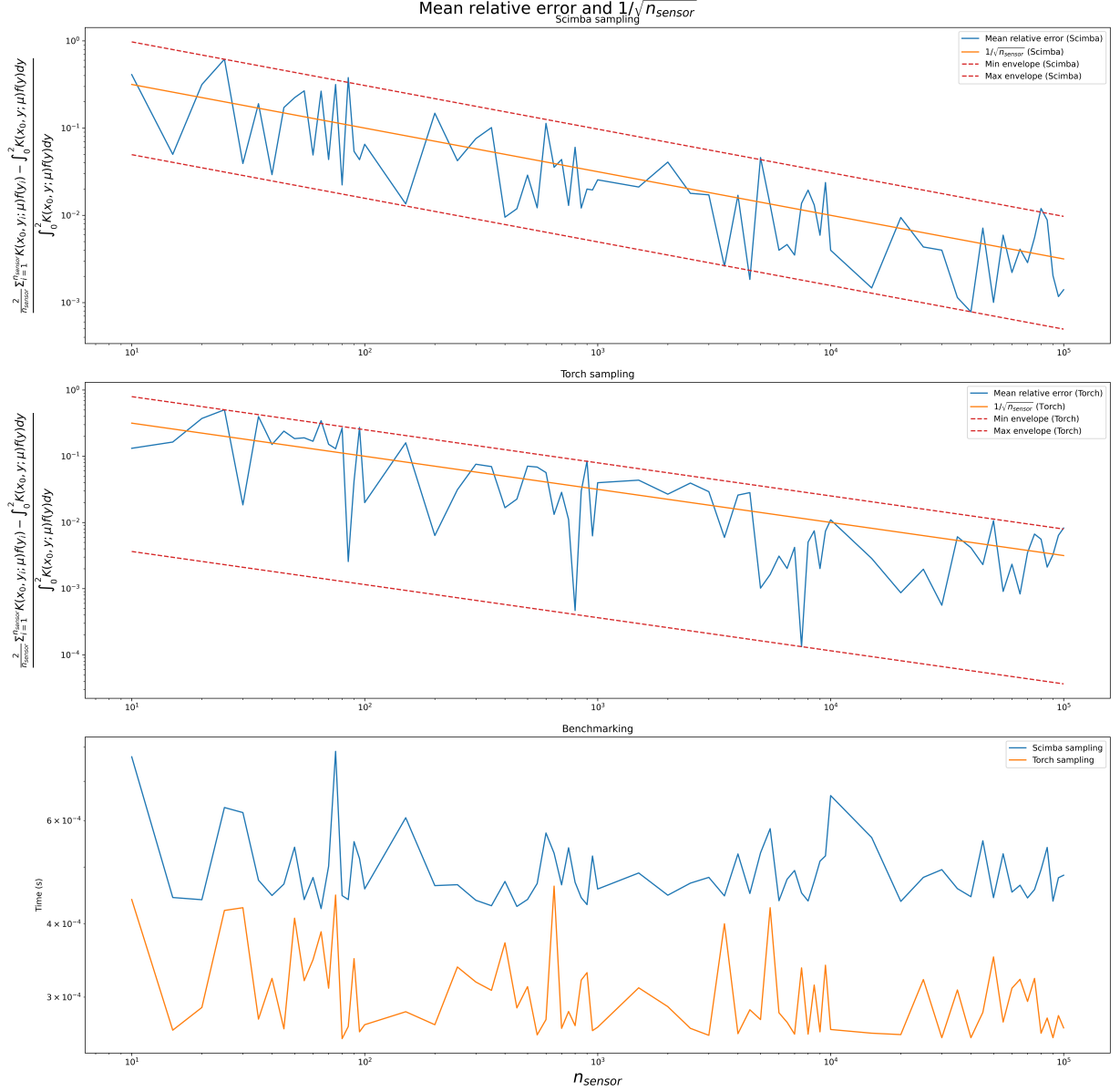


Figure 5.1: Mean relative error on the Monte-Carlo convolution and Benchmarking for Scimba and Torch sampling

Analysis: The top and middle subplots indicate that both Scimba and Torch sampling methods show a reduction in mean relative error with increasing N_{sensor} , consistent with theoretical expectations. Both methods' errors fluctuate around the expected $1/\sqrt{N_{\text{sensor}}}$ trend, with some variability, as indicated by the error envelopes. The bottom subplot shows that both methods are computationally similar, with slightly different performance at various points, but generally following the same trend in time as N_{sensor} increases. In conclusion, the figure illustrates that both Scimba and Torch sampling methods are effective in reducing mean relative error as the number of sensor points increases, aligning with the theoretical convergence rate. The computational costs, as measured by time, are also comparable between the two methods, making both viable depending on specific requirements or preferences. The error and benchmarking analyses

provide a comprehensive view of the performance and efficiency of these sampling techniques in the context of Monte Carlo convolution approximations.

5.3 Plots of Gaussian Processes and Solutions

5.3.1 Introduction

In this section, we present the results of the Gaussian processes used to generate the source terms and the corresponding solutions. The Gaussian processes were characterized by the Scipy RBF solver, which provided diverse and stochastic source terms for training the neural network. The solutions were computed using a finite difference method, ensuring high accuracy and consistency. The plots showcase the variability and complexity of the source terms and solutions, highlighting the effectiveness of the data generation process.

5.3.2 The configuration and the test cases involved

As mentioned in Section 4.5, we generated source terms using random samplings from RBF kernel-related Gaussian processes with a length-scale parameter of $\ell = 0.03$. The source terms were sampled at 100 points within the space domain $\Omega = [0, 1]$, providing a diverse set of excitations for the PDEs. The corresponding solutions were computed using a finite difference method, ensuring high accuracy and consistency.

The three test cases considered were the Laplace equation, the advection-diffusion equation, and the Helmholtz equation, each with unique source terms and solutions.

5.3.3 Results and Analysis

The Laplace Equation

Problem Setup and Plots In our problem setup, the Laplace equation is defined as $-\Delta u = f$ on the interval $[0, 1]$, where u is the unknown function and f is the source term. The boundary conditions are given by $u(0) = u(1) = 0$.

Below are the plots of the Gaussian process samples for the source terms and the corresponding solutions for the Laplace equation. We sampled 100 source terms and the space domain $\Omega = [0, 1]$ was discretized into 200 points.

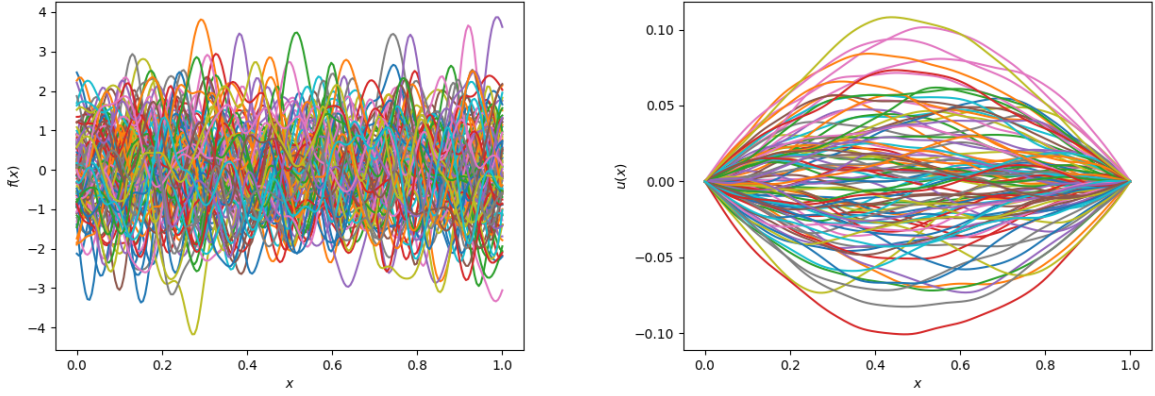


Figure 5.2: Gaussian Processes (RBF) as source terms for the Laplace equation and the corresponding solutions obtained using the SciPy BVP solver

Analysis The plots demonstrate the effectiveness of the data generation process for the Laplace equation. The source terms generated by the Gaussian processes exhibit variability and stochasticity, while the solutions obtained using the SciPy BVP solver accurately capture the behavior of the equation. The diverse source terms provide a robust training dataset for the neural network, enabling it to effectively learn the underlying Green's function and homogeneous solution. The solutions also respect the boundary conditions $u(0) = 0$ and $u(1) = 0$.

The Advection-Diffusion Equation

Problem Setup and Plots In our problem setup, the advection-diffusion equation is defined as $Du'' + 2u' + u = f$ on the interval $[0, 1]$, where u is the unknown function, f is the source term, and D is the diffusion coefficient. The boundary conditions are given by $u(0) = 1$ and $u(1) = -2$.

Below are the plots of the Gaussian process samples for the source terms and the corresponding solutions for the Laplace equation. We also sampled 100 source terms and the space domain $\Omega = [0, 1]$ was discretized into 200 points. We fixed the diffusion coefficient $D = 1$ for this test case.

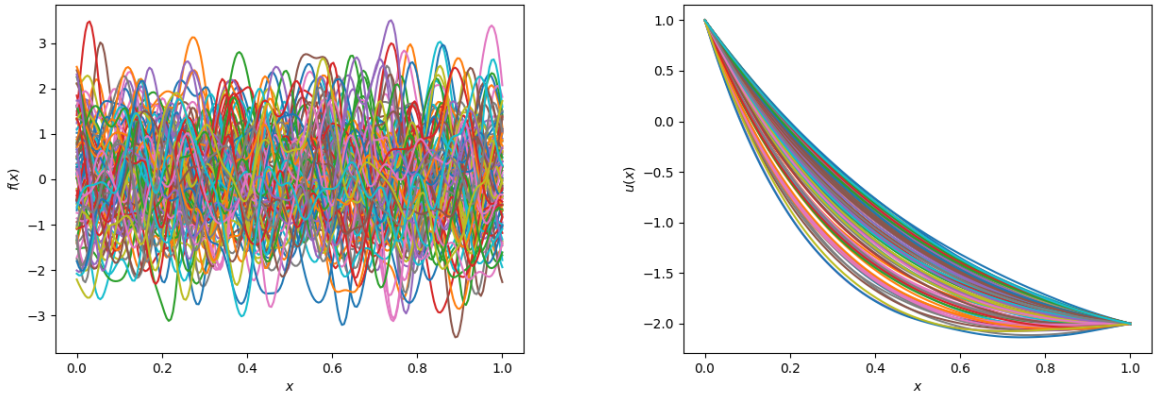


Figure 5.3: Gaussian Processes (RBF) as source terms for the Advection-Diffusion equation and the corresponding solutions obtained using the SciPy BVP solver

Analysis The plots highlight the efficacy of the data generation methodology for the advection-diffusion equation. The source terms exhibit substantial variability and complexity, effectively capturing the diverse excitations intrinsic to the system. The solutions, computed via the SciPy BVP solver, precisely replicate the equation's dynamics, incorporating a diffusion coefficient $D = 1$ and adhering to the boundary conditions $u(0) = 1$ and $u(1) = -2$.

Problem Setup and Plots In our problem setup, the Laplace equation is defined as $\Delta u + k^2 u = f$ on the interval $[0, 1]$, where u is the unknown function, f is the source term, and k is the wave number. The boundary conditions are given by $u(0) = 0$ and $u(1) = 0$.

Below are the plots of the Gaussian process samples for the source terms and the corresponding solutions for the Laplace equation. We again sampled 100 source terms and the space domain $\Omega = [0, 1]$ was discretized into 200 points. The wave number $k = 8$ was fixed for this test case.

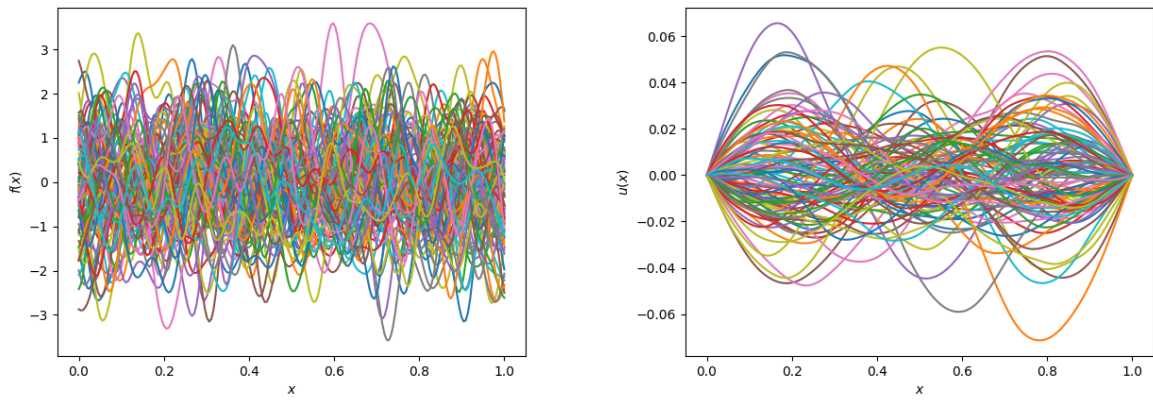


Figure 5.4: Gaussian Processes (RBF) as source terms for the Helmholtz equation and the corresponding solutions obtained using the SciPy BVP solver

Analysis The same can be observed for the Helmholtz equation, where the Gaussian processes generate diverse and stochastic source terms, capturing the system's excitations effectively. The solutions, computed using the SciPy BVP solver, accurately represent the equation's behavior, incorporating a fixed wave number $k = 8$ and satisfying the boundary conditions $u(0) = 0$ and $u(1) = 0$. Let us add that the solutions are oscillatory, as expected for the Helmholtz equation.

5.4 Order of Convergence of the Finite Difference Solver

5.4.1 Introduction

In this section, we analyze the order of convergence of the finite difference solver used to compute the solutions for the PDEs. The order of convergence provides insights into the accuracy and consistency of the solver, indicating how the error decreases as the grid size is refined. By evaluating the convergence rate, we can assess the solver's performance and reliability in solving PDEs.

The order of convergence is typically determined by computing the error between the numerical solution obtained with a given grid size and the reference solution. The error is then plotted against the grid size, and the slope of the resulting curve provides the order of convergence.

When it comes to the Laplace equation, the order of convergence is expected to be second-order, as the finite difference method is known to exhibit second-order convergence for this type of equation.

5.4.2 The test case used

For the order of convergence analysis, we considered the Laplace equation as the test case. The Laplace equation is defined as $-\Delta u = f$ on the interval $[0, 2]$, where u is the unknown function and f is the source term. The boundary conditions are given by $u(0) = 0$ and $u(2) = 0$.

The reference solution was defined as $u(x; \mu) = \sin(2\pi x) \cdot \exp\left(-\frac{(x-1)^2}{0.05\mu}\right)$, where μ is a parameter. We fixed the parameter μ to 0.5 for this test case.

The grid size was varied from 1,000 to 16,000 points (double the previous grid size each time), and the corresponding solutions were computed using the finite difference solver. Then, the L2 error between the numerical solution and the reference solution was calculated for each grid size.

5.4.3 Results and Analysis

Results: The figure is a subplot of two plots. The first plots shows the reference solution and the numerical solution obtained with the finite difference solver for $N = 16,000$ points. The second plot shows the error between the numerical solution and the reference solution for different grid sizes. The scale is logarithmic, with the grid size on the x-axis and the L2 error on the y-axis.

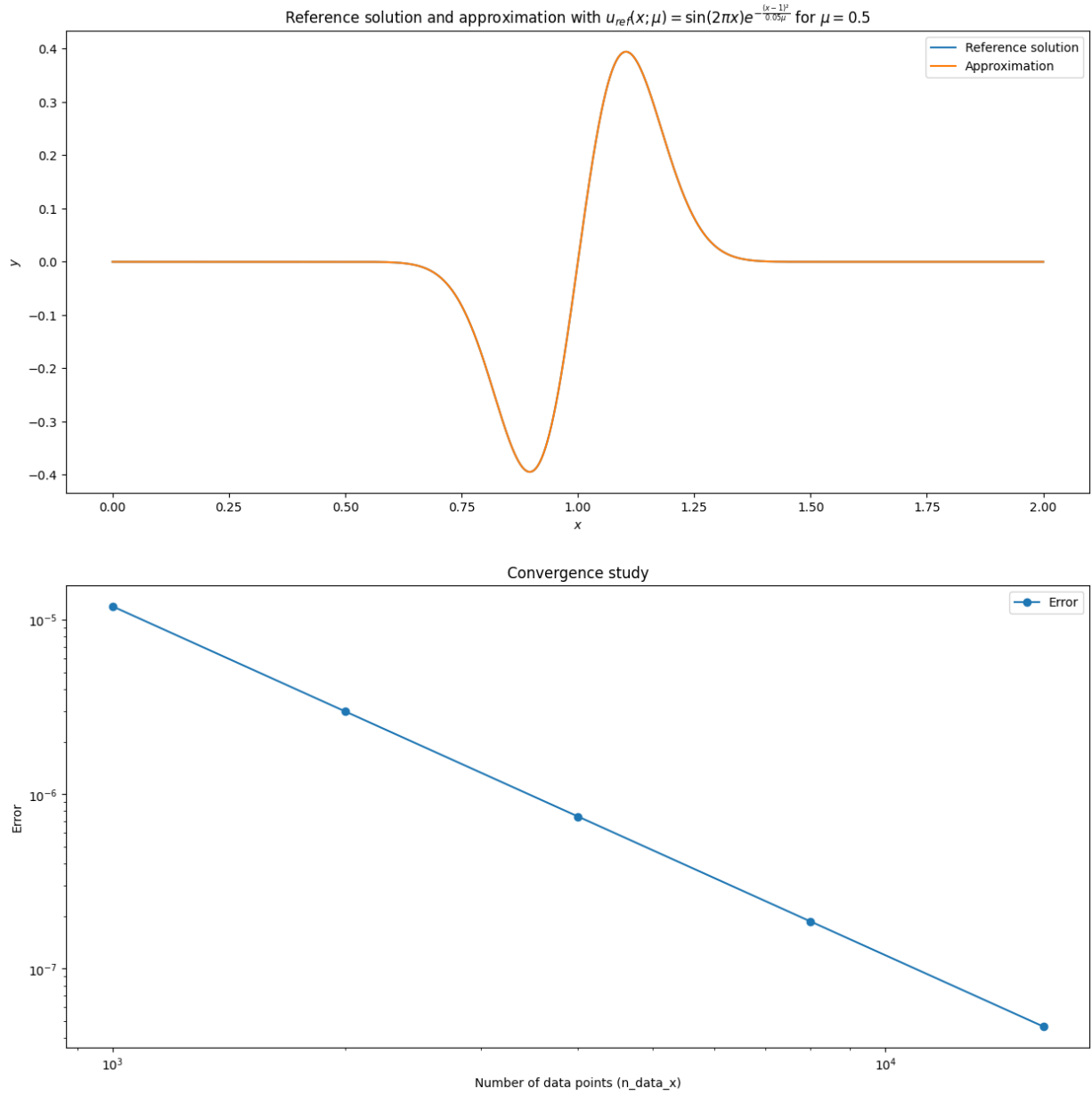


Figure 5.5: Order of Convergence for the Laplace Equation

Then the convergence orders are stored in a table as follows:

Number of Points	Order of Convergence
1000	
2000	2.0015
4000	2.0007
8000	2.0004
16000	2.0002

Table 5.1: Order of convergence for different numbers of points

Analysis: The finite difference method applied to the Laplace equation on the interval $[0, 2]$ with zero Dirichlet boundary conditions has been analyzed and shown to exhibit second-order convergence, as expected. The comparison between the reference solution and the finite difference approximation demonstrates a high degree of accuracy, with the approximation closely aligning with the reference solution, indicating minimal error. The convergence study further supports this, showing a linear relationship on a log-log scale with a slope of approximately -2, confirming a second-order convergence rate. Additionally, the numerical data in the table corroborates this finding, with the order of convergence consistently close to 2 across various grid sizes. These results collectively verify that the finite difference method has been correctly implemented and achieves the expected level of accuracy for this problem.

5.5 Final Results for the Laplace Equation

5.5.1 Introduction

In this section, we present the final results obtained for the Laplace equation using the neural network approach. The Laplace equation is defined as $-\Delta u = f$ on the interval $[0, 1]$, where u is the unknown function and f is the source term. The boundary conditions are given by $u(0) = 0$ and $u(1) = 0$. The neural network was trained to learn the Green's function for the Laplace equation, enabling the solution of the PDE with high precision.

The results showcase the accuracy and effectiveness of the neural network in capturing the essential characteristics of the Laplace equation. By learning the Green's function and homogeneous solution, the network provides accurate predictions and enables the solution of the PDE for various source terms.

5.5.2 The configuration and the test cases involved

Neural Network Configuration	Value
Fourier Features	70
Physical Parameter	None
Layer Format	[50, 50, 50, 50, 50]
Local Layer Activation	Yes
Local Layer Format	[50, 50, 50, 50, 50]
Optimizer	Adam (100 iterations) + LBFGS (2400 iterations)
Batch Size (Collocation and sensor points)	200
Gaussian Sampling	RBF Kernel with $\ell = 0.03$
Standard Deviation of Fourier Features	1.0
Number of Data/Collocation Points (Batch Size)	200
Number of Simulations	100
Activation Function	Rational

Table 5.2: Neural Network Configuration and Test Cases for the Laplace Equation

5.5.3 Results and Analysis

The Green's Function

The figure below shows the predicted Green's function obtained from the neural network for the Laplace equation. The Green's function captures the fundamental behavior of the Laplace equation, providing insights into the system's dynamics and characteristics.

The expression of the exact Green's function for the Laplace equation is provided in the section C.1 in the appendix.

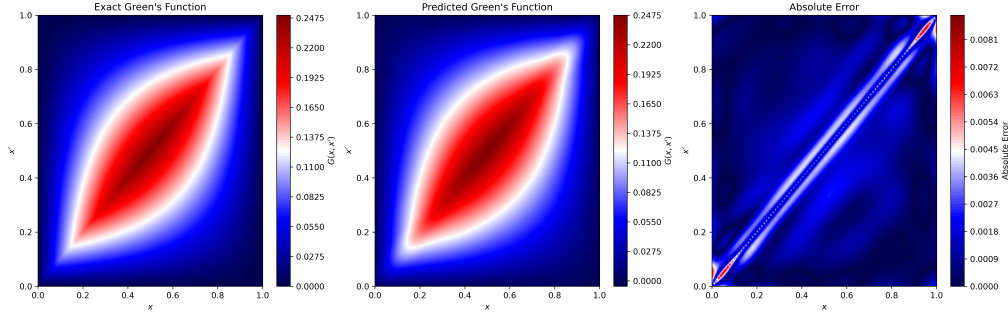


Figure 5.6: Predicted Green's Function for the Laplace Equation

Analysis: The predicted Green's function accurately captures the essential features of the Laplace equation, providing a reliable representation of the system's behavior. The Green's function serves as a fundamental building block for solving the Laplace equation, enabling the neural network to predict the solution with high precision.

5.5.4 Getting the predicted solution on several source terms

The figure below presents the relative error between the predicted solution and the reference solution for Laplace's equation across different source terms. The errors are analyzed as functions of the space domain, ranging from 0 to 1, with a resolution of 0.01.

Remark 5.5.1. *The source terms used for the statistical analysis below can be found in the appendix in section D.*

The corresponding reference solutions were computed using the SciPy BVP solver on a grid of 100 points in the space domain $[0, 1]$.

The space relative errors were computed using the formula:

$$\text{Relative Error} = \frac{|u_{\text{pred}} - u_{\text{ref}}|}{|u_{\text{ref}}|} \times 100,$$

where u_{pred} is the predicted solution, u_{ref} is the reference solution, and $|\cdot|$ denotes the absolute value.

The space domain $[0, 1]$ was composed of 100 points, and the relative errors were computed at each point to analyze the model's performance across the domain.

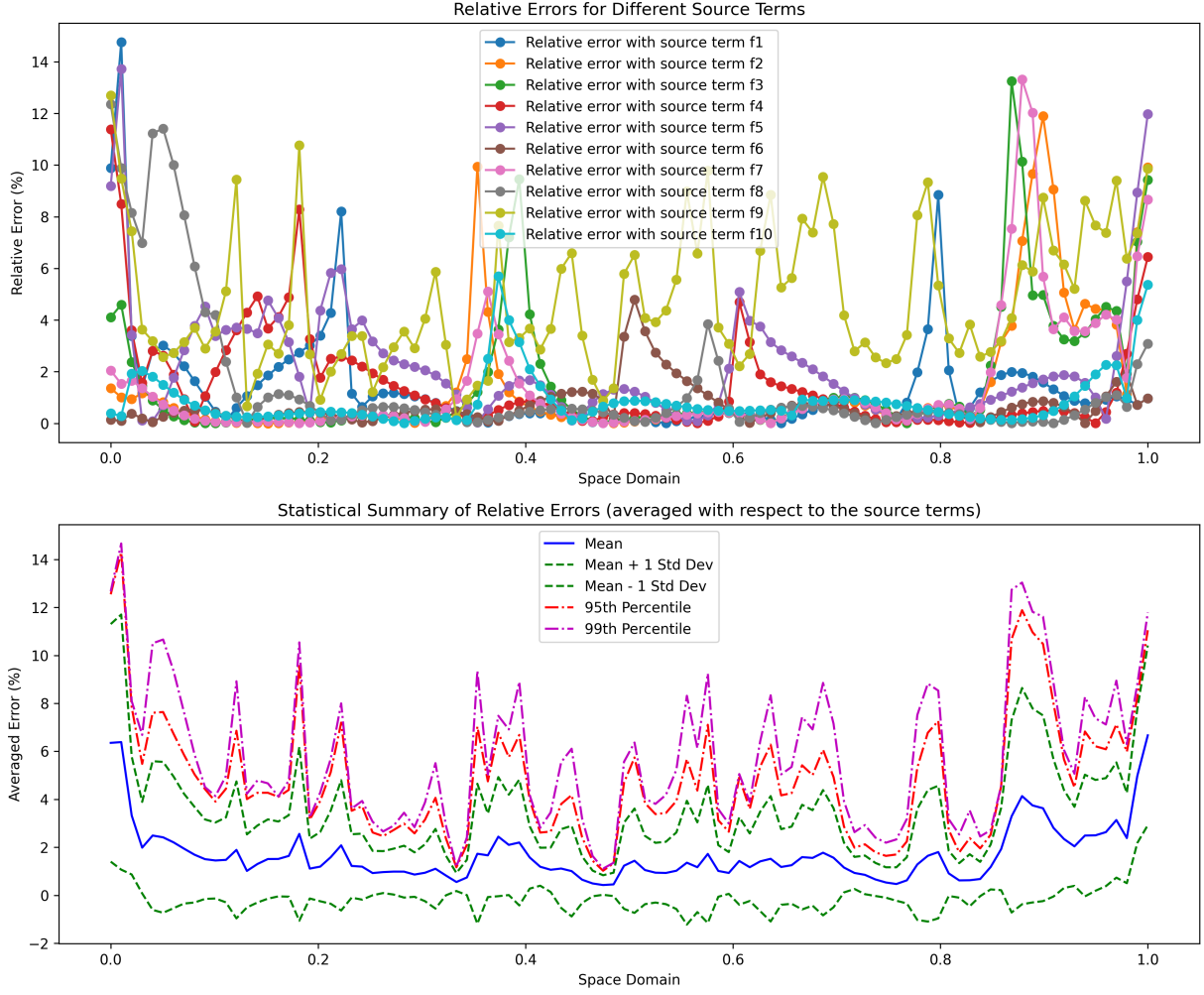


Figure 5.7: Analysis of relative errors for different source terms in the Laplace equation as a function of the space domain.

Analysis: The plot analyzes relative errors across different source terms for the Laplace equation over the space domain, revealing that while the model performs consistently with low mean errors and variability in the central region of the domain, there are significant spikes in errors near the boundaries (around 0 and 1). These spikes indicate regions where the model's predictions are less reliable, suggesting that the model's accuracy diminishes near the domain boundaries, likely due to challenges in handling boundary conditions. This pattern underscores the need for potential refinements to enhance the model's robustness, particularly in these critical regions.

5.5.5 The convergence of the loss function

The figure below shows the convergence of the loss function during training for the Laplace equation. The loss function decreases rapidly during the initial epochs and stabilizes over time, indicating that the network has effectively learned the Green's function.

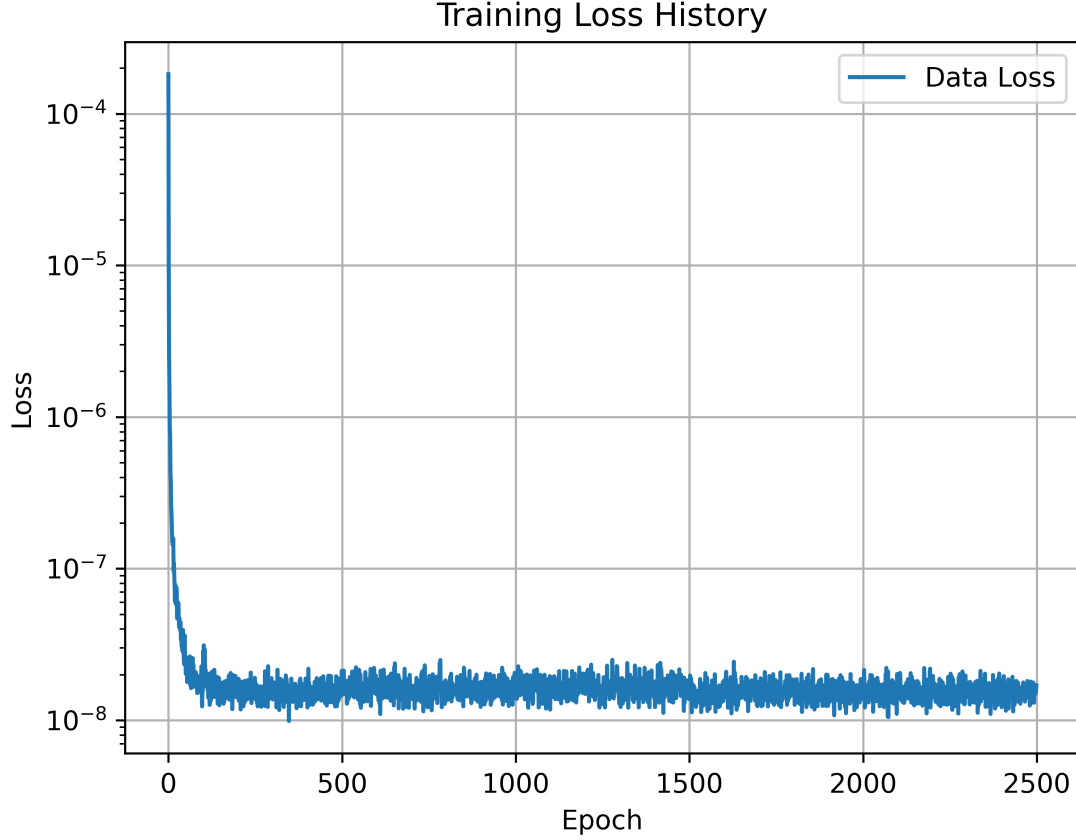


Figure 5.8: Convergence of the Loss Function for the Laplace Equation

5.6 Final Results for the Advection-Diffusion Equation

5.6.1 Introduction

The advection-diffusion equation is given by $Du'' + 2u' + u = f$ over the interval $[0, 1]$, where u is the unknown function, f is the source term, and D is the diffusion coefficient. The boundary conditions are specified as $u(0) = 1$ and $u(1) = -2$. The neural network was trained to approximate the Green's function for the advection-diffusion equation, allowing for precise solutions of the PDE.

The results demonstrate the neural network's ability to accurately capture the key features of the advection-diffusion equation. By learning the Green's function and the homogeneous solution, the network delivers precise predictions and facilitates solving the PDE for various source terms.

Here, the relevancy of such a test case lies in the fact that the boundary conditions are non-homogeneous. Furthermore, the solution of the homogeneous equation with such boundary conditions is not trivial.

5.6.2 The configuration and the test cases involved

Neural Network Configuration	Value
Fourier Features	70
Physical Parameter	$\mu = D$ (varies between 0.5 and 1.5)
Layer Format	[50, 50, 50, 50, 50]
Local Layer Activation	Yes
Local Layer Format	[50, 50, 50, 50, 50]
Optimizer	Adam (100 iterations) + LBFGS (1000 iterations)
Batch Size (Collocation and sensor points)	200
Gaussian Sampling	RBF Kernel with $\ell = 0.03$
Standard Deviation of Fourier Features	1.0
Number of Data/Collocation Points (Batch Size)	200
Number of Simulations	100
Activation Function	Rational

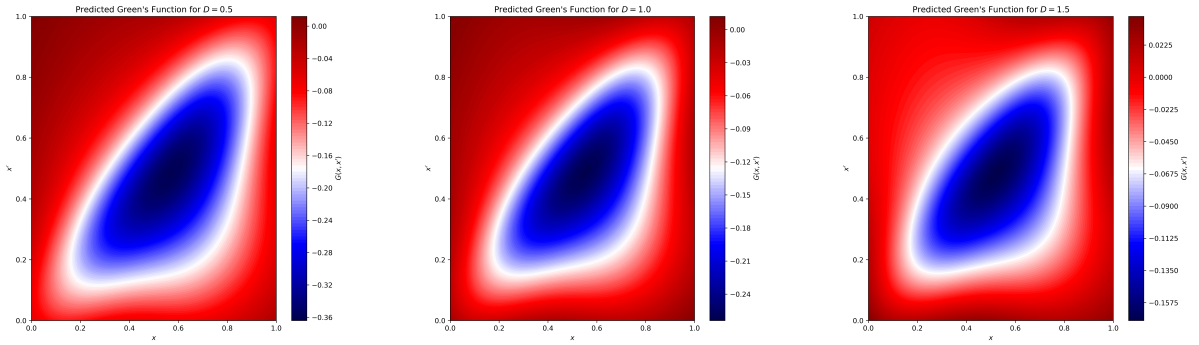
Table 5.3: Neural Network Configuration and Test Cases

5.6.3 Results and Analysis

The Green's Function

The figure below shows the predicted Green's function obtained from the neural network for the advection-diffusion equation. The Green's function captures the essential characteristics of the advection-diffusion equation, providing insights into the system's dynamics and behavior.

Let us note that the exact Green's function for the advection-diffusion equation has not been provided because it cannot be understood under the framework of the variation of parameters method, in section 2.3.1.



(a) Predicted Green's Function for $D = 0.5$

(b) Predicted Green's Function for $D = 1.0$

(c) Predicted Green's Function for $D = 1.5$

Analysis: The predicted Green's function accurately captures the essential features of the advection-diffusion equation for different diffusion coefficients. The Green's function provides valuable insights into the system's dynamics and behavior, enabling the neural network to solve the PDE with high precision. The results showcase the network's ability to learn the Green's function and ho-

mogeneous solution, facilitating the solution of the advection-diffusion equation for various source terms.

5.6.4 The homogeneous solution

Remark 5.6.1. *The reference homogeneous solutions were computed using the SciPy BVP solver on a grid of 100 points in the space domain $[0, 1]$.*

Statistical Analysis of the Relative Errors with respect to the space domain

The figure below illustrates the spatial relative error between the predicted solution and the reference solution for the advection-diffusion equation, evaluated across a range of diffusion coefficients D from 0.5 to 1.5. The top plot depicts the spatial distribution of relative errors for each value of D , showing how errors vary within the spatial domain. The bottom plot provides a statistical summary of these errors, averaged over all values of D , and includes the mean, standard deviation, and high-percentile values, offering insights into the overall accuracy and stability of the numerical method across different diffusion coefficients. This analysis helps identify regions within the spatial domain where the numerical method is most and least accurate, highlighting potential areas for improvement.

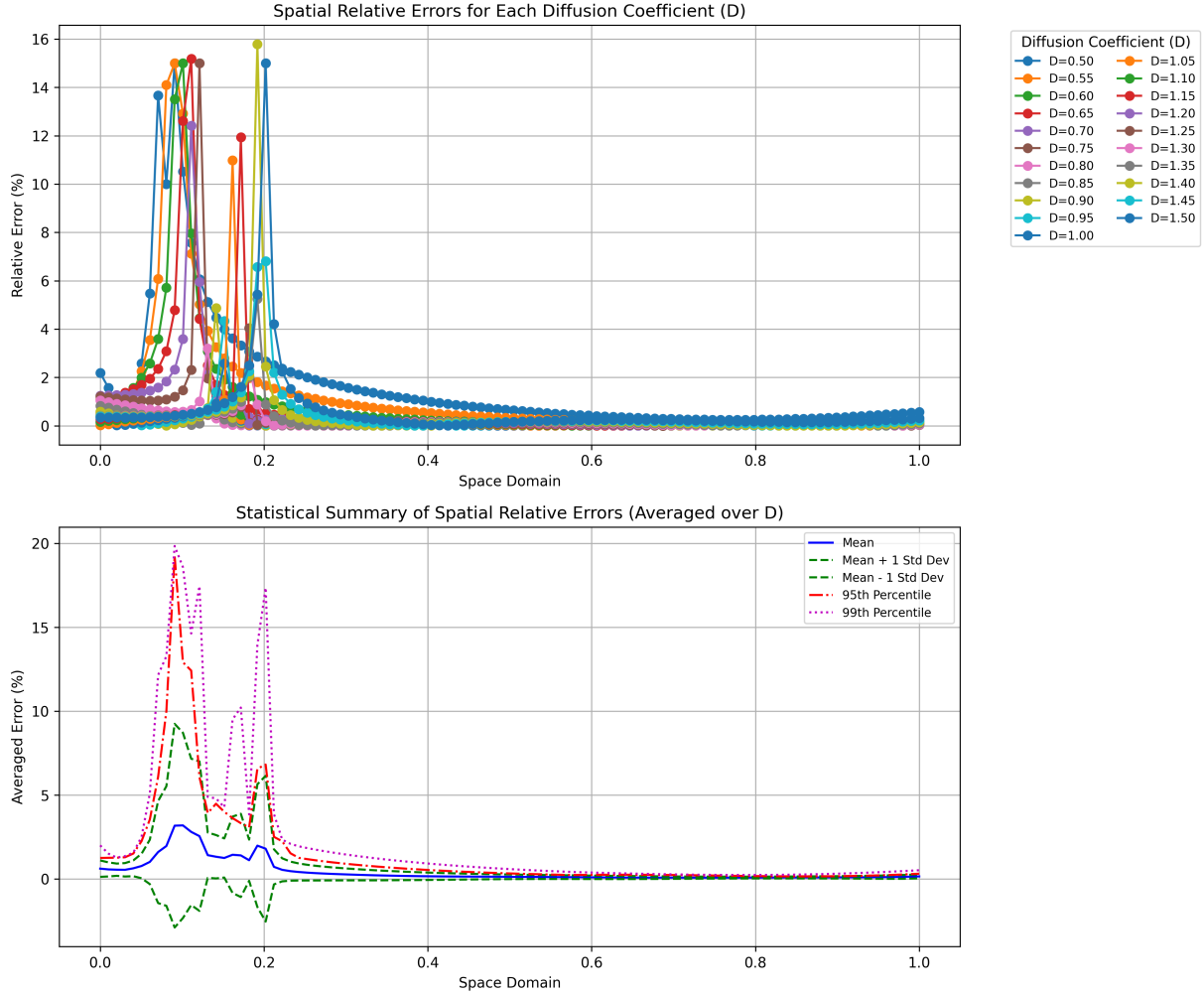


Figure 5.10: Analysis of relative errors for different source terms in the parametric Advection-Diffusion equation as a function of the space domain

Analysis: The figure provides a detailed analysis of the spatial relative errors in solving the advection-diffusion equation across various diffusion coefficients D ranging from 0.5 to 1.5. In the top plot, the relative errors for each D are plotted as functions of the spatial domain, revealing significant error peaks near the beginning of the domain, likely due to boundary effects or the influence of initial conditions. These errors decrease sharply as the spatial domain progresses, suggesting that the numerical method becomes more accurate away from the boundaries or initial step gradients. The plot indicates that lower diffusion coefficients are associated with higher error peaks, which dissipate more slowly, highlighting the critical role of diffusion in smoothing out errors, particularly near boundaries. In contrast, higher diffusion coefficients are more effective at reducing errors across the domain, leading to more stable and accurate solutions. The bottom plot complements this analysis by providing a statistical summary of these spatial relative errors, averaged over all diffusion coefficients. The mean error remains low across most of the domain, with noticeable increases near the beginning, consistent with the individual error plots. The standard deviation and the 95th and 99th percentiles of the errors indicate significant variability near the domain's start, with occasional extreme outliers, particularly in the initial region, underscoring

the challenges the numerical method faces in handling steep gradients or boundary conditions. The large gap between the mean error and the higher percentiles further suggests that, while the method performs well on average, there are instances of substantial error spikes, pointing to potential numerical instability or sensitivity to specific conditions. Overall, the analysis demonstrates that while the numerical method is generally effective, its accuracy is highly dependent on the diffusion coefficient and the spatial location, with the most significant challenges occurring near the boundaries or regions of high gradient. Therefore, improvements could be focused on enhancing boundary condition handling or adopting adaptive methods to better manage steep gradients, thereby reducing the occurrence of large errors and improving overall solution stability.

Statistical Analysis of the Relative Errors with respect to the Diffusion Coefficient

The figure below shows the space-averaged relative error between the predicted and reference solutions for the advection-diffusion equation across various diffusion coefficients D , ranging from 0.5 to 1.5 with a 0.05 step size. The top plot tracks how the average error changes with D , revealing a decrease as D nears 1.0, followed by a slight rise. The bottom plot uses a violin plot to depict the distribution of errors for each D , highlighting the error variability and stability across different diffusion levels.

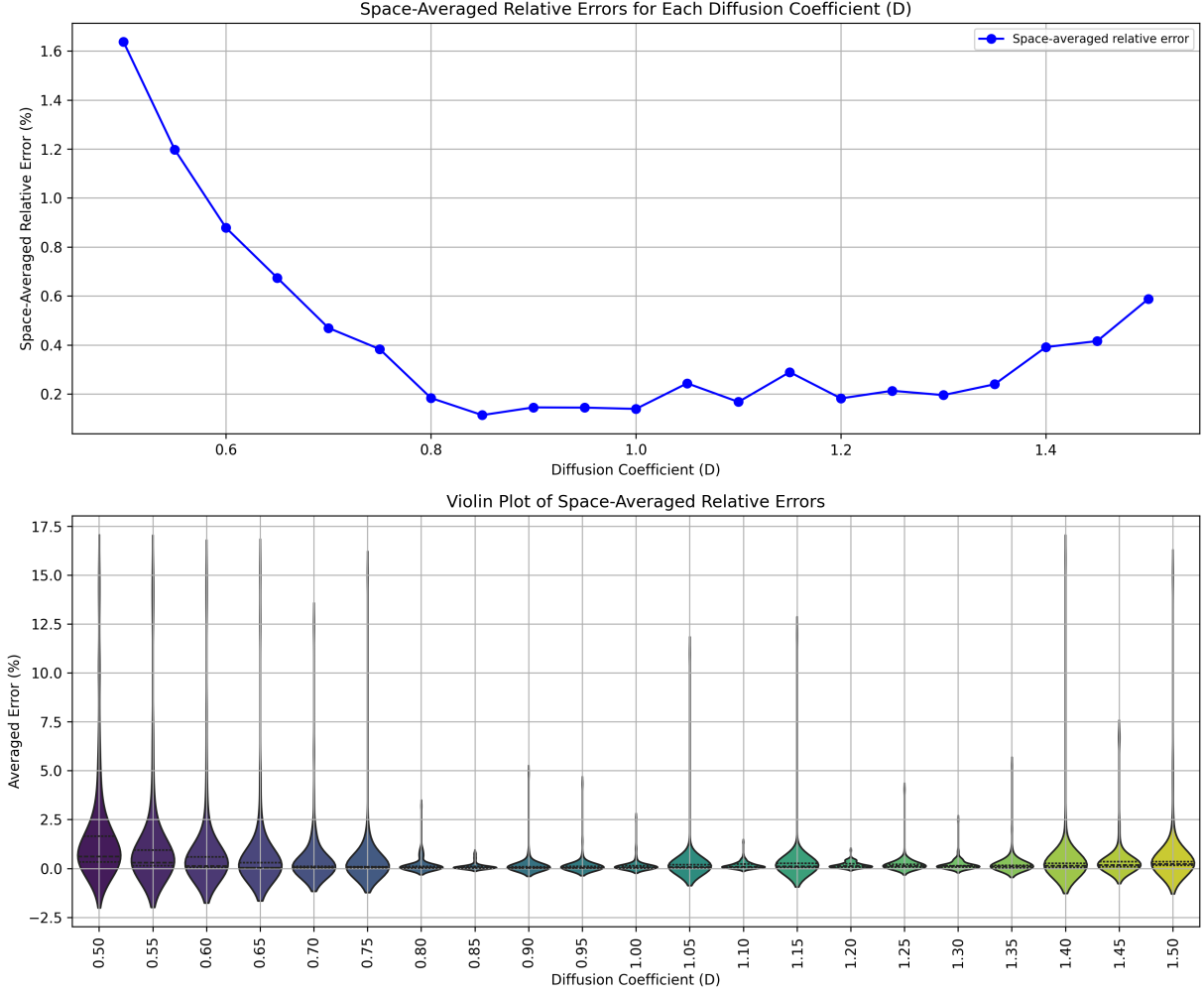


Figure 5.11: Analysis of relative errors for different source terms in the parametric Advection-Diffusion equation as a function of the Diffusion coefficient D

Analysis: The figure presents a thorough analysis of the space-averaged relative errors across various diffusion coefficients D in solving the advection-diffusion equation. The top plot depicts the relationship between the space-averaged relative error and D , showing a clear trend where the error diminishes as D increases from 0.5 to approximately 1.0, suggesting that higher diffusion coefficients enhance the accuracy of the solution. Beyond $D = 1.0$, the error begins to rise again, indicating a non-linear relationship between D and the error, possibly due to over-smoothing or other numerical effects. The bottom plot supports this by using a violin plot to depict the distribution of space-averaged relative errors for each D . This plot shows that for lower D values (closer to 0.5), the error distribution is broader with greater variability and occasional negative errors, implying instability or oscillatory behavior in the numerical method. As D increases, the distribution narrows significantly, indicating more consistent performance, although slight increases in variability are observed again at the highest D values. This dual visualization underscores the importance of selecting an appropriate diffusion coefficient to balance accuracy and stability in the numerical solution, with D values around 1.0 offering the best overall performance based on these error metrics.

5.6.5 Getting the predicted solution on several source terms

Remark 5.6.2. The source terms used for the statistical analysis below can be found in the appendix in section D.

The corresponding reference solutions and the space relative errors were computed in the same way as for the Laplace equation in section 5.5.4.

The figure below presents the relative error between the predicted solution and the reference solution for the advection-diffusion equation across different source terms. The errors are analyzed as functions of the diffusion coefficient D , ranging from 0.5 to 1.5, with a step size of 0.05.

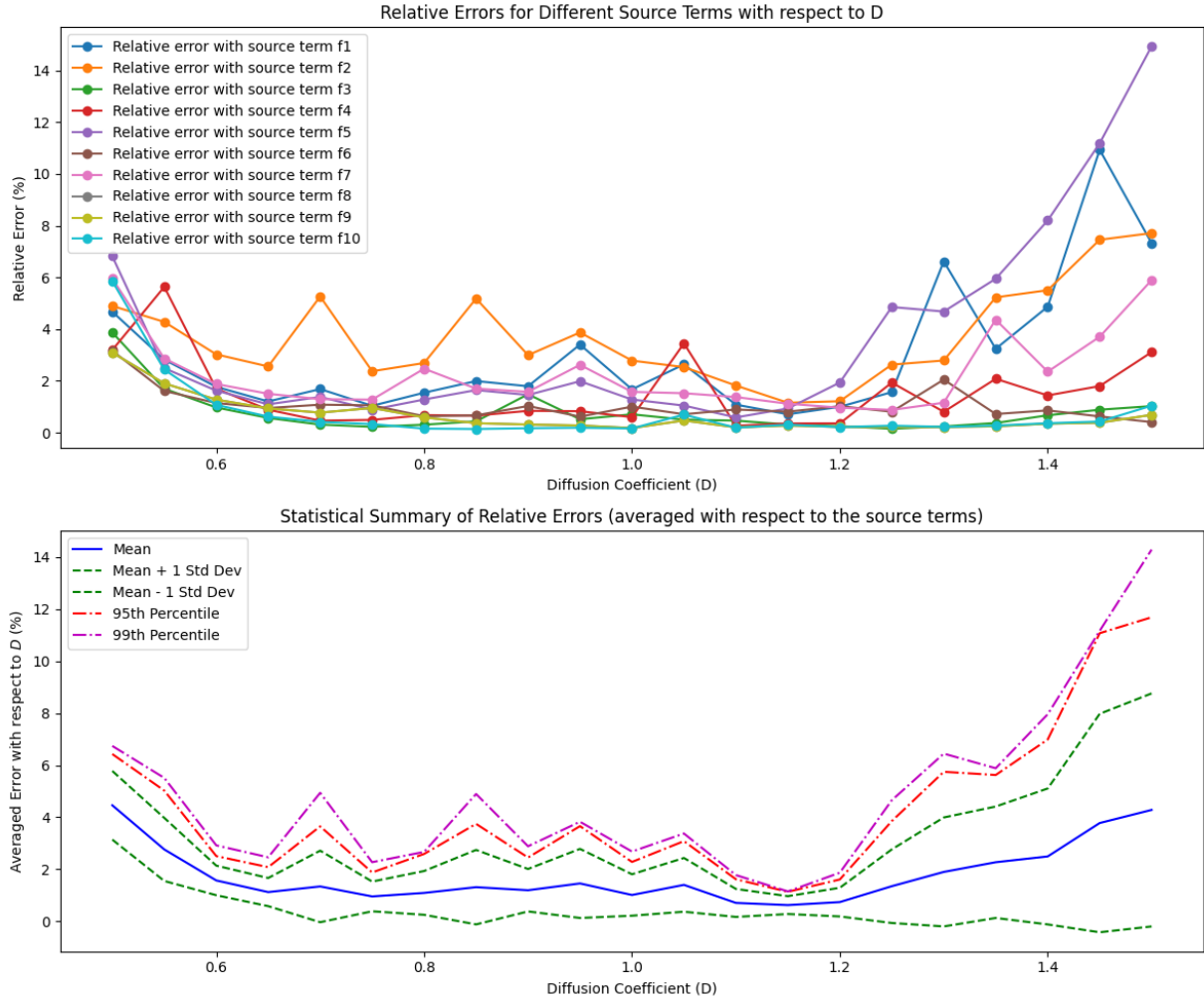


Figure 5.12: Analysis of the relative errors for the overall solution of the Advection-Diffusion Equation

Analysis: The figure illustrates the relative error between the predicted and reference solutions for the advection-diffusion equation across various source terms, analyzed against diffusion coefficients (D) ranging from 0.5 to 1.5. In the top plot, the relative errors for different source terms are depicted, showing a general trend where errors decrease as D increases up to around 1.0, indicating optimal numerical performance within this range. However, beyond $D = 1.2$, the errors begin to rise significantly for certain source terms, particularly f_3 , f_4 , f_8 , and f_{10} , suggesting a decline in

accuracy at higher diffusion coefficients. The lower plot provides a statistical summary, averaging the relative errors over all source terms, with the mean error demonstrating a reduction between $D = 0.6$ and $D = 1.0$, followed by an upward trend as D increases further. The standard deviation mirrors this pattern, with lower error variability around $D = 1.0$ and greater variability at higher D values. The 95th and 99th percentiles indicate that while the mean error remains relatively low, there are substantial outliers at higher D values, where some source terms exhibit markedly higher errors. This comprehensive analysis suggests that the numerical method is most reliable within the D range of 0.6 to 1.0, but its performance deteriorates outside this range, particularly at higher diffusion coefficients, where both error magnitude and variability increase, highlighting the need for careful consideration when applying the method to cases with larger D values to avoid significant inaccuracies.

5.6.6 The convergence of the loss function

The figure below shows the convergence of the loss function during training for the advection-diffusion equation. The loss function decreases rapidly during the initial epochs and stabilizes over time, indicating that the network has effectively learned the Green's function.

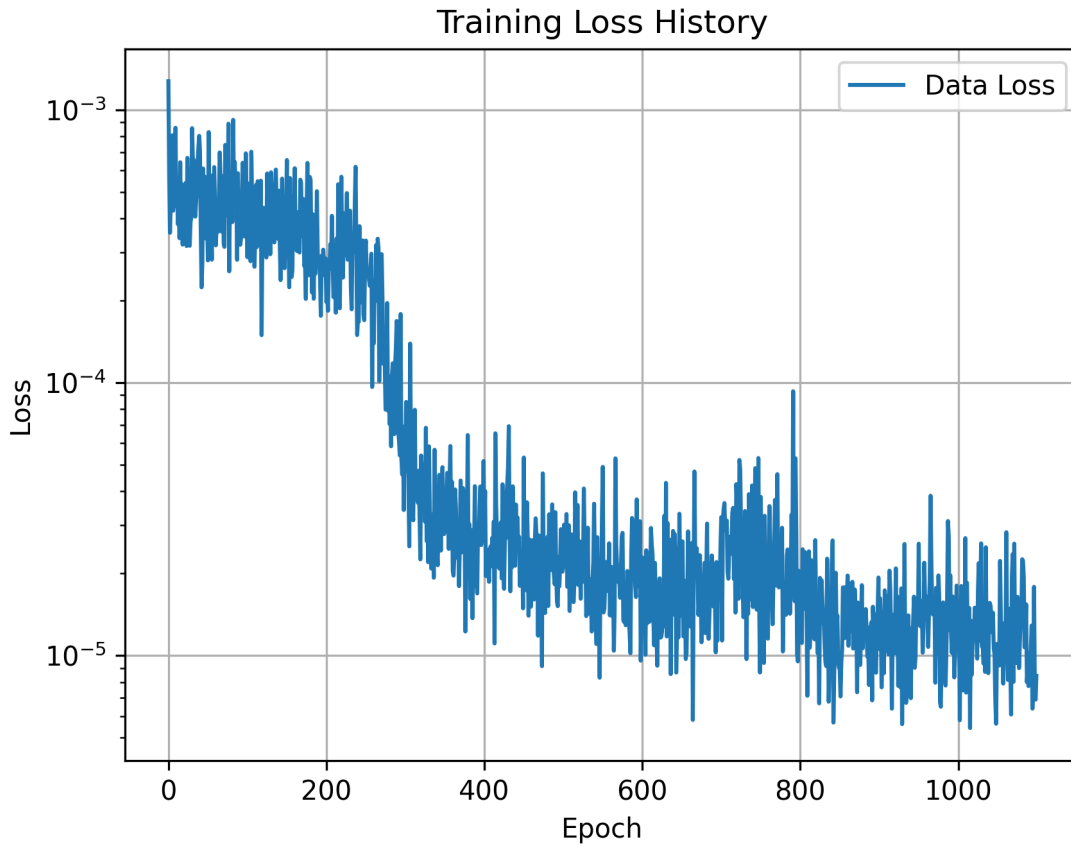


Figure 5.13: Convergence of the Loss Function for the Advection-Diffusion Equation

5.7 Final Results for the Helmholtz Equation with a fixed wavenumber

5.7.1 Introduction

Let us introduce the final results for the Helmholtz equation with a fixed wave number using a neural network-based approach. The Helmholtz equation is formulated as $\Delta u + k^2 u = f$ over the interval $[0, 1]$, where u is the unknown function, f is the source term, and k is the wave number. The boundary conditions are specified as $u(0) = 0$ and $u(1) = 0$. The neural network was trained to approximate the Green's function for the Helmholtz equation, enabling precise solutions of the PDE.

The wavenumber k was fixed at 8 for this test case, and the results demonstrate the neural network's ability to accurately capture the essential characteristics of the Helmholtz equation. By learning the Green's function and homogeneous solution, the network provides accurate predictions and enables the solution of the PDE for various source terms.

5.7.2 The configuration and the test cases involved

Neural Network Configuration	Value
Fourier Features	70
Physical Parameter	$\mu = k = 8$
Layer Format	[50, 50, 50, 50, 50]
Local Layer Activation	Yes
Local Layer Format	[50, 50, 50, 50, 50]
Optimizer	Adam (100 iterations) + LBFGS (2400 iterations)
Batch Size (Collocation and sensor points)	200
Gaussian Sampling	RBF Kernel with $\ell = 0.03$
Standard Deviation of Fourier Features	2.0
Number of Data/Collocation Points (Batch Size)	200
Number of Simulations	100
Activation Function	Rational

Table 5.4: Neural Network Configuration and Test Cases for the Helmholtz Equation with a Fixed Wavenumber

5.7.3 Results and Analysis

The Green's Function

The figure below shows the predicted Green's function obtained from the neural network for the Helmholtz equation with a fixed wave number $k = 8$. The Green's function captures the essential characteristics of the Helmholtz equation, providing insights into the system's dynamics and behavior.

The expression of the exact Green's function for the Helmholtz equation is provided in the section C.3 in the appendix.

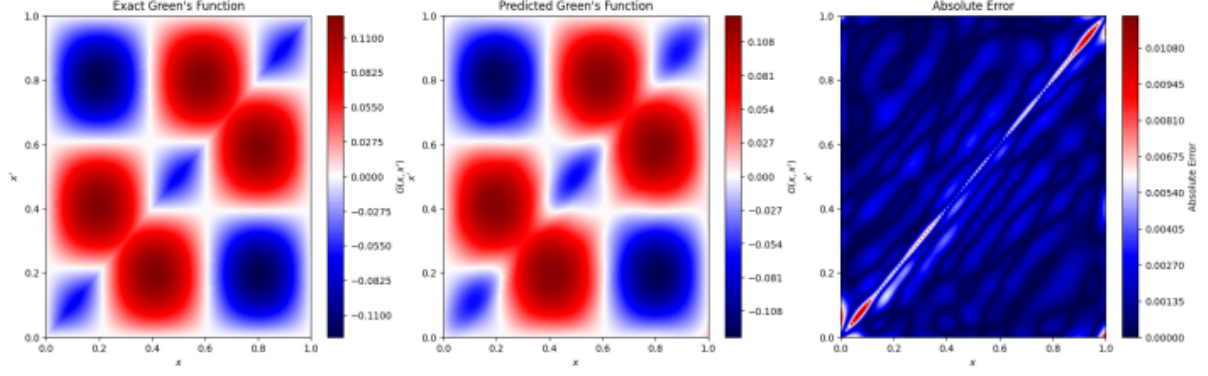


Figure 5.14: Predicted Green's Function for the Helmholtz Equation with $k = 8$

Analysis: The predicted Green's function accurately captures the essential features of the Helmholtz equation with a fixed wave number $k = 8$. The Green's function provides valuable insights into the system's dynamics and behavior, enabling the neural network to solve the PDE with high precision. The results showcase the network's ability to learn the Green's function and homogeneous solution, facilitating the solution of the Helmholtz equation for various source terms.

5.7.4 Getting the predicted solution on several source terms

The figure below presents the relative error between the predicted solution and the reference solution for the Helmholtz equation across different source terms. The errors are analyzed as functions of the space domain, ranging from 0 to 1, with a step size of 0.01.

Remark 5.7.1. *The source terms used for the statistical analysis below can be found in the appendix in section D.*

The corresponding reference solutions and the space relative errors were computed in the same way as for the Laplace equation in section 5.5.4.

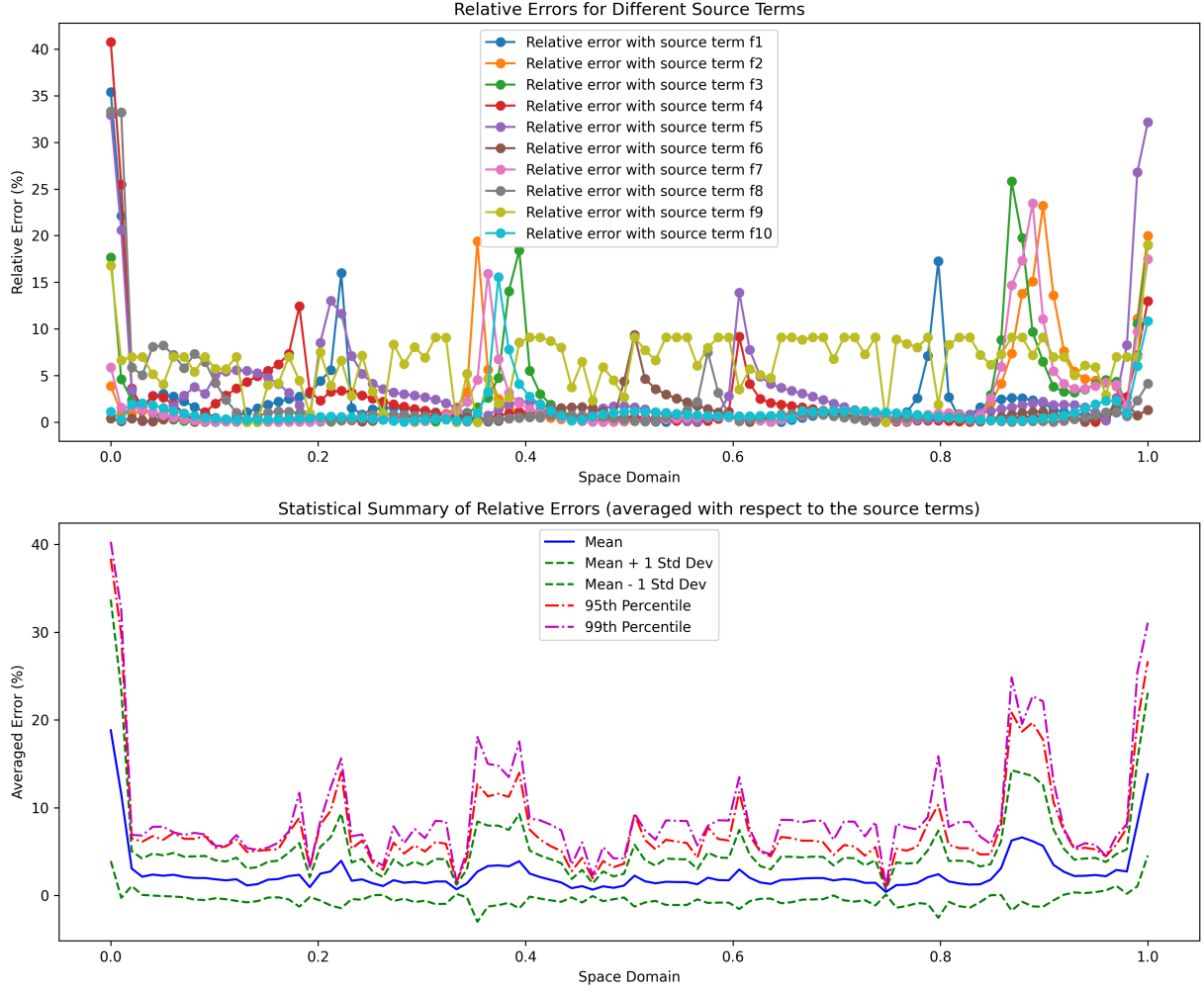


Figure 5.15: Analysis of relative errors for different source terms in the Helmholtz equation as a function of the space domain.

Analysis: The plot provides an analysis of relative errors for various source terms in the context of the Helmholtz equation across the space domain. In the top plot, the relative errors for ten different source terms are displayed, showing noticeable spikes at the domain boundaries (around 0 and 1), indicating that the model's accuracy diminishes near the edges of the domain. The errors in the central region of the space domain are generally lower and more consistent. The bottom plot offers a statistical summary, revealing that the mean error and standard deviation increase significantly near the boundaries, where the 95th and 99th percentiles also highlight the presence of outliers with much higher errors. This suggests that while the model performs adequately in the central part of the domain, there are significant challenges and variability in accuracy at the boundaries, indicating potential areas for refinement to improve overall performance.

The convergence of the loss function

The figure below shows the convergence of the loss function during training for the Helmholtz equation with a fixed wave number $k = 8$. The loss function decreases rapidly during the initial epochs and stabilizes over time, indicating that the network has effectively learned the Green's

function.

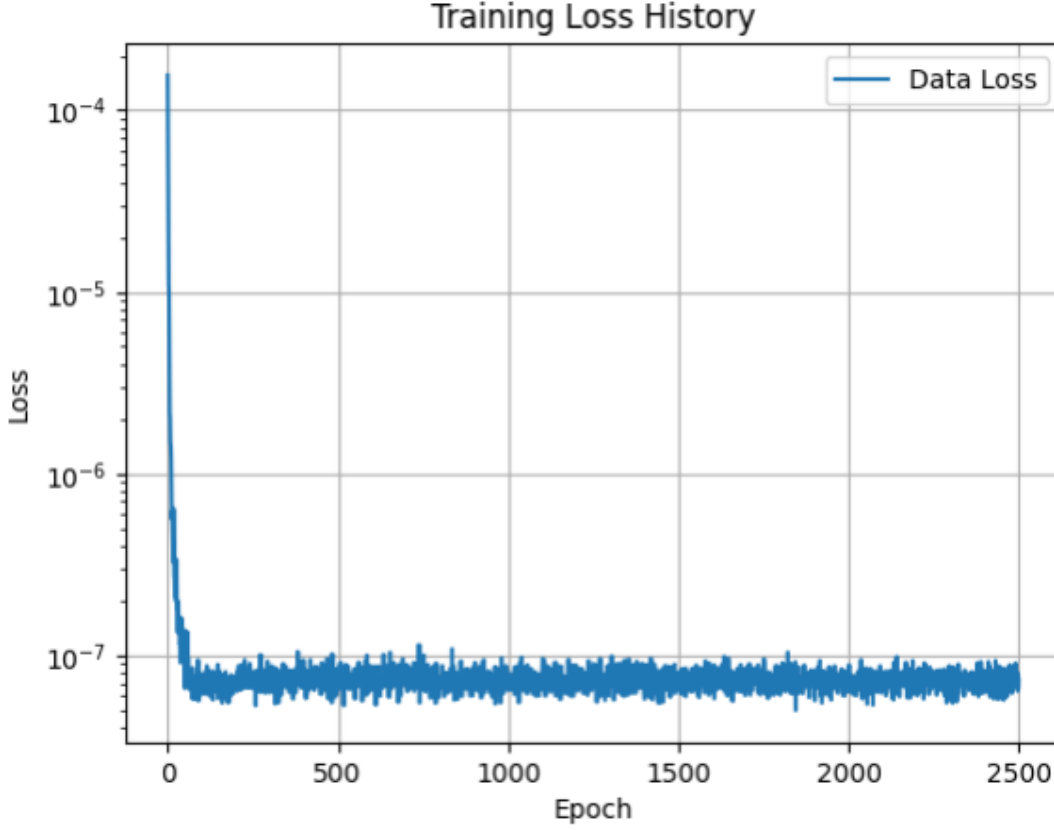


Figure 5.16: Convergence of the Loss Function for the Helmholtz Equation with $k = 8$

5.8 Final results for the Helmholtz equation with a varying wavenumber

5.8.1 Introduction

In this section, we showcase the final results for the Helmholtz equation with a varying wave number using a neural network-based approach. The Helmholtz equation is given by $\Delta u + k^2 u = f$ over the interval $[0, 1]$, where u is the unknown function, f is the source term, and k is the wave number. The boundary conditions are specified as $u(0) = 0$ and $u(1) = 0$. The neural network was trained to approximate the Green's function for the Helmholtz equation, enabling precise solutions of the PDE.

The results highlight the neural network's accuracy and effectiveness in capturing the fundamental characteristics of the Helmholtz equation with a varying wave number. By learning both the Green's function and the homogeneous solution, the network delivers accurate predictions and facilitates the solution of the PDE for various source terms.

5.8.2 The configuration and the test cases involved

Neural Network Configuration	Value
Fourier Features	70
Physical Parameter	$\mu = k$ (varies between 1.0 and 2.0)
Layer Format	[50, 50, 50, 50, 50]
Local Layer Activation	Yes
Local Layer Format	[50, 50, 50, 50, 50]
Optimizer	Adam (100 iterations) + LBFGS (2000 iterations)
Batch Size (Collocation and sensor points)	200
Gaussian Sampling	RBF Kernel with $\ell = 0.03$
Standard Deviation of Fourier Features	2.0
Number of Data/Collocation Points (Batch Size)	200
Number of Simulations	100
Activation Function	Rational

Table 5.5: Neural Network Configuration and Test Cases

5.8.3 Results and Analysis

The Green's Function

The figure below shows the predicted Green's function obtained from the neural network for the Helmholtz equation with a varying wave number. The Green's function captures the essential characteristics of the Helmholtz equation with different wave numbers, providing insights into the system's dynamics and behavior.

For $k = 1$:

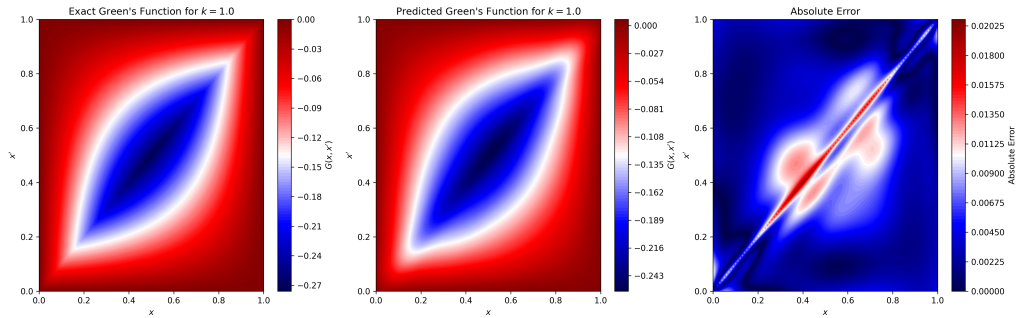


Figure 5.17: Predicted Green's Function for the Helmholtz Equation with $k = 1$

For $k = 1.5$:

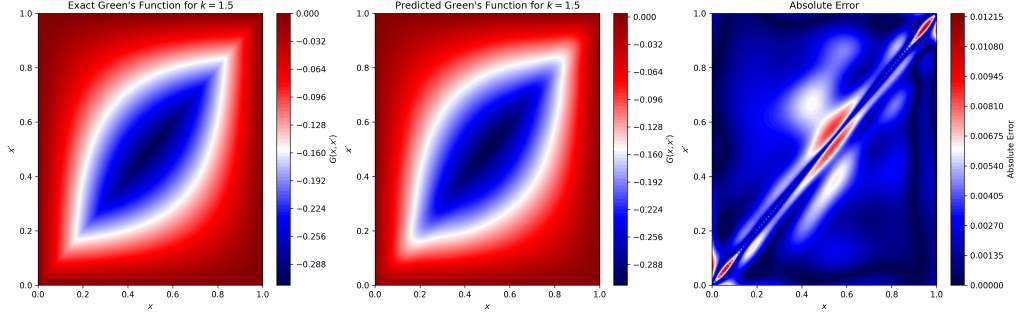


Figure 5.18: Predicted Green's Function for the Helmholtz Equation with $k = 1.5$

For $k = 2$:

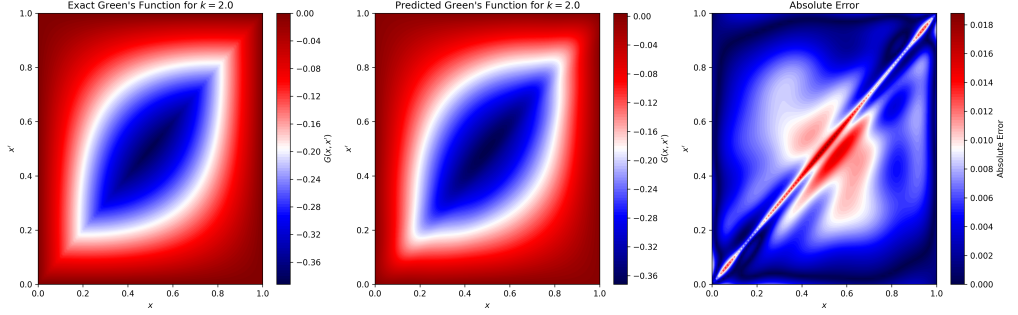


Figure 5.19: Predicted Green's Function for the Helmholtz Equation with $k = 2$

Analysis: The predicted Green's function accurately captures the essential features of the Helmholtz equation with varying wave numbers. The Green's function provides valuable insights into the system's dynamics and behavior, enabling the neural network to solve the PDE with high precision. The results showcase the network's ability to learn the Green's function and homogeneous solution, facilitating the solution of the Helmholtz equation for various source terms.

5.8.4 Getting the predicted solution on several source terms

The figure below presents the relative error between the predicted solution and the reference solution for the parametric Helmholtz equation across different source terms. The errors are analyzed as functions of the wavenumber k , ranging from 1.0 to 2.0, with a step size of 0.05.

Remark 5.8.1. *The source terms used for the statistical analysis below can be found in the appendix in section D.*

The corresponding reference solutions and the space relative errors were computed in the same way as for the Laplace equation in section 5.5.4.

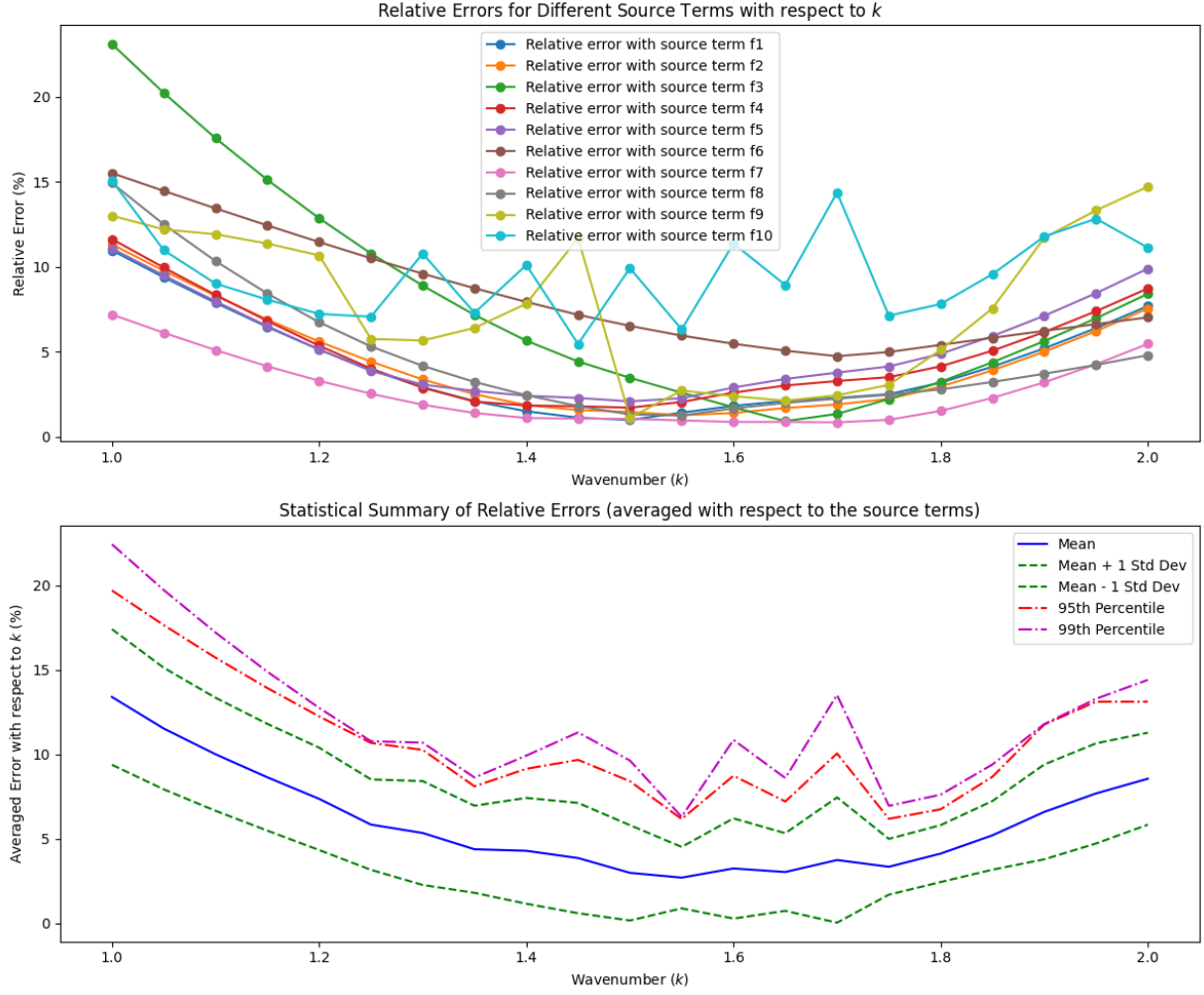


Figure 5.20: Analysis of relative errors for different source terms in the parametric Helmholtz equation as a function of the wavenumber k .

Analysis: The provided plot analyzes the relative errors for different source terms in the context of the parametric Helmholtz equation, revealing that errors generally decrease as the wavenumber k increases from 1.0 to 1.4, with a slight increase observed towards $k = 2.0$. The variability in error curves suggests different sensitivities among source terms to changes in k , with some showing smooth error reduction while others fluctuate, especially at higher wavenumbers. The statistical summary shows that the mean error reaches its minimum around $k = 1.4$, while the variability, indicated by standard deviation, remains stable until this point before increasing. The 95th and 99th percentiles highlight that most errors are low, but there are notable outliers at both ends of the wavenumber spectrum. Overall, the model performs best at intermediate wavenumbers, but certain source terms and wavenumber ranges still present challenges, emphasizing the need to consider both average performance and error variability in such complex parametric PDEs.

The convergence of the loss function

The figure below shows the convergence of the loss function during training for the Helmholtz equation with varying wave numbers. The loss function decreases rapidly during the initial epochs

and stabilizes over time, indicating that the network has effectively learned the Green's function.

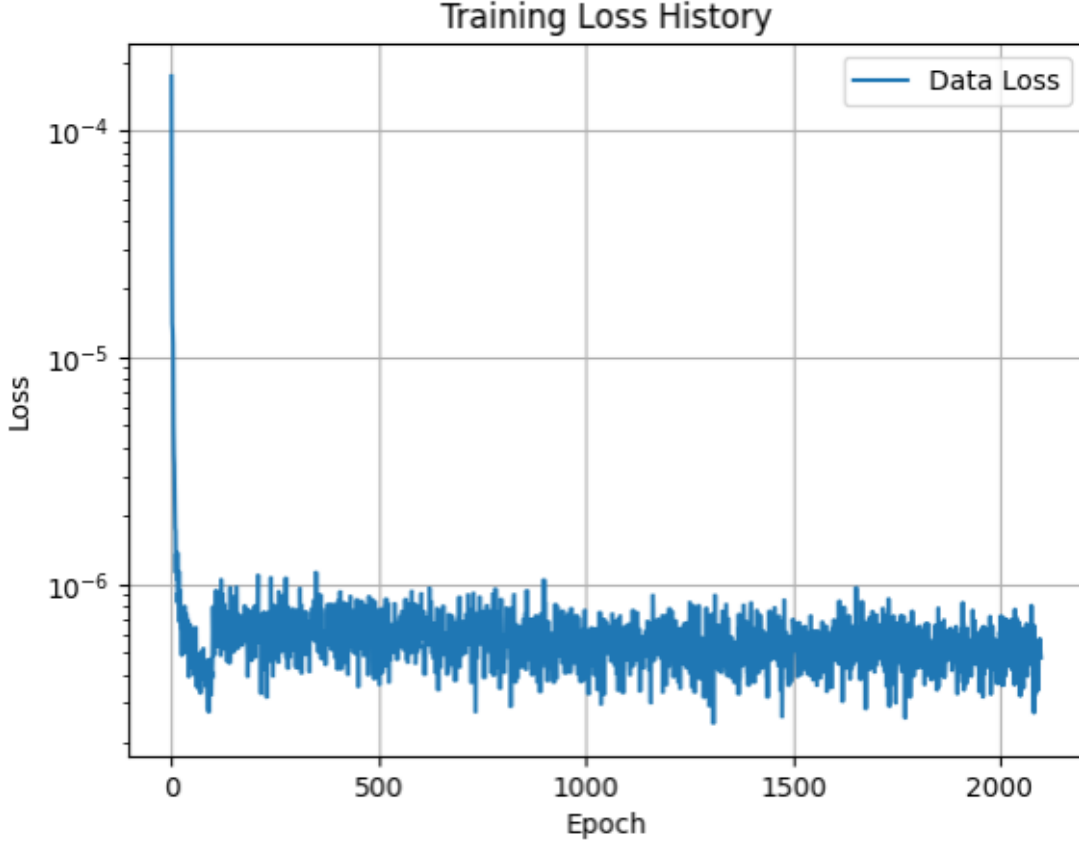


Figure 5.21: Convergence of the Loss Function for the Helmholtz Equation with Varying Wave Numbers

5.9 Performance Comparison: Neural Network vs Finite Differences

5.9.1 Introduction

In this benchmark, we compare the performance of a neural network approach and a classical finite difference method in solving the Laplace equation—a fundamental partial differential equation (PDE) often used as a benchmark in numerical analysis. The neural network employed is the same model previously used to predict the Green function, thereby ensuring consistency across different applications. Both methods were tested under identical boundary conditions, and we evaluated their performance based on CPU time and the accuracy of the solution.

The source term for the Laplace equation was the same to evaluate the expectations of the finite difference method solver. Therefore the source term was $f(x; \mu) = -\frac{d^2}{dx^2} (\sin(2\pi x)) \exp\left(-\frac{(x-1)^2}{0.05\mu}\right)$, where $\mu = 0.5$. The solution was computed on the interval $[0, 1]$ with boundary conditions $u(0) = 0$ and $u(1) = 0$.

5.9.2 Neural Network Approach

We used a neural network to approximate the Green's function for the Laplace equation. The setting was the same as in the neural network that provided the prediction of the Green's function, as showcased in the section 5.5. **The only difference is that we are using the finite difference method to generate the training data (and not the BVP solver from SciPy).**

5.9.3 Finite Differences

The finite difference method is a well-established numerical technique for solving differential equations, including the Laplace equation. This method discretizes the domain and approximates derivatives using difference formulas, transforming the PDE into a system of algebraic equations. These equations are then solved iteratively to find the numerical solution. For this test case, we implemented a finite difference scheme under the same boundary conditions used in the neural network approach, and we measured the CPU time required to solve the system of equations.

To ensure accurate measurements, we performed 100 different computations for each grid size. For the CPU time, we calculated the average of the 100 computations for each grid size. For the error, we calculated the average of the three smallest times obtained from the 100 computations. This approach provides a reliable estimate of the computational efficiency and accuracy of the finite difference method.

5.9.4 Results and Analysis

The figure below shows the comparison of the CPU time and error between the neural network approach and finite differences for the Laplace equation. The neural network approach achieved a lower error with comparable CPU time, demonstrating its effectiveness in solving the PDE.

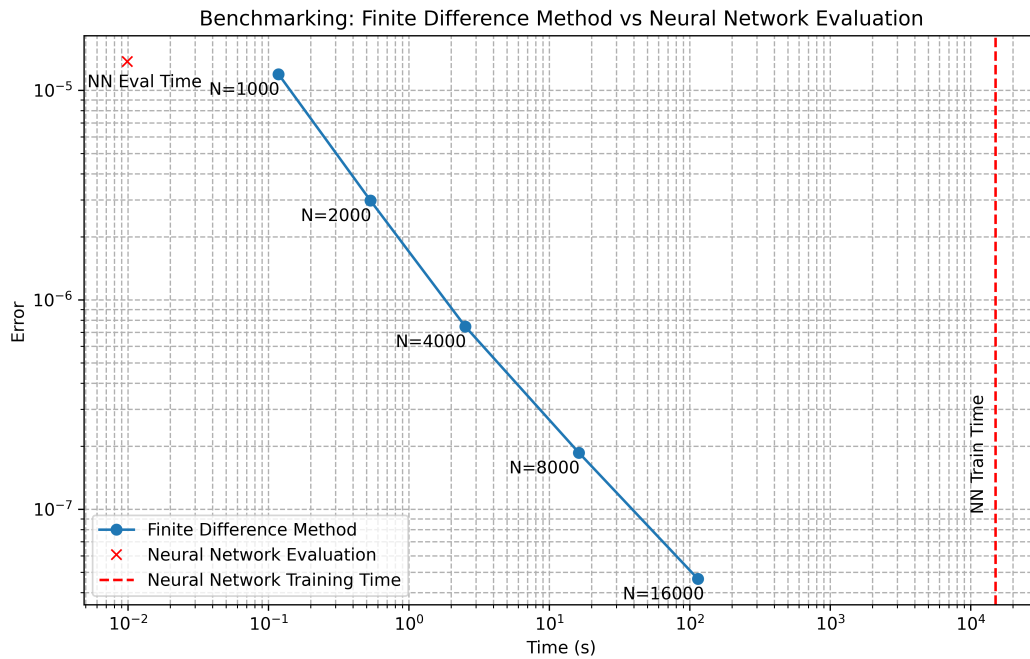


Figure 5.22: Performance Comparison: Neural Network vs Finite Differences

Analysis: The benchmarking plot demonstrates that while the finite difference method achieves lower errors with increased grid size, it does so at a significant computational cost, as indicated by the linear relationship on the log-log scale. In contrast, the neural network evaluation achieves a comparable error with vastly less computation time, making it a highly efficient alternative. This analysis underscores the neural network's potential to provide a favorable balance between accuracy and computational efficiency, particularly in scenarios where reducing computation time is crucial.

In figure 5.22, the neural network (NN) exhibits a clear computational advantage, particularly when running multiple simulations with varying parameters. Training the NN takes around 10,000 seconds, with each evaluation requiring only 0.01 seconds, compared to the finite difference (FD) method, where each simulation may take between 1 to 100 seconds depending on the grid size N .

For instance, at $N = 16,000$, where the FD method takes over 100 seconds per simulation, the NN becomes computationally advantageous after about 100 simulations. This efficiency makes the NN especially suitable for tasks like parameter sweeps or sensitivity analyses, where numerous simulations are needed. The NN's ability to handle extensive simulations quickly highlights its practical benefit in scenarios demanding both accuracy and computational efficiency.

5.10 Unsatisfactory Results for the Parametric Helmholtz Equation in the Vicinity of the Eigenmodes

The homogeneous Helmholtz equation has solutions called eigenmodes, which represent the system's natural oscillation patterns and occur at wavenumbers that are multiples of π . The Green's function, describing the response to a point source, has a singularity at the source point, posing challenges in analysis. Special techniques are needed to handle this singularity in both analytical and numerical solutions.

Let us indeed recall that the exact Green's function for the Helmholtz equation (provided in equation C.3.1) has a singularity.

It is indeed written as:

$$G(x, \xi; k) = \begin{cases} -\frac{\sin(k\xi) \sin(k(1-x))}{k \sin(k)}, & 0 \leq \xi \leq x, \\ -\frac{\sin(kx) \sin(k(1-\xi))}{k \sin(k)}, & x \leq \xi \leq 1, \end{cases}$$

5.10.1 The configuration and the test cases involved

Three different neural networks were used, all with the same settings as the final results for the Helmholtz equation with a varying wavenumber. The parameter training domains for each neural network were as follows:

For the first neural network, the parameter training domain was $[2, 3]$. Then, for the second neural network, the parameter training domain was $[2, 3.1]$. Finally, for the third neural network, the parameter training domain was $[2, 3.14]$.

5.10.2 Results and Analysis

The Green's Function

The figure below shows the predicted Green's function obtained from the neural network for the Helmholtz equation with a varying wave number in the vicinity of the eigenmodes. The

Green's function captures the essential characteristics of the Helmholtz equation with different wave numbers, providing insights into the system's dynamics and behavior.

For $k_{\max} = 3$:

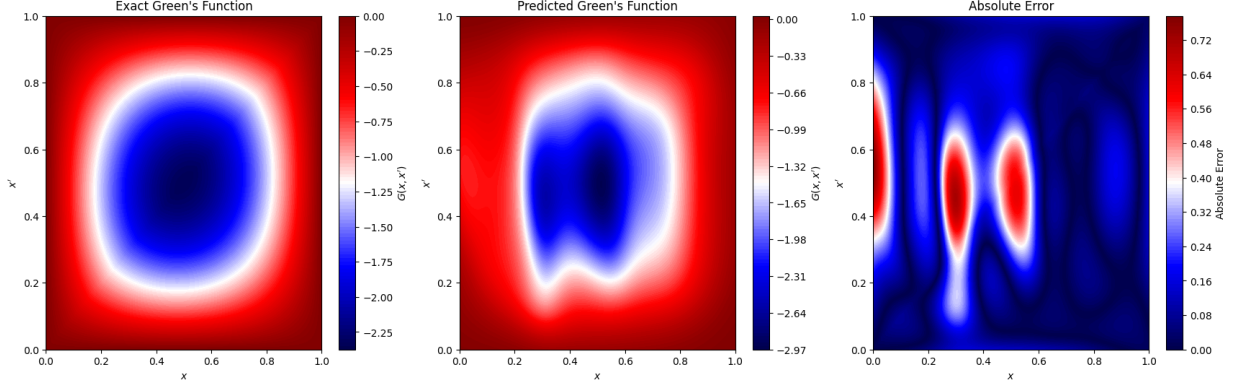


Figure 5.23: Predicted Green's Function for the Helmholtz Equation with $k_{\max} = 3$

For $k_{\max} = 3.1$

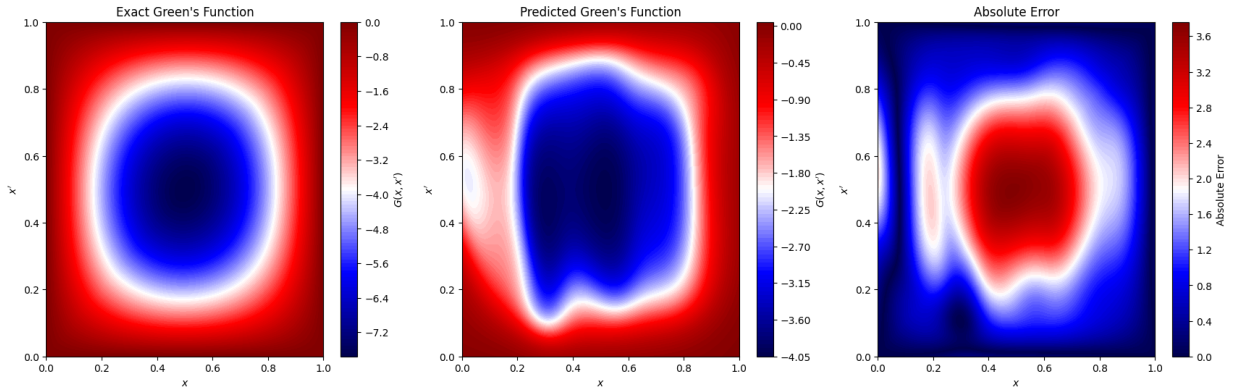


Figure 5.24: Predicted Green's Function for the Helmholtz Equation with $k_{\max} = 3.1$

For $k_{\max} = 3.14$:

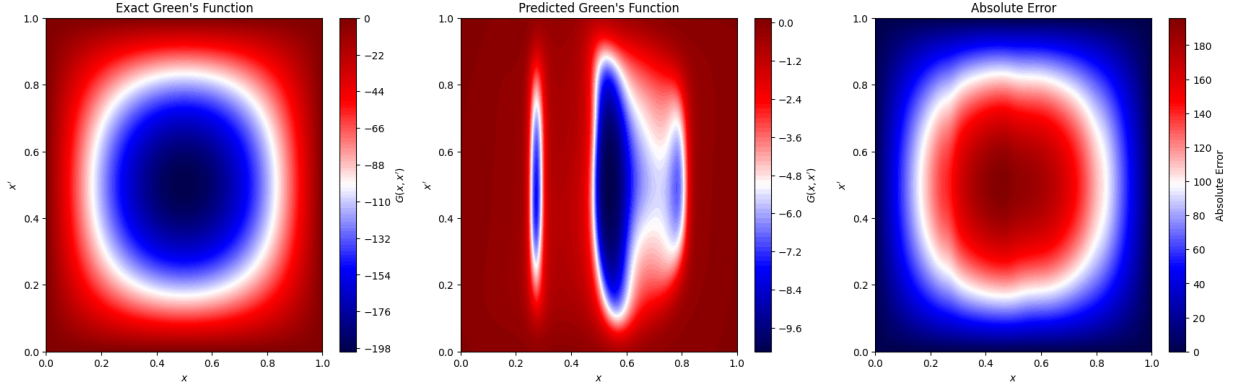


Figure 5.25: Predicted Green's Function for the Helmholtz Equation with $k_{\max} = 3.14$

Analysis: While neural networks are capable of predicting the Green's function for the Helmholtz equation, their reliability deteriorates markedly as the wavenumber approaches critical values tied to eigenmodes. At $k_{\max} = 3$, the network manages to capture the fundamental characteristics with minimal error. However, by $k_{\max} = 3.1$, clear inaccuracies start to surface, indicating the network's increasing struggle to maintain precision. The situation worsens significantly at $k_{\max} = 3.14$, where the network fails to accurately replicate the exact solution, with errors becoming pronounced. This pronounced decline in performance near eigenmodes underscores the severe limitations of current neural network models in handling the heightened sensitivity and complexity in these regions. The findings strongly suggest the need for more robust methods or significantly enhanced neural network architectures to ensure accurate predictions in these challenging scenarios.

Chapter 6

Conclusion

In this report, we have explored the use of neural networks to solve PDEs, more specifically 1D stationary PDEs, through the Green's function approach. Our study demonstrated that neural networks are highly effective in learning both the Green's function and the homogeneous solution for a range of PDEs, including the Laplace, Helmholtz, and advection-diffusion equations. By training the network on a large and stochastic set of source terms and boundary conditions, we established that the neural network can reliably predict PDE solutions with remarkable accuracy, showcasing its potential as a powerful tool for solving complex differential equations.

As underlined in section 5.10, the neural network struggles to accurately predict the Green's function for the Helmholtz equation in the vicinity of the eigenmodes. This section provides the exact Green's function for the Helmholtz equation using the method of variation of parameters, offering a reference point for comparison with the neural network predictions. An interesting question to explore would be the development of deep-learning techniques that can accurately determine the extent to which a spatial region is close to an eigenmode, and then switch to a different model that can handle this situation. It could also involve finding the vicinities of singularities in the Green's function and then switching to a different model that can handle these situations.

This internship provided an in-depth exploration of neural networks for solving PDEs, highlighting both their potential and limitations. It offered valuable insights into deep learning, scientific machine learning, and working on complex, emerging subjects. The results demonstrated the capabilities of neural networks in solving differential equations and emphasized the need for further research to improve performance in challenging scenarios. Combining deep-learning techniques with traditional numerical methods can lead to more robust algorithms for PDEs, benefiting scientific computing and engineering. Additionally, this experience significantly enhanced my sense of autonomy, when it comes to doing a research project, and my ability to work on complex problems in a structured and systematic manner.

Bibliography

- [1] P. J. Olver. (2014). *Introduction to Partial Differential Equations*. Springer. Available at: <https://doi.org/10.1007/978-3-319-02099-0>. ISBN: 978-3-319-02098-3.
- [2] H. Reinhard. (2004). *Équations aux dérivées partielles, introduction*. Paris: Dunod Université, coll. « Sciences Sup ». (Reprint, first edition 1991, 291 p.). ISBN: 978-2100484225.
- [3] D. G. Duffy. (2001). *Green's Functions with Applications*. CRC Press. <https://doi.org/10.1201/9781420034790>. ISBN: 9780429142697.
- [4] R. L. Herman. (2024). *Introduction to Partial Differential Equations*. Self-published. Reformatted using a modification of the Tufte-book document class in L^AT_EX. Available at: https://people.uncw.edu/hermanr/pde1/PDE1notes/PDE1_Main.pdf.
- [5] S. Hassani. (2013). *Mathematical Physics: A Modern Introduction to Its Foundations* (2nd ed.). Springer, Cham. <https://doi.org/10.1007/978-3-319-01195-0>. ISBN: 978-3-319-01194-3.
- [6] F. Sur. (2023). *Introduction à l'apprentissage automatique*. Available at: https://members.loria.fr/FSur/enseignement/apprauto/poly_apprauto_FSur.pdf.
- [7] M. Tancik, P. P. Srinivasan, B. Mildenhall, S. Fridovich-Keil, N. Raghavan, U. Singhal, R. Ramamoorthi, J. T. Barron, and R. Ng. (2020). *Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains*. *arXiv preprint arXiv:2006.10739*. Available at: <https://doi.org/10.48550/arXiv.2006.10739>.
- [8] Z. Hao, S. Liu, Y. Zhang, C. Ying, Y. Feng, H. Su, J. Zhu. (2023). *Physics-Informed Machine Learning: A Survey on Problems, Methods and Applications*. arXiv:2211.08064v2 [cs.LG]. Available at: <https://arxiv.org/pdf/2211.08064v2>.
- [9] N. Boullé. (2022). *Data-Driven Discovery of Green's Functions*. (Doctoral thesis, University of Oxford). Available at: <https://nboulle.github.io/files/thesis.pdf>.
- [10] G. Pang, L. Yang, and G. E. Karniadakis, *Neural-net-induced Gaussian process regression for function approximation and PDE solution*, J. Comput. Phys., 384 (2019), pp. 270–288.
- [11] Z. Li, N. B. Kovachki, K. Azizzadenesheli, B. Liu, A. Stuart, A. Anandkumar. (2024). *Neural Operator: Learning Maps Between Function Spaces with Applications to PDEs*. Available at: <https://arxiv.org/pdf/2108.08481>.
- [12] N. Boullé, S. Kim, T. Shi, A. Townsend. (2023). *Learning Green's Functions Associated with Time-Dependent Partial Differential Equations*. Available at: <https://arxiv.org/pdf/2301.12345>.

- [13] *Scikit-learn: Machine Learning in Python*. Available at: https://scikit-learn.org/stable/modules/gaussian_process.html.
- [14] P. Frey. (n.d.). *The Finite Difference Method*. Available at: https://www.ljll.fr/frey/cours/UdC/ma691/ma691_ch6.pdf.
- [15] SciPy Developers. (n.d.). *scipy.integrate.solve_bvp*. SciPy Documentation. Available at: https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.solve_bvp.html.
- [16] W. Jarosz. (2008). *Efficient Monte Carlo Methods for Light Transport in Scattering Media*. Ph.D. Dissertation, Dartmouth College. Available at: Appendix A: <https://cs.dartmouth.edu/~wjarosz/publications/dissertation/appendixA.pdf>.
- [17] C. E. Rasmussen, C. K. I. Williams. (2006). *Gaussian Processes for Machine Learning*. The MIT Press. Available at: <https://gaussianprocess.org/gpml/chapters/RW.pdf>.

Appendix A

The theoretical framework for Boundary Value Problems

A.1 Introduction

In physics, boundary value problems (BVPs) [4] frequently manifest as second-order ordinary differential equations (ODEs). A typical instance might involve solving an equation of the form

$$a_2(x)u'' + a_1(x)u' + a_0(x)u = f(x),$$

subject to boundary conditions.

We can express such an equation using a differential operator $\mathcal{L}$:

$$\mathcal{L} = a_2(x)\frac{d^2}{dx^2} + a_1(x)\frac{d}{dx} + a_0(x)\text{Id}.$$

Remark A.1.1. *This framework is particularly useful in my internship because the test cases are second-order linear ODEs (since they are stationary one-dimensional PDEs) and are useful to easily determine the exact Green function for a given test case, which will then be used as a true target for the testing of a given neural operator. Since the Laplace equation and the Helmholtz equation can be expressed under the BVP framework, the Green's function have been determined in section C.1 and section C.3 respectively.*

A.2 Sturm-Liouville Operator

Definition 17. *We define the Sturm-Liouville operator as*

$$\mathcal{L}_{SL} = \frac{d}{dx} \left(p(x) \frac{d}{dx} \right) + q(x). \quad (\text{A.2.1})$$

The Sturm-Liouville operator takes part in the determination of the exact Green function for a given test case. This operator is particularly useful in the context of boundary value problems, as it allows for the transformation of a second-order ODE into a Sturm-Liouville form.

Example 7 (Laplace Equation). *The Laplace equation is a special case of the Helmholtz equation with $\lambda = 0$. It is given by:*

$$-\Delta u = 0,$$

where Δ is the Laplacian operator. In one dimension, this reduces to:

$$-\frac{d^2u}{dx^2} = 0.$$

Using the Sturm-Liouville operator, we can write this as:

$$\frac{d}{dx} \left(-1 \cdot \frac{du}{dx} \right) = 0,$$

where $p(x) = -1$ and $q(x) = 0$.

Example 8 (Helmholtz Equation). The Helmholtz equation is given by:

$$\Delta u + k^2 u = 0,$$

where k is the wave number. In one dimension, this reduces to:

$$\frac{d^2u}{dx^2} + k^2 u = 0.$$

Using the Sturm-Liouville operator, we can write this as:

$$\frac{d}{dx} \left(1 \cdot \frac{du}{dx} \right) + k^2 u = 0,$$

where $p(x) = 1$ and $q(x) = k^2$.

Appendix B

About Gaussian Processes

B.1 Introduction to Gaussian Processes

Gaussian Processes (GPs) [17] are a nonparametric framework used for supervised learning tasks, including regression and probabilistic classification. They define a prior over functions, which, when combined with observed data, results in a posterior distribution for making predictions.

GPs have several advantages, such as accurate interpolation of observed data points with appropriate kernel choices, providing a full probabilistic model that quantifies uncertainty, and offering versatility in kernel selection. However, they also have disadvantages, including high computational complexity due to the inversion of the covariance matrix and reduced efficiency in high-dimensional input spaces.

B.2 Mathematical Definitions related to Gaussian Processes

Definition 18. *A multivariate Gaussian distribution is a generalization of the one-dimensional (univariate) Gaussian distribution to higher dimensions. A random vector $\mathbf{x} \in \mathbb{R}^n$ is said to follow a multivariate Gaussian distribution if it has the probability density function:*

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} \exp \left(-\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu})^\top \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu}) \right) \quad (\text{B.2.1})$$

where $\boldsymbol{\mu} \in \mathbb{R}^n$ is the mean vector and $\Sigma \in \mathbb{R}^{n \times n}$ is the covariance matrix. The mean vector $\boldsymbol{\mu}$ represents the expected value of the random vector $\mathbf{x}$, and the covariance matrix Σ represents the covariance between each pair of elements in $\mathbf{x}$.

Formally, we write:

$$\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$$

The mean vector $\boldsymbol{\mu}$ is defined as:

$$\boldsymbol{\mu} = \mathbb{E}[\mathbf{x}]$$

The covariance matrix Σ is defined as:

$$\Sigma = \mathbb{E}[(\mathbf{x} - \boldsymbol{\mu})(\mathbf{x} - \boldsymbol{\mu})^\top]$$

Remark B.2.1. The multivariate Gaussian distribution has several important properties. Any linear transformation of a multivariate Gaussian random vector is also Gaussian. The marginal distribution of any subset of a multivariate Gaussian random vector is also Gaussian. The conditional distribution of a subset of a multivariate Gaussian random vector, given the values of the remaining elements, is also Gaussian.

Definition 19. A Gaussian Process (GP) is a collection of random variables, any finite number of which have a joint Gaussian distribution. Formally, a GP is defined as:

$$f(x) \sim \mathcal{GP}(m(x), k(x, x')) \quad (\text{B.2.2})$$

where $m(x)$ is the mean function and $k(x, x')$ is the covariance function (kernel). These are defined as:

$$m(x) = \mathbb{E}[f(x)], \quad k(x, x') = \mathbb{E}[(f(x) - m(x))(f(x') - m(x'))]$$

Given inputs $X = \{x_1, x_2, \dots, x_n\}$, the outputs $\mathbf{f} = [f(x_1), f(x_2), \dots, f(x_n)]$ are jointly Gaussian:

$$\mathbf{f} \sim \mathcal{N}(\mathbf{m}, K)$$

where $\mathbf{m} = [m(x_1), m(x_2), \dots, m(x_n)]$ and K is the covariance matrix with entries $K_{ij} = k(x_i, x_j)$.

For a new input x_* , the joint distribution of the observed outputs $\mathbf{y}$ and the function value $f_* = f(x_*)$ is:

$$\begin{bmatrix} \mathbf{y} \\ f_* \end{bmatrix} \sim \mathcal{N} \left(\begin{bmatrix} \mathbf{m} \\ m_* \end{bmatrix}, \begin{bmatrix} K(X, X) + \sigma_n^2 I & K(X, x_*) \\ K(x_*, X) & k(x_*, x_*) \end{bmatrix} \right)$$

where $K(X, X)$ is the covariance matrix of the training inputs, $K(X, x_*)$ is the covariance vector between the training inputs and the new input, and $k(x_*, x_*)$ is the covariance at the new input.

B.3 RBF Kernel

Definition 20. The Radial Basis Function (RBF) kernel, also referred to as the squared exponential kernel, is a fundamental stationary and isotropic kernel. It is mathematically defined as:

$$K_{RBF}(x, x') = \exp \left(-\frac{\|x - x'\|^2}{2\ell^2} \right), \quad (\text{B.3.1})$$

where ℓ denotes the length-scale parameter. The parameter ℓ can be a scalar, resulting in an isotropic kernel, or a vector, leading to an anisotropic kernel that allows for different length-scales across dimensions.

Remark B.3.1. The RBF kernel is infinitely differentiable, implying that sample functions drawn from a Gaussian Process (GP) with this kernel are smooth to any desired degree.

In our implementation, we relied on the RBF Kernel implementation from the SciKit-Learn library [13] to generate samples for the source terms of the PDEs for which we desire to find the Green's function.

Appendix C

Exact Green's Functions

C.1 Finding the Expression for the Green's Function Using the Method of Variation of Parameters

The Method of Variation of Parameters is a powerful technique for finding the Green's function for differential equations. Here, we outline the steps to derive the Green's function for both the Laplace and Helmholtz equations.

C.2 Green's Function for the Laplace Equation

To solve the problem step-by-step, let's follow the instructions provided:

C.2.1 Step 1: Finding u_1 and u_2

Given the Laplace equation $-\Delta u = f$ with boundary conditions $u(0) = 0$ and $u(1) = 0$, we need to find the solutions to the homogeneous equation $-\Delta u = 0$.

Let us consider the functions $u_1(x)$ and $u_2(x)$ as solutions to the homogeneous problem, where $u_1(0) = 0$, $u_2(1) = 0$, and $u_1(1) \neq 0$, $u_2(0) \neq 0$ (as evoked under the equation 2.3.1).

Specifically:

1. The function $u_1(x) = x$ satisfies $u_1(0) = 0$ and $u_1(1) = 1$.
2. The function $u_2(x) = 1 - x$ satisfies $u_2(1) = 0$ and $u_2(0) = 1$.

These functions $u_1(x)$ and $u_2(x)$ are solutions of the homogeneous equation that meet the following conditions:

$$\begin{aligned}u_1(0) &= 0 & \text{and} & & u_2(1) &= 0, \\u_1(1) &\neq 0 & \text{and} & & u_2(0) &\neq 0.\end{aligned}$$

C.2.2 Step 2: Sturm-Liouville Equation and $p(x)$

For the Laplace equation, the Sturm-Liouville form is:

$$\frac{d}{dx} \left(p(x) \frac{du}{dx} \right) + q(x)u = 0$$

Here, $p(x) = -1$ and $q(x) = 0$.

C.2.3 Step 3: Finding the Wronskian

The Wronskian $W(u_1, u_2)$ is given by:

$$W(u_1, u_2) = u_1(x)u_2'(x) - u_1'(x)u_2(x)$$

Substituting $u_1(x) = x$ and $u_2(x) = 1 - x$:

$$u_1'(x) = 1$$

$$u_2'(x) = -1$$

Thus,

$$W(u_1, u_2) = x(-1) - 1(1 - x) = -x - 1 + x = -1$$

C.2.4 Step 4: Applying the Green's Function Formula

Using the formula:

$$G(x, \xi) = \begin{cases} \frac{u_1(\xi)u_2(x)}{p(\xi)W(\xi)}, & a \leq \xi \leq x, \\ \frac{u_1(x)u_2(\xi)}{p(\xi)W(\xi)}, & x \leq \xi \leq b, \end{cases}$$

Substituting $u_1(x) = x$, $u_2(x) = 1 - x$, $p(\xi) = -1$, and $W(\xi) = -1$:

$$G(x, \xi) = \begin{cases} x(1 - \xi), & 0 \leq \xi \leq x, \\ \xi(1 - x), & x \leq \xi \leq 1, \end{cases} \quad (\text{C.2.1})$$

C.2.5 Step 5: Verifying the Equation

We need to verify that $G(x, \xi)$ satisfies:

$$-\Delta G(x, \xi) = \delta(x - \xi)$$

For $0 \leq \xi \leq x$:

$$G(x, \xi) = x(1 - \xi)$$

$$\frac{\partial^2 G}{\partial x^2} = 0$$

For $x \leq \xi \leq 1$:

$$G(x, \xi) = \xi(1 - x)$$

$$\frac{\partial^2 G}{\partial x^2} = 0$$

At $x = \xi$, there is a discontinuity in the first derivative, which corresponds to the delta function $\delta(x - \xi)$.

C.2.6 Step 6: Checking the Boundary Conditions

The Green's function $G(x, \xi)$ should satisfy the boundary conditions:

$$G(0, \xi) = 0$$

$$G(1, \xi) = 0$$

For $0 \leq \xi \leq x$:

$$G(0, \xi) = 0(1 - \xi) = 0$$

$$G(1, \xi) = 1(1 - \xi) = 1 - \xi$$

For $x \leq \xi \leq 1$:

$$G(0, \xi) = \xi(1 - 0) = \xi$$

$$G(1, \xi) = \xi(1 - 1) = 0$$

Thus, the Green's function $G(x, \xi)$ satisfies all the required properties.

C.3 Green's Function for the Helmholtz Equation

To solve the problem step-by-step, let's follow the instructions provided:

C.3.1 Step 1: Finding u_1 and u_2

Let us consider the functions $u_1(x)$ and $u_2(x)$ as solutions to the homogeneous Helmholtz problem, where $u_1(0) = 0$, $u_2(1) = 0$, and $u_1(1) \neq 0$, $u_2(0) \neq 0$. Specifically:

1. The function $u_1(x) = \sin(kx)$ satisfies $u_1(0) = 0$ and $u_1(1) = \sin(k)$.
2. The function $u_2(x) = \sin(k(1 - x))$ satisfies $u_2(1) = 0$ and $u_2(0) = \sin(k)$.

These functions $u_1(x)$ and $u_2(x)$ are solutions of the homogeneous Helmholtz equation that meet the following conditions:

$$u_1(0) = 0 \quad \text{and} \quad u_2(1) = 0,$$

$$u_1(1) \neq 0 \quad \text{and} \quad u_2(0) \neq 0.$$

C.3.2 Step 2: Sturm-Liouville Equation and $p(x)$

For the Helmholtz equation, the Sturm-Liouville form is:

$$\frac{d}{dx} \left(p(x) \frac{du}{dx} \right) + q(x)u = 0$$

Here, $p(x) = 1$ and $q(x) = k^2$.

C.3.3 Step 3: Finding the Wronskian

The Wronskian $W(u_1, u_2)$ is given by:

$$W(u_1, u_2) = u_1(x)u_2'(x) - u_1'(x)u_2(x)$$

Substituting $u_1(x) = \sin(kx)$ and $u_2(x) = \sin(k(1-x))$:

$$u_1'(x) = k \cos(kx)$$

$$u_2'(x) = -k \cos(k(1-x))$$

Thus,

$$W(u_1, u_2) = \sin(kx)(-k \cos(k(1-x))) - k \cos(kx) \sin(k(1-x)) = -k[\sin(kx) \cos(k(1-x)) + \cos(kx) \sin(k(1-x))]$$

Using the trigonometric identity $\sin(A+B) = \sin(A)\cos(B) + \cos(A)\sin(B)$, we get:

$$W(u_1, u_2) = -k \sin(k(x+1-x)) = -k \sin(k) = -k \sin(k)$$

C.3.4 Step 4: Applying the Green's Function Formula

Using the formula:

$$G(x, \xi) = \begin{cases} \frac{u_1(\xi)u_2(x)}{p(\xi)W(\xi)}, & 0 \leq \xi \leq x, \\ \frac{u_1(x)u_2(\xi)}{p(\xi)W(\xi)}, & x \leq \xi \leq 1, \end{cases}$$

Substituting $u_1(x) = \sin(kx)$, $u_2(x) = \sin(k(1-x))$, $p(\xi) = 1$, and $W(\xi) = -k \sin(k)$:

$$G(x, \xi) = \begin{cases} -\frac{\sin(k\xi) \sin(k(1-x))}{k \sin(k)}, & 0 \leq \xi \leq x, \\ -\frac{\sin(kx) \sin(k(1-\xi))}{k \sin(k)}, & x \leq \xi \leq 1, \end{cases}$$

Simplifying:

$$G(x, \xi) = \begin{cases} -\frac{\sin(k\xi) \sin(k(1-x))}{k \sin(k)}, & 0 \leq \xi \leq x, \\ -\frac{\sin(kx) \sin(k(1-\xi))}{k \sin(k)}, & x \leq \xi \leq 1, \end{cases} \quad (\text{C.3.1})$$

C.3.5 Step 5: Verifying the Equation

We need to verify that $G(x, \xi)$ satisfies:

$$\left(\frac{d^2}{dx^2} + k^2 \right) G(x, \xi) = \delta(x - \xi)$$

For $0 \leq \xi \leq x$:

$$G(x, \xi) = \frac{-\sin(k\xi) \sin(k(1-x))}{k \sin(k)}$$

$$\frac{d^2 G}{dx^2} + k^2 G = \frac{-k^2 \sin(k\xi) \sin(k(1-x))}{k \sin(k)} + k^2 G = 0$$

For $x \leq \xi \leq 1$:

$$G(x, \xi) = \frac{-\sin(kx) \sin(k(1-\xi))}{k \sin(k)}$$

$$\frac{d^2 G}{dx^2} + k^2 G = \frac{-k^2 \sin(kx) \sin(k(1-\xi))}{k \sin(k)} + k^2 G = 0$$

At $x = \xi$, there is a discontinuity in the first derivative, which corresponds to the delta function $\delta(x - \xi)$.

C.3.6 Step 6: Checking the Boundary Conditions

The Green's function $G(x, \xi)$ should satisfy the boundary conditions:

$$G(0, \xi) = 0$$

$$G(1, \xi) = 0$$

For $0 \leq \xi \leq x$:

$$G(0, \xi) = \frac{-\sin(k\xi)\sin(k)}{k\sin(k)} = -\frac{\sin(k\xi)}{k} = 0 \text{ since } \sin(0) = 0$$

$$G(1, \xi) = \frac{-\sin(k\xi)\sin(0)}{k\sin(k)} = 0$$

For $x \leq \xi \leq 1$:

$$G(0, \xi) = \frac{-\sin(k(0))\sin(k(1-\xi))}{k\sin(k)} = 0 \text{ since } \sin(0) = 0$$

$$G(1, \xi) = \frac{-\sin(k)\sin(k(1-\xi))}{k\sin(k)} = 0$$

Thus, the Green's function $G(x, \xi)$ satisfies all the required properties.

Remark C.3.1. *The denominator $k\sin(k)$ in the Green's function for the Helmholtz equation is non-zero for $k \neq 0$. However, it becomes a singularity when $k = n\pi$ for $n \in \mathbb{Z}$. In these cases, the Green's function should be handled with care to avoid numerical instabilities. Such aspect is underlined in section 5.10.*

Appendix D

Table of the several source terms used for the numerical results

Here below is a table of the source terms used in the numerical results, especially with regards to the Laplace equation (in section 5.5.4), the Advection-Diffusion equation (in section 5.6.5), and the Helmholtz equation (in sections 5.7.4 and 5.8.4), especially for the study of the relative error (%) between predicted solutions by the Green Neural Operator and the corresponding true target solutions. The source terms are denoted as $f_1(x; \mu)$ to $f_{10}(x; \mu)$, where μ is a physical parameter that can be adjusted to control the behavior of the source term.

Source Term	Expression
$f_1(x; \mu)$	$x + \mu$
$f_2(x; \mu)$	$\mu x^2 + 2x + 1$
$f_3(x; \mu)$	μx^3
$f_4(x; \mu)$	$e^{-\mu x}$
$f_5(x; \mu)$	$e^{(x-\mu)^2}$
$f_6(x; \mu)$	$\sin(2\pi x)$
$f_7(x; \mu)$	$\frac{\mu x^2 + 2x + 1}{x^2 - 3}$
$f_8(x; \mu)$	$\sin(2\pi x) \cdot e^{\frac{(x-1)^2}{0.05\mu}}$
$f_9(x; \mu)$	$x \cdot \tan(x) \cdot \sin(2\pi \mu x) \cdot e^{-\mu x}$
$f_{10}(x; \mu)$	$\frac{e^{-\mu x} - 1}{x^2 - 4} \cdot \cos(2\pi x)$

Table D.1: Source terms $f_1(x; \mu)$ to $f_{10}(x; \mu)$ used in the analysis.

Appendix E

Other numerical results

E.1 Some miscallenous results for the Laplace equation

E.1.1 Getting the predicted solution on a specific source term

The figure below shows the predicted solution obtained from the neural network for a specific source term. The solution accurately captures the dynamics of the Laplace equation, aligning with the theoretical expectations and the reference solution.

The reference solution used for this test case was $u(x; \mu) = \sin(2\pi x) \cdot \exp\left(-\frac{(x-1)^2}{0.05\mu}\right)$, where $\mu = 0.5$. The solution was computed on the interval $[0, 1]$ with boundary conditions $u(0) = 0$ and $u(1) = 0$.

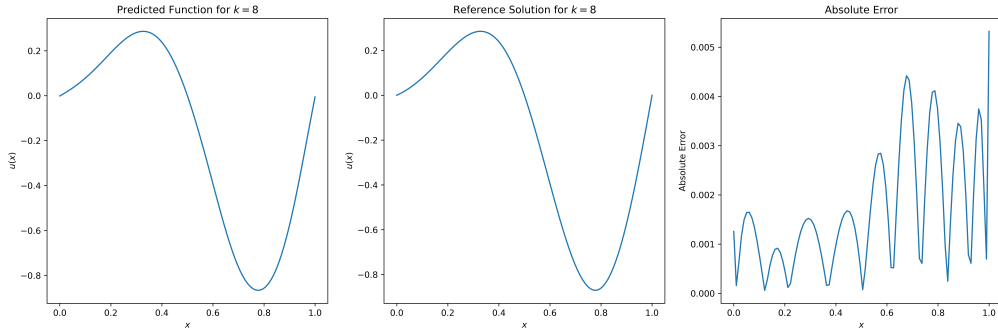


Figure E.1: Predicted Solution for the Laplace Equation

Analysis: The predicted solution accurately captures the behavior of the Laplace equation for the given source term, demonstrating the neural network's ability to solve the PDE with high precision. The solution aligns with the theoretical expectations and the reference solution, showcasing the network's effectiveness in learning the Green's function and homogeneous solution.

E.2 Some miscallenous results for the Advection-Diffusion equation

E.2.1 About the test case used

The reference solution used for this test case was $u(x; D) = -x^2 - 2x + 1$, where $\mu = 0.5$. The solution was computed on the interval $[0, 1]$ with boundary conditions $u(0) = 1$ and $u(1) = -2$.

The corresponding source term was term was therefore $f(x; D) = -x^2 - 6x - 3.0 - 2D$.

E.2.2 The homogeneous solution

The figure below shows the predicted homogeneous solution obtained from the neural network for the advection-diffusion equation. The homogeneous solution aligns with the theoretical expectations and the reference solution, demonstrating the network's ability to accurately capture the system's dynamics.

For $D = 0.5$:

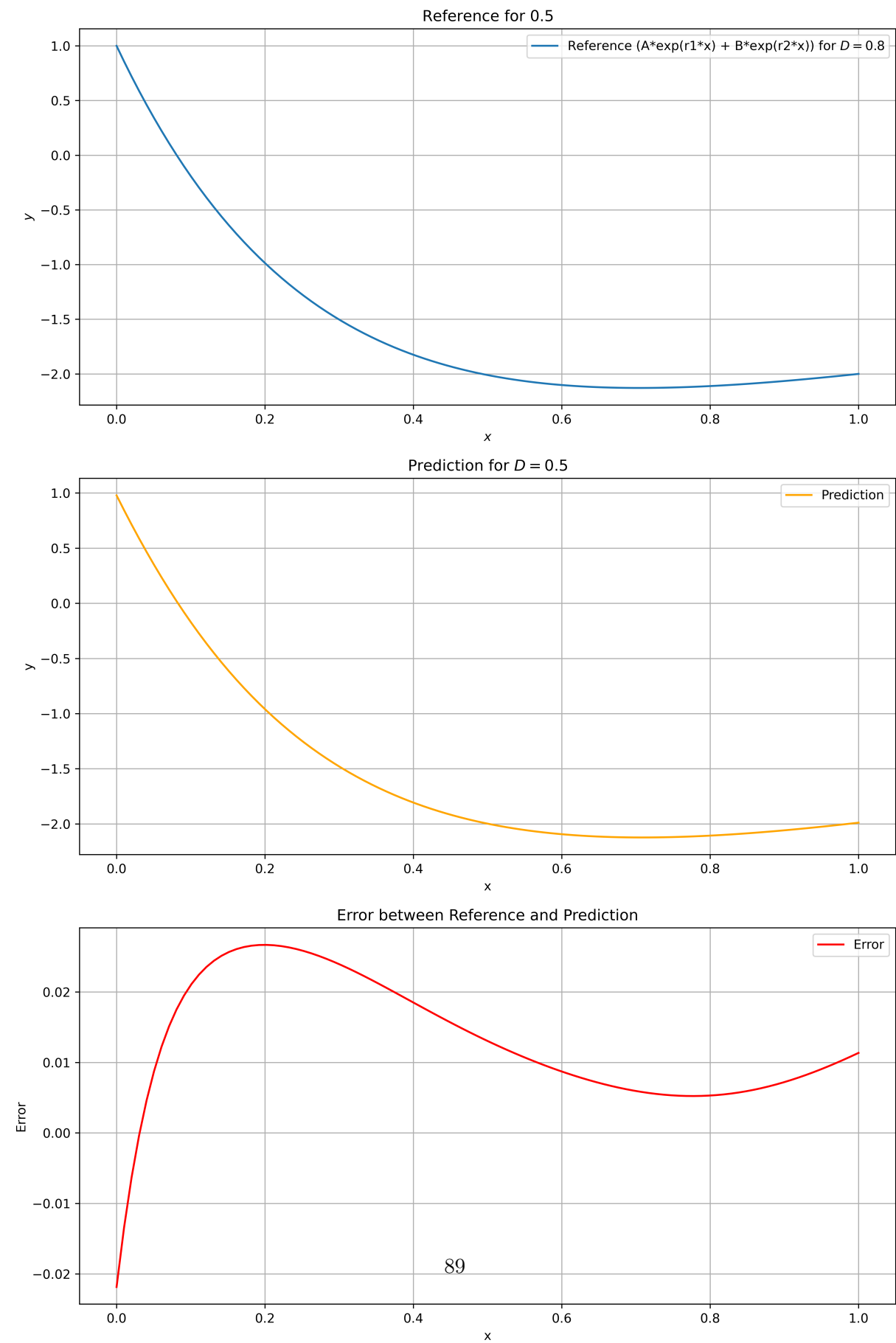


Figure E.2: Predicted Homogeneous Solution for the Advection-Diffusion Equation with $D = 0.5$

For $D = 1.0$:

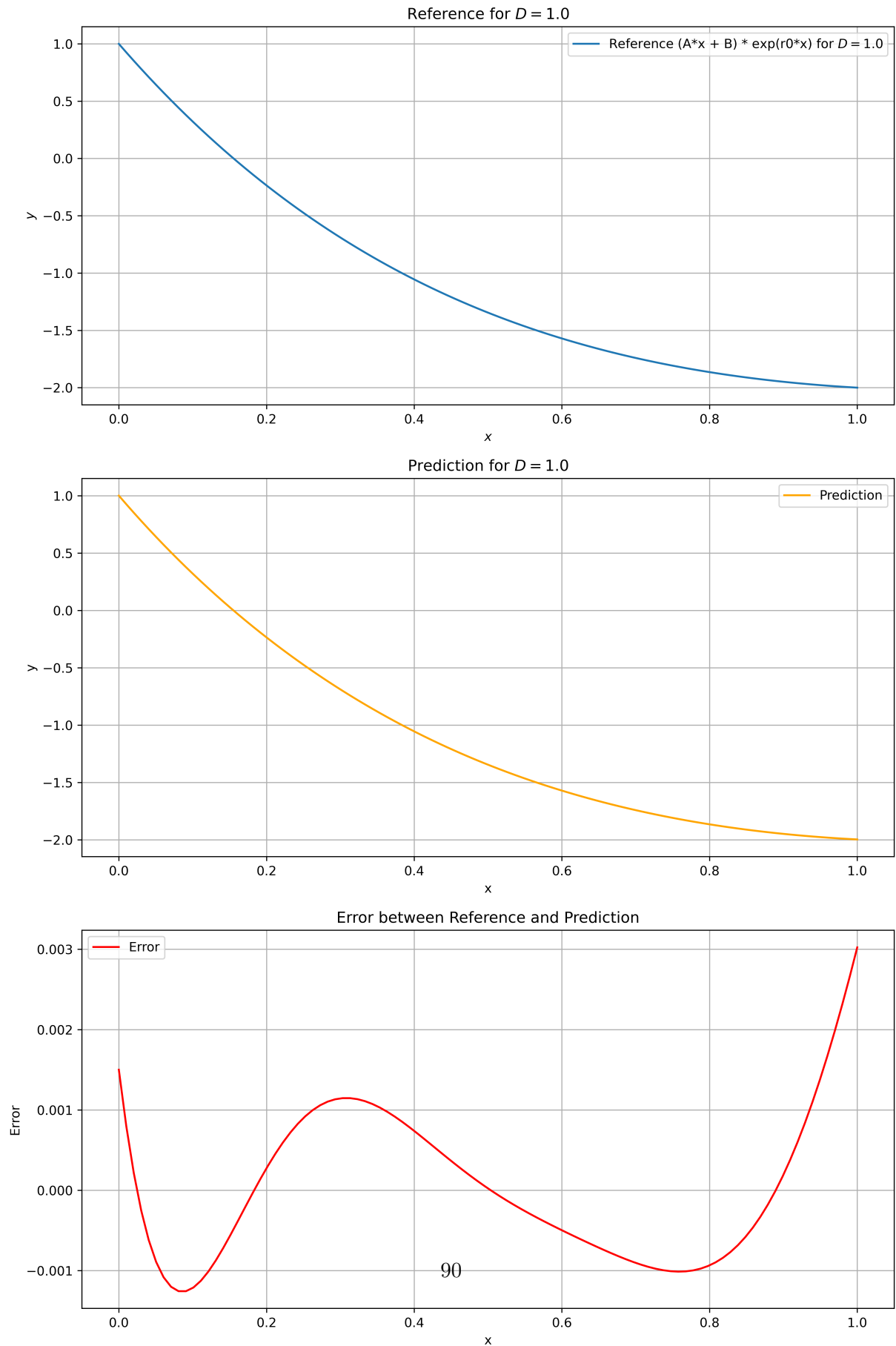


Figure E.3: Predicted Homogeneous Solution for the Advection-Diffusion Equation with $D = 1.0$

For $D = 1.5$:

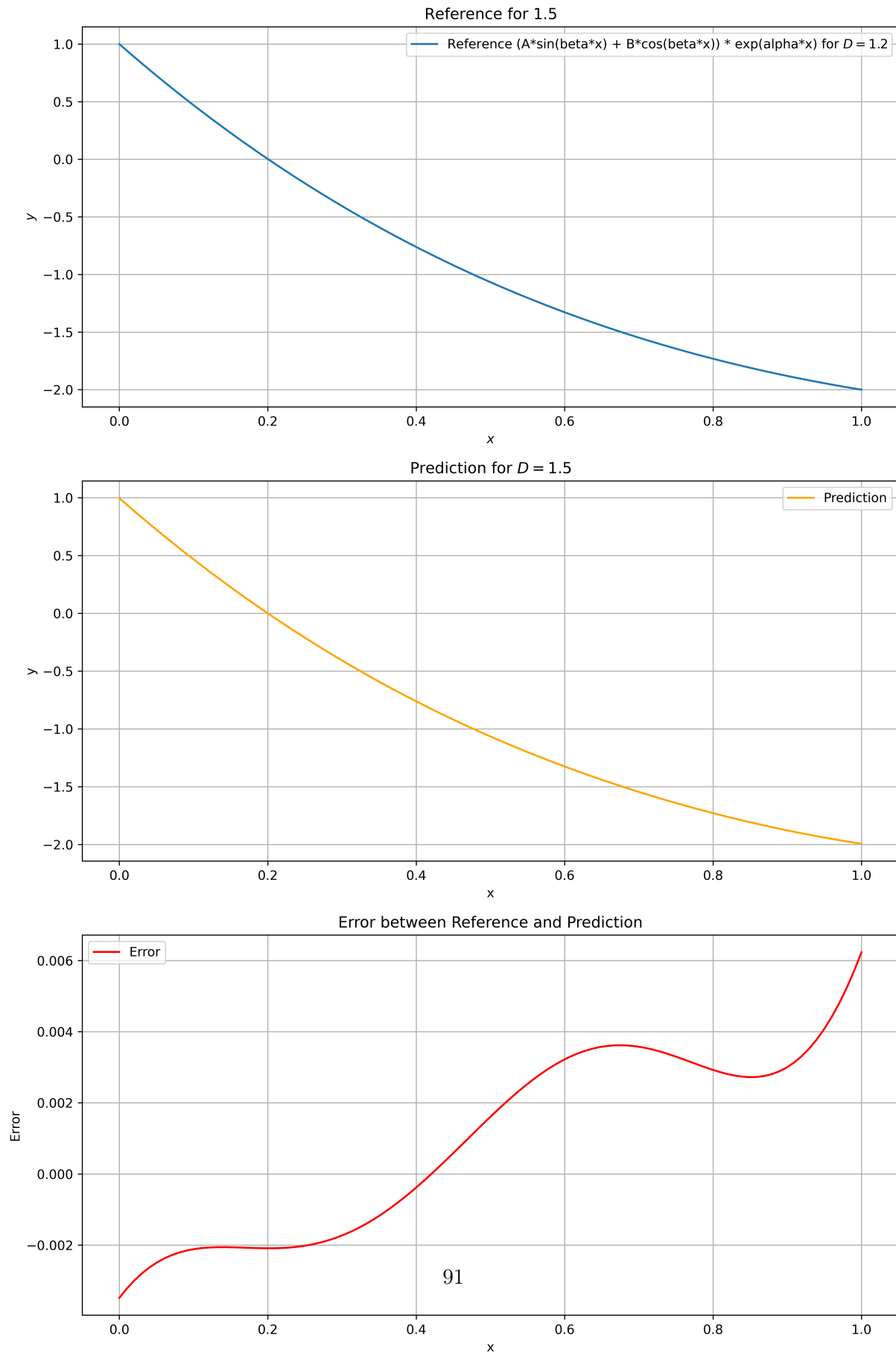


Figure E.4: Predicted Homogeneous Solution for the Advection-Diffusion Equation with $D = 1.5$

Analysis: The predicted homogeneous solution aligns with the theoretical expectations and the reference solution for the advection-diffusion equation. The solution accurately captures the system's dynamics and behavior, showcasing the network's ability to learn the Green's function and homogeneous solution. The results demonstrate the network's effectiveness in solving the PDE with high precision, providing valuable insights into the advection-diffusion equation's characteristics.

E.2.3 Getting the predicted solution on a specific source term

The figure below shows the predicted solution obtained from the neural network for a specific source term. The solution accurately captures the dynamics of the advection-diffusion equation, aligning with the theoretical expectations and the reference solution.

For $D = 0.5$:

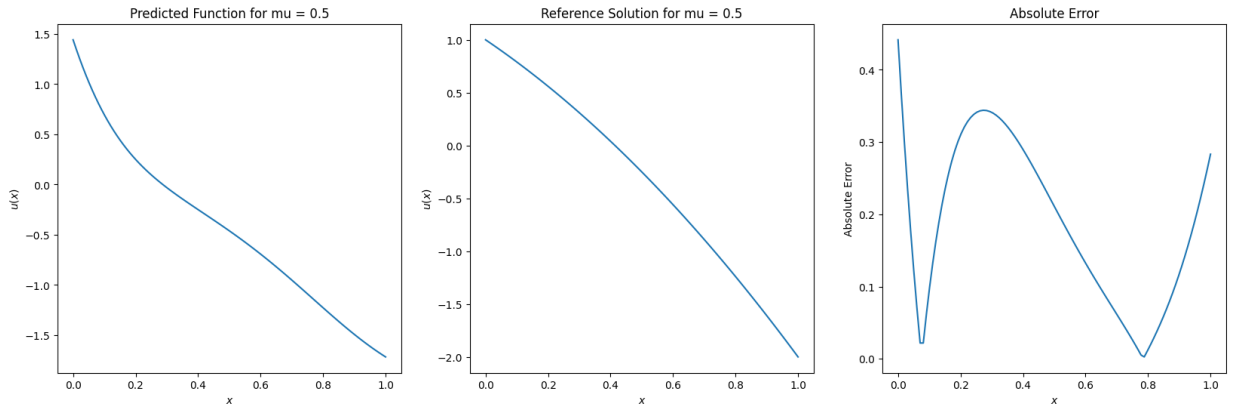


Figure E.5: Predicted Solution for the Advection-Diffusion Equation

For $D = 1.0$:

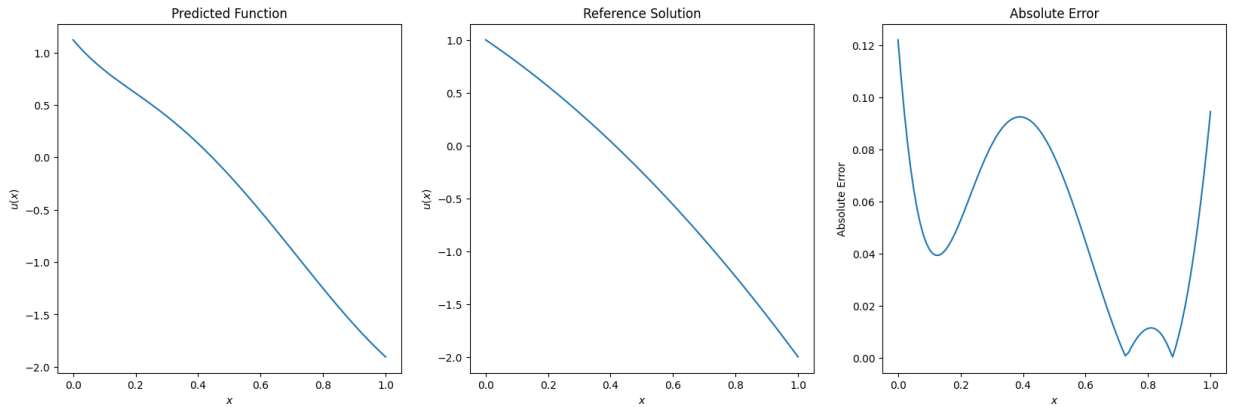


Figure E.6: Predicted Solution for the Advection-Diffusion Equation

For $D = 1.5$:

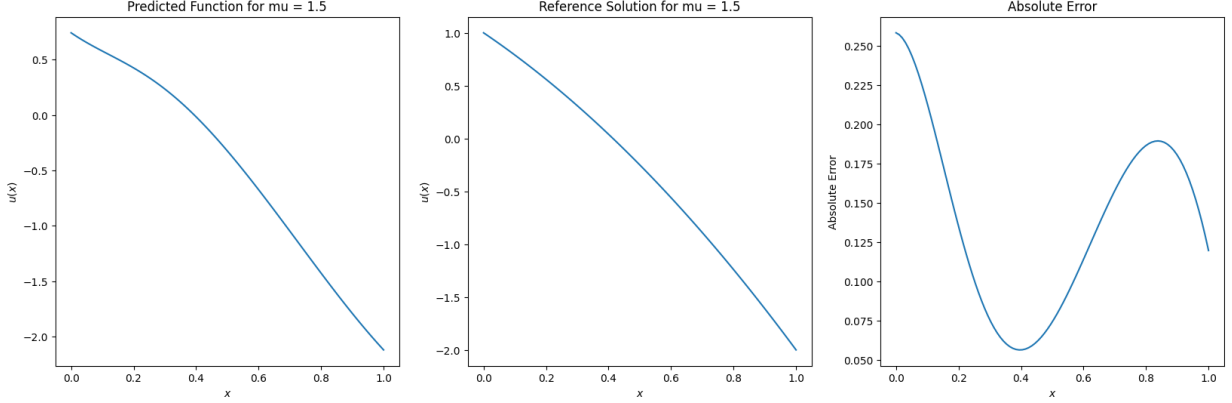


Figure E.7: Predicted Solution for the Advection-Diffusion Equation

Analysis: The predicted solution accurately captures the behavior of the advection-diffusion equation for the given source term, demonstrating the neural network’s ability to solve the PDE with high precision. The solution aligns with the theoretical expectations and the reference solution, showcasing the network’s effectiveness in learning the Green’s function and homogeneous solution.

E.3 Some miscallaneous results for the Helmholtz equation with $k = 8$

E.3.1 Getting the predicted solution on a specific source term

The figure below shows the predicted solution obtained from the neural network for a specific source term. The solution accurately captures the dynamics of the Helmholtz equation with a fixed wave number $k = 8$, aligning with the theoretical expectations and the reference solution.

The reference solution used for this test case was $u(x; \mu) = \sin(2\pi x) \cdot \exp\left(-\frac{(x-1)^2}{0.05\mu}\right)$, where $\mu = 0.5$. The solution was computed on the interval $[0, 1]$ with boundary conditions $u(0) = 0$ and $u(1) = 0$.

Instances of plots for the solution for the Helmholtz equation can be found in the appendix in the section E.4.1.

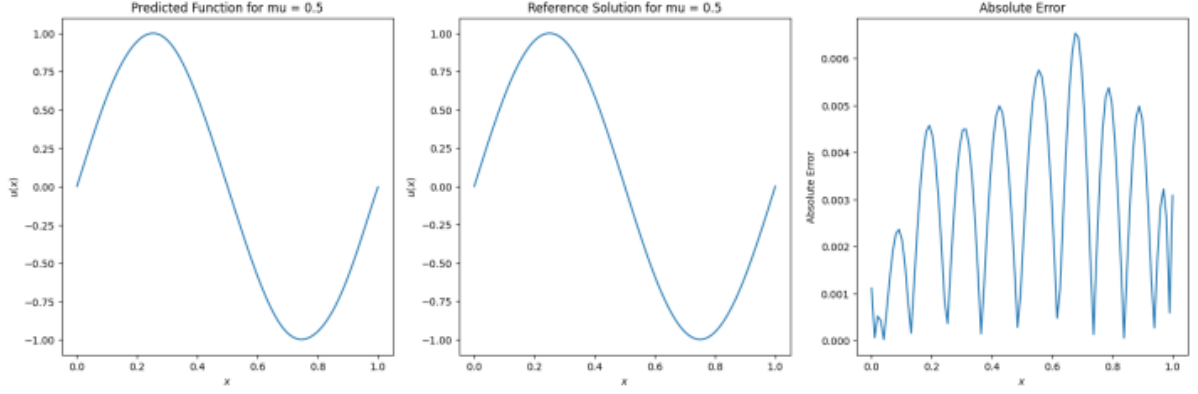


Figure E.8: Predicted Solution for the Helmholtz Equation with $k = 8$

Analysis: The predicted solution accurately captures the behavior of the Helmholtz equation with a fixed wave number $k = 8$ for the given source term, demonstrating the neural network's ability to solve the PDE with high precision. The solution aligns with the theoretical expectations and the reference solution, showcasing the network's effectiveness in learning the Green's function and homogeneous solution.

E.4 Some miscallenous results for the parameterized Helmholtz equation

E.4.1 Getting the predicted solution on a specific source term

The figure below shows the predicted solution obtained from the neural network for a specific source term. The solution accurately captures the dynamics of the Helmholtz equation with varying wave numbers, aligning with the theoretical expectations and the reference solution.

The reference solution used for this test case was $u(x; \mu) = \sin(2\pi x) \cdot \exp\left(-\frac{(x-1)^2}{0.05\mu}\right)$, where $\mu = 0.5$. The solution was computed on the interval $[0, 1]$ with boundary conditions $u(0) = 0$ and $u(1) = 0$.

For $k = 1$:

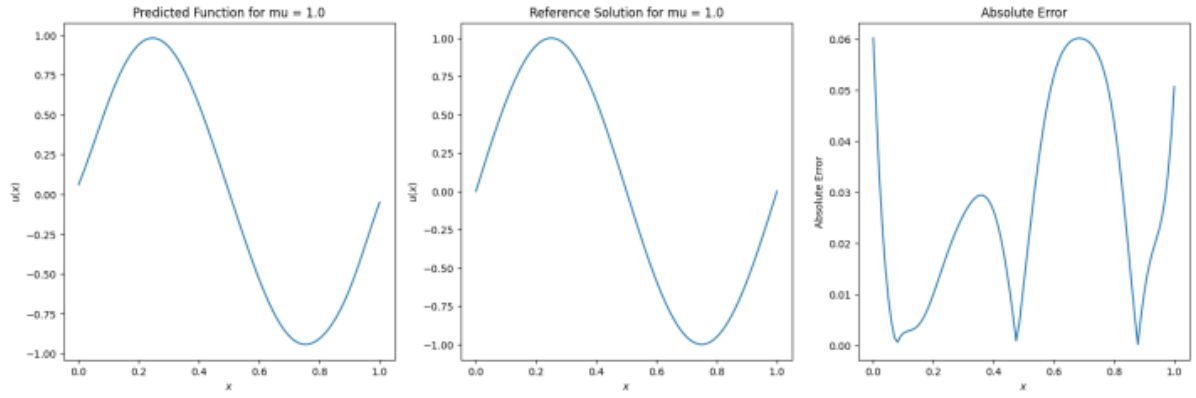


Figure E.9: Predicted Solution for the Helmholtz Equation with $k = 1$

For $k = 1.5$:

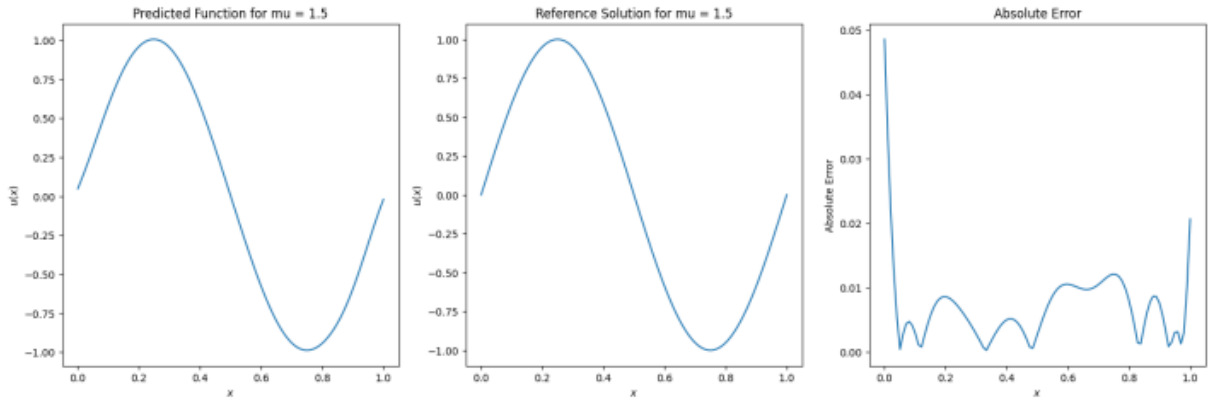


Figure E.10: Predicted Solution for the Helmholtz Equation with $k = 1.5$

For $k = 2$:

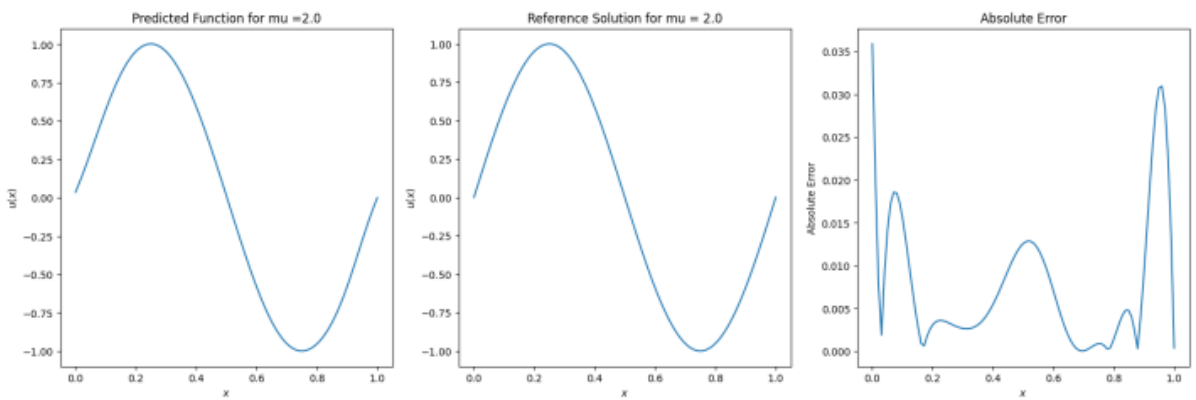


Figure E.11: Predicted Solution for the Helmholtz Equation with $k = 2$

Analysis: The predicted solution accurately captures the behavior of the Helmholtz equation with varying wave numbers for the given source term, demonstrating the neural network's ability to solve the PDE with high precision. The solution aligns with the theoretical expectations and the reference solution, showcasing the network's effectiveness in learning the Green's function and homogeneous solution.