Eötvös Loránd University

Faculty of Mathematics

Dept. of Applied Analysis and Computation

Machine learning approach for the numerical solution of initial and boundary value problems

Supervisor:
Izsák Ferenc
Associate professor

Author:
Hoang Trung Hieu
Applied Mathematics MSc student

Budapest, 2022

DECLARATION OF AUTHORSHIP

Name: Hoang Trung Hieu

Program: MSc in Applied Mathematics, Faculty of Science, ELTE

NEPTUN ID: WK87EA

Title of the thesis: Machine learning approach for the numerical solution of initial and boundary value problems.

I, as the sole author of this work hereby certify that the thesis I am submitting is entirely my own original work except where otherwise indicated. All references and quotations are done in a form that is standard for the subject and any use of the works of any other author, in any form, is properly acknowledged at their point of use.

Budapest, 2022.05.30

Signature of the student

ACKNOWLEDGEMENTS

I would like to give my great thanks to my supervisor Professor Izsák Ferenc for his great supervision, continuous support, helpful conversations, advice throughout the research of my MSc study, sharing knowledge and expertise. In fact, words cannot express my heartfelt gratitude, appreciation and thanks for all the support, guidance and time he had provided during my stay in ELTE.

My gratitude extends to Imre Fekete for his valuable advice and motivation that makes me fond of numerical methods.

A part of my thesis was supported by the National Research, Development and Innovation Office within the framework of the Thematic Excellence Program 2021 - National Research Sub programme: ”Artificial intelligence, large networks, data security: mathematical foundation and applications”.

Contents

1 Method of fundamental solutions for Laplace equations 7

1.1 The first approach 7

1.1.1 The Laplace's equation with Dirichlet boundary conditions . . 7

1.1.2 The Laplace's equation with Neumann boundary conditions . 10

1.1.3 The Laplace's equation with mixed boundary conditions . . . 11

1.2 The second approach 11

1.2.1 The Laplace's equation with Dirichlet boundary conditions . . 11

1.2.2 The Laplace's equation with Neumann boundary conditions . 12

1.2.3 The Laplace's equation with mixed boundary conditions . . . 13

1.3 Numerical experiments 13

1.3.1 Dirichlet boundary condition on the unit square 14

1.3.2 Dirichlet boundary condition on the unit circle 18

1.3.3 Dirichlet boundary conditions on the epitrochoid 19

1.3.4 Case study for Neumann type boundary conditions 19

1.3.5 Mixed boundary value problem on an Amoeba-like domain . . 21

1.3.6 Mixed boundary condition problem on a flower-shaped domain 22

1.3.7 Different neural networks and comparisons. 22

2 Method of fundamental solutions for Helmholtz equations 25

2.1 Solving Helmholtz-type equations 25

2.2 Numerical results 27

2.2.1 Solving the modified Helmholtz equation on sphere 27

2.2.2 Solving modified Helmholtz equation on the cube 28

2.2.3 Solving modified Helmholtz equation on the brick 30

3 Method of particular solutions 32

3.1 The method for Poisson equation 32

3.2 The method for modified Helmholtz equations 34

3.3 An alternative method for Helmholtz equation 36
3.4 The method of particular solutions for the wave equation 37
3.5 Numerical experiments 39

4 Error analysis **41**
4.1 Recent results 41
4.2 Our results 42
4.3 Numerical experiments 45

5 Adaptive finite difference methods using artificial neural networks **48**
5.1 Motivations 48
5.2 Description of a neural network-based method for solving ODEs . . . 49
5.3 Numerical Experiments for ODEs 51
5.3.1 A motivating example 51
5.3.2 System of ODEs 52
5.3.3 Case study for a non-linear ODE 55
5.4 Case study for the Burgers equation 56
5.4.1 Numerical schemes as a neural net 56
5.4.2 Adaptive method 57

A Special functions **60**
A.1 Bessel functions 60
A.2 Modified Bessel functions 61
A.3 Hankel functions 61

Bibliography **62**

List of Figures **67**

List of Tables **69**

Introduction

Overview

The dissertation includes three main parts: machine learning points of view method of fundamental solutions, method of particular solutions and machine learning framework for solving stiff ordinary differential equations- partial differential equations(ODEs-PDEs).

The purpose of the first part (the first and second chapters) is to introduce two approaches to solving homogeneous Laplace's equation and Helmholtz's equation. We use mesh-free methods which are based on the relation among fundamental solutions and can be implemented by neural networks. We construct an approximation of an analytical solution as a linear combination of fundamental solutions at outer points. We also try to find a mapping between fundamental solutions at boundary points and interior points. The first one is memory-efficient and the learning data in the second one is independent of the boundary conditions. Both of these methods work efficiently giving small errors in a variety of domain shapes and in case of different boundary conditions. Moreover, the different neural networks can lead to the same accuracy.

The second part(the third chapter) is the extension of the first part, we develop the method of particular solutions for dealing with inhomogeneous problems. The numerical experiments demonstrate the efficiency of this method. In collaboration with Izsák and Gábor [1], the convergence rate for three-dimensional domains in both $L2$-norm and $H1$-norm was also added to this thesis in chapter 4.

Chapter 5, on the other hand, is independent of the first two parts. We use transfer learning to accelerate the accuracy without decreasing the step size. The method's idea is adapted from the multigrid methods to optimize the loss function. In this part, we propose two improvements: the one-by-one loss optimizer and more

flexible neural network architectures. It can be applied for stiff problems, both ODEs and PDEs. The numerical results show that it outweighs the standard numerical methods.

Literature review

The method of fundamental solutions (MFS), initiated in [2], has a long history. The main idea of this approach is to approximate the solution u of some boundary value problem $Lu = 0$ as a linear combination of some fundamental solutions of the free-space differential operator L. Owing to this simple principle and the mesh-free nature of the corresponding approximations, it became a popular computational tool in the engineering practice [3]. Corresponding algorithms were developed for a number of linear differential operators including inhomogeneous problems and time-dependent cases [4] and later on, for non-linear operators [5], as well.

Recently, the development of the practice and software tools for neural networks made it as one of the most powerful tools in applied mathematics. In this way, it is a natural attempt, to apply this tool somehow for the solution of the major problems in numerical analysis. An important class of these problems is the numerical solution of PDE's. Accordingly, some related works were published in the last years. The main research direction was to mimic the geometry of the domain and the corresponding finite element or finite difference discretization, called the *physics informed neural networks*. At the level of the linear solvers, their motivation was the multigrid method and related convolutional neural networks were constructed. This is extended to a number of PDE's and a whole library of neural networks was prepared for computing [6].

The cornerstone of the corresponding convergence theory is the approximation property of the family of fundamental solutions for L. For this, a general result - using Holder norms - was developed in [7], with a summary of the preceding results. A convergence rate depending on the discretization parameters is, however, missing in these works. Regarding this, a significant result was developed earlier in [8] for (two-dimensional) Jordan domains with an analytic boundary. Later on, a full analysis was performed in [9], where the case of the Dirichlet Laplacian operator and the unit disk was investigated. In both cases, an exponential convergence rate was proved in a special Fourier type norm and in the $\|.\|_\infty$-norm, respectively. The analysis was

further extended to the spheres [10] and for the Helmholtz operator in [11] and error estimates were developed in the $\|.\|_2$-norm. Regarding the practical numerical simulations, we have to face with huge condition numbers in the corresponding linear systems [12]. To avoid direct solution algorithms, the least-squares approach was proposed, for an advanced version, see [13] . This approach is of great importance in case of computationally expensive problems [14]. The aim of the present contribution is to extend the above approximation results in two ways. First, we prove them on general 3-dimensional domains with a given convergence rate. Second, the corresponding error estimates will be given in the natural energy norm $\|.\|_{H_1}$. After collecting the mathematical tools, we prove the main result, where the theory of the boundary integral methods will be utilized. We also demonstrate the efficiency of the method in a real 3D-case. Finally, we discuss the results and mention new perspectives.

In [15] and [16], the physics informed neural networks(PINNs) were introduced to solve ODEs/PDEs relied on the physics-based laws of nature such as conservation of mass, energy and momentum. There are three scenarios according to the ratio between physical invariances and data that can be utilized in the training process. Adding the physical constraints can assist in generalisation improvement in case we have fewer data and it saves the computational time by reducing the free parameter of neural networks in case we have heavy data. The first scenario is to simulate a massive amount of data and try to minimize the least square error between the data and the neural network approximation. There is no guarantee that these conservation terms are preserved and the amount of data to be trained is not feasible when we investigate three-dimensional problems. The second scenario is to impose conservation laws as constraints (see [17]). It poses issues that training the network efficiently with these constraints and designing neural network architectures properly. The last one is the combination of the above two scenarios when the loss includes both penalties from the data and the physical rules. It has many advantages like not requiring meshes and can be scope with higher-order problems with automatic differentiation tools. Nevertheless, in [18], pointed out that PINNs fail to learn the relevant Physics in all cases they observed. For example, the PINNs performed badly in the advection equation when the velocity parameter is big and it also fails to solve the Fisher equation and the Burger equation when the diffusion terms are significant. Compared to the existent convention methods, the PINNs are slower due

to the full-batch optimization and the complexity of the neural network structure. That is the main reason that we follow the direction of using other families based on existing numerical methods to enhance the numerical solutions. The slight variation of PINNs is numerical Physics-Informed Neural Network (N-PINN) ([19]) that the structure of neural networks is adapted from familiar numerical schemes. In [20] and [21], they put more neural network layers in the relation between numerical solutions at the two consecutive times and apply the convection methods on the finer grids to obtain the synthesis training data. This is the offline training process for building up a simple model on the course grids with optimized parameters and has the same accuracy when solving on the fine grids. In chapter 5 of this thesis, we propose the continuations of this paper [22] where the sequence-to-sequence technique can be applied and the number of free parameters is generalized.

Chapter 1

Method of fundamental solutions for Laplace equations

The basic idea to use fundamental solutions for training was taken from the original work [23]. I have further developed this method to get the approach in Section 1.1. Furthermore, both approaches were implemented and tested in a number of cases. The basis of this chapter is my TDK work submitted in October 2021.

1.1 The first approach

1.1.1 The Laplace's equation with Dirichlet boundary conditions

Let us consider the Laplace equation

$$\begin{cases} \Delta u = 0 \quad \text{in} \ \ \Omega \\ u = g \quad \text{on} \ \ \partial\Omega \end{cases} \tag{1.1.1}$$

for the unknown function $u \in H^1(\Omega)$ on the domain $\Omega \subset \mathbb{R}^d$ with Dirichlet boundary conditions given by $g \in H^{\frac{1}{2}}(\partial\Omega)$.

In the sense of distribution, the fundamental solution of the Laplacian operator is a solution of the first equation in (1.1.1) if $\mathbf{0} \notin \Omega$. Using the notation ω_d for the surface of the d-dimensional unit sphere, this can be given explicitly as

$$\phi : \mathbb{R}^d \setminus \{\mathbf{0}\}, \quad \phi(x) = \begin{cases} -\frac{1}{2\pi}\log(|x|) & \text{for } d = 2 \\ \frac{1}{(d-2)\omega_d |x|^{d-2}} & \text{for } d \geq 3. \end{cases}$$

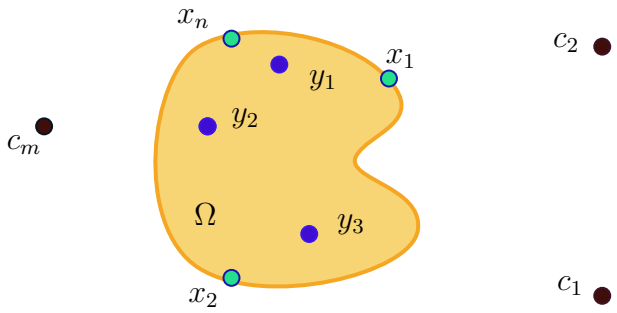


Figure 1.1: Inner (blue), boundary (green) and outer (brown) points.

Take n arbitrary boundary points $x_1, x_2, \ldots, x_n \in \partial\Omega$ and m arbitrary points $c_1, c_2, \ldots, c_m \in \Omega^C$, which will be called the outer points, see Figure 1.1. The fundamental solution associated with the point c_j in case of $d = 2$ is

$$\phi_{c_j}(x) = \phi(x - c_j) = -\frac{1}{2\pi} \log(|x - c_j|). \tag{1.1.2}$$

We will approximate the solution of (1.1.1) as a linear combination of the functions $\{\phi_{c_j}\}_{j=1}^m$. This method was first proposed in [2], whenever for its convergence on an arbitrary domain still no rigorous proof was developed.

A main motivation of this approximation is the fact that according to the theory of boundary integral equations [24], there is a unique function $\mathcal{G} \in H^{-\frac{1}{2}}(\partial\Omega)$ such that

$$u(x) = \int_{\partial\Omega} \phi(x - y)\mathcal{G}(y)\, \mathrm{d}y. \tag{1.1.3}$$

If this could be approximated with an appropriate sum, we would get a sum of some terms $\phi_{c_j^*}$. At the same time, on the boundary point c_j^*, the function $\phi_{c_j^*}$ becomes singular. Therefore, instead of c_j^*, we rather choose an outer point c_j close to c_j^*.

In any case, if we take more outer points, we can enhance accuracy of the approximation. In concrete terms, we are looking for the approximation

$$u(x) \approx \sum_{j=1}^{m} a_j \phi_{c_j}(x), \quad (+ \text{ constant}), \tag{1.1.4}$$

where the coefficients a_j will be determined by the boundary condition such that

$$\sum_{j=1}^{m} a_j \phi_{c_j}(x_i) \approx g(x_i) \quad i = 1, 2, \ldots, n. \tag{1.1.5}$$

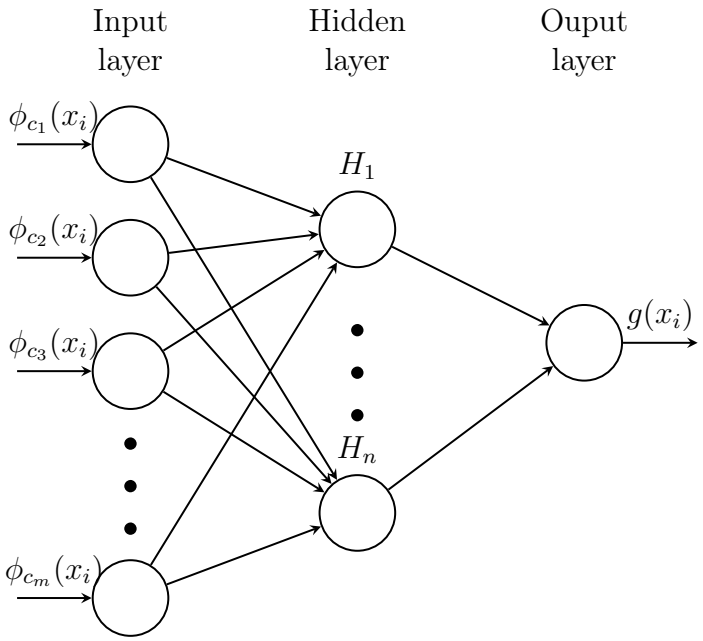


Figure 1.2: Neural network structure of the first approach

In the language of neural networks, we are looking for the parameters $a_1, a_2, \ldots, a_m$ such that the corresponding linear mapping in (1.1.4) delivers a minimal error in (1.1.5). A training data pair consisting an input and output data is associated to the boundary point x_i. Accordingly:

- The input of the training data includes n vectors of length m: $(\phi_{c_1}(x_i), \cdots, \phi_{c_m}(x_i))$, where $i = 1, \cdots, n$.
- The output of the training data are boundary function values at the point x_i : $g(x_i)$, where $i = 1, \ldots, n$).
- Weights in the most simple case (in lack of a hidden layer) are a_j and in the general case, these are components to get a_j.
- We use the least square loss function with regularization.

The setup of the corresponding neural network is displayed in Figure (1.2).

Using the gradient-based methods and tuning the parameters properly to minimize the mean squared error, we obtain the desired parameters $a_1, a_2, \ldots, a_m$. This model gives us the approximation of u at any inner point, let say y_k, then

$$u(y_k) \approx \sum_{j=1}^{m} a_j \phi_{c_j}(y_k).$$

1.1.2 The Laplace's equation with Neumann boundary conditions

The general interior Neumann problem for Laplace's equation in Ω can be formulated as follows:

$$\begin{cases} \Delta u = 0 \quad \text{in} \quad \Omega \\ \frac{\partial u}{\partial \mathbf{n}} = f_{\mathbf{n}} \quad \text{on} \quad \partial\Omega, \end{cases} \tag{1.1.6}$$

where $\mathbf{n}$ is a unit outward normal vector to the boundary $\partial\Omega$ and $f_{\mathbf{n}}$ is a prescribed function defined on the Lipschitz boundary $\partial\Omega$ of the domain $\Omega \subset \mathbb{R}^d$.

Again, we are looking for an approximation in form (1.1.4). At the same time, we have to replace the Dirichlet boundary conditions with Neumann type boundary conditions such that we obtain

$$\frac{\partial}{\partial \mathbf{n}} u(x_i) \approx \sum_{j=1}^{m} a_j \frac{\partial}{\partial \mathbf{n}} \phi_{c_j}(x_i). \tag{1.1.7}$$

Here using $c = [c_1, c_2]$ for a generic outer point and $x = [x_1, x_2]$ for a generic inner point, we have

$$\frac{\partial}{\partial x_1}\phi_c(x) = -\frac{1}{2\pi}\frac{x_1 - c_1}{(x_1 - c_1)^2 + (x_2 - c_2)^2}$$
$$\frac{\partial}{\partial x_2}\phi_c(x) = -\frac{1}{2\pi}\frac{x_2 - c_2}{(x_1 - c_1)^2 + (x_2 - c_2)^2}.$$

Accordingly, the new neural network should be altered in terms of the inputs and outputs as follows:

- Training data input includes vectors $\left(\frac{\partial}{\partial \mathbf{n}}\phi_{c_1}(x_i), \cdots, \frac{\partial}{\partial \mathbf{n}}\phi_{c_m}(x_i)\right)$, where $i = 1, \cdots, n$.
- Training data output are boundary function at the point x_i : $f_{\mathbf{n}}(x_i)$ $(i = 1, \cdots, n)$.

After training, we obtain the weights $a_1, a_2, \ldots, a_m$ in the approximation (1.1.4).

1.1.3 The Laplace's equation with mixed boundary conditions

For the definition of mixed boundary condition for the Laplace equation, we split $\partial\Omega$ of two disjoint parts, Γ_1 and Γ_2, such that $\partial\Omega = \Gamma_1 \cup \Gamma_2$. Then the problem

$$\begin{cases} \Delta u = 0 \quad \text{in } \Omega \\ u|_{\Gamma_1} = g \quad \text{and} \quad \frac{\partial u}{\partial \mathbf{n}}\big|_{\Gamma_2} = f \end{cases} \tag{1.1.8}$$

is called the Laplace equation with mixed boundary condition, which, for $f \in H^{-\frac{1}{2}}(\partial\Omega)$ and $g \in H^{\frac{1}{2}}(\partial\Omega)$ has a unique solution $u \in H^1(\Omega)$.

In this case, the neural network is a combination of the two networks above: the Dirichlet input-output training data is: $(\phi_{c_1}(x_i), \cdots, \phi_{c_m}(x_i))$ and $g(x_i)$ where $x_i \in \Gamma_1$ and the Neumann input-output training data includes $\left(\frac{\partial}{\partial \mathbf{n}}\phi_{c_1}(x_i), \cdots, \frac{\partial}{\partial \mathbf{n}}\phi_{c_m}(x_i)\right)$ and $f_{\mathbf{n}}(x_i)$ where $x_i \in \Gamma_2$.

1.2 The second approach

1.2.1 The Laplace's equation with Dirichlet boundary conditions

We discuss another approach to design the neural network, which uses only the fundamental solutions as learning data. Here, for each outer point c_j,

- The input of training data is a vector of length n: $\left(\phi_{c_j}(x_1), \cdots, \phi_{c_j}(x_n)\right)$.
- The output of training data is $\phi_{c_j}(y)$ where y is an interior point, where we want to estimate the solution.

This idea is visualized in Figure 1.3.

Here we assume that fundamental solution at any inner point can be written as linear combination of fundamental solution at boundary points as

$$\phi_{c_j}(y) \approx \sum_{i=1}^{n} b_i \phi_{c_j}(x_i). \tag{1.2.1}$$

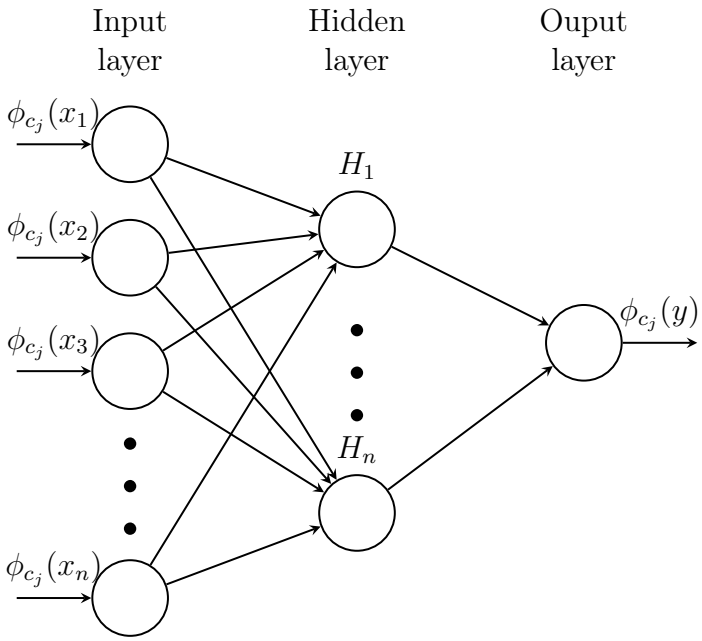


Figure 1.3: Neural network structure of the second approach

Using approximation (1.1.4), we get

$$u(y) \approx \sum_{j=1}^{m} a_j \phi_{c_j}(y) \approx \sum_{j=1}^{m} a_j \sum_{i=1}^{n} b_i \phi_{c_j}(x_i) = \sum_{i=1}^{n} \sum_{j=1}^{m} a_j b_i \phi_{c_j}(x_i) \approx \sum_{i=1}^{n} b_i u(x_i).$$

Hence, the numerical solution at the inner point y can be estimated by linear combination of this function at the boundary points.

1.2.2 The Laplace's equation with Neumann boundary conditions

We assume that at the source point c_j, the fundamental solution of the any points can be approximated by a linear combination of derivative of fundamental solution of the boundary points:

$$\phi_{c_j}(y) \approx \sum_{i=1}^{n} b_i \frac{\partial}{\partial \mathbf{n}} \phi_{c_j}(x_i).$$

Using the approximation in (1.1.4), we get

$$u(y) \approx \sum_{j=1}^{m} a_j \phi_{c_j}(y) \approx \sum_{j=1}^{m} a_j \sum_{i=1}^{n} b_i \frac{\partial}{\partial \mathbf{n}} \phi_{c_j}(x_i) = \sum_{i=1}^{n} \sum_{j=1}^{m} a_j b_i \frac{\partial}{\partial \mathbf{n}} \phi_{c_j}(x_i) \approx$$
$$\approx \sum_{i=1}^{n} b_i \frac{\partial}{\partial \mathbf{n}} u(x_i).$$

Accordingly, the structure of neural network is the following:

- Training data input includes vectors $\left(\frac{\partial}{\partial \mathbf{n}}\phi_{c_j}(x_1), \cdots, \frac{\partial}{\partial \mathbf{n}}\phi_{c_j}(x_n)\right)$, where $j = 1, \cdots, m$.
- The output of training data is $\phi_{c_j}(y)$ where y is an inner point.

1.2.3 The Laplace's equation with mixed boundary conditions

Let us consider the mixed boundary value problem with the conditions in (1.1.8). The second approach is estimating the fundamental solution at an inner point by both fundamental solution and derivative of fundamental solution at boundary points

$$\phi_{c_j}(y) \approx \sum_{x_i \in \Gamma_1} b_i \phi_{c_j}(x_i) + \sum_{x_i' \in \Gamma_2} b_i' \frac{\partial}{\partial \mathbf{n}} \phi_{c_j}(x_i'). \tag{1.2.2}$$

Indeed, the neural network will optimize the parameters $\{b_i\}$, $\{b_i'\}$ here.

Using approximation (1.1.4) again, we get

$$u(y) \approx \sum_{x_i \in \Gamma_1} b_i u(x_i) + \sum_{x_i' \in \Gamma_2} b_i' \frac{\partial}{\partial \mathbf{n}} u(x_i').$$

Hence, the output of training data in the second method is always the fundamental solution $\phi_{c_j}(y)$ and the input data in mixed boundary problem is

$$\left(\phi_{c_j}(x_1), \cdots, \phi_{c_j}(x_n), \frac{\partial}{\partial \mathbf{n}}\phi_{c_j}(x_1'), \cdots, \frac{\partial}{\partial \mathbf{n}}\phi_{c_j}(x_n')\right),$$

where $x_1, \cdots, x_n$ are boundary points generated on Γ_1 and $x_1', \cdots, x_n'$ are generated on Γ_2.

1.3 Numerical experiments

In this section, we dig deep into how these methods work on specific problems, how to set up the position of collocation points, source points and how to tune parameters properly. In particular, we focus to the following questions.

- What is the optimal number (or rather: the ratio) of the boundary and outer points?

Figure 1.4: The outer and boundary points for example 1

- What is the optimal distance of the outer points from the boundary?
- What is the optimal distribution of the boundary and outer points?
- Does the two approaches deliver similar accuracy?
- With an optimal choice of all parameters, which convergence rate can be achieved?
- What is the setup and the parameters in a neural network used in the computations?

1.3.1 Dirichlet boundary condition on the unit square

Example 1. In a unit square $\Omega = \{x, y \mid 0 < x < 1, 0 < y < 1\}$ let us consider the following equation

$$\begin{cases} \Delta u = 0 & \text{for } (x,y) \in \Omega \\ u(x,0) = 0,\ u(x,1) = \sin(\pi x) & \text{for } 0 < x < 1 \\ u(0,y) = 0,\ u(1,y) = 0 & \text{for } 0 < y < 1. \end{cases} \tag{1.3.1}$$

The analytical solution of this problem is:

$$u(x,y) = \frac{1}{e^{\pi} - e^{-\pi}} \sin(\pi x) \left(e^{\pi y} - e^{-\pi y}\right).$$

Firstly, we implement the first method by choosing the position of outer points on a square which is obtained by a magnification of the unit square (see Figure 1.4). We define the magnification factor ϵ as the distance between two parallel sides of each square.

In Figure 1.5 (left figure), we use the following parameters:

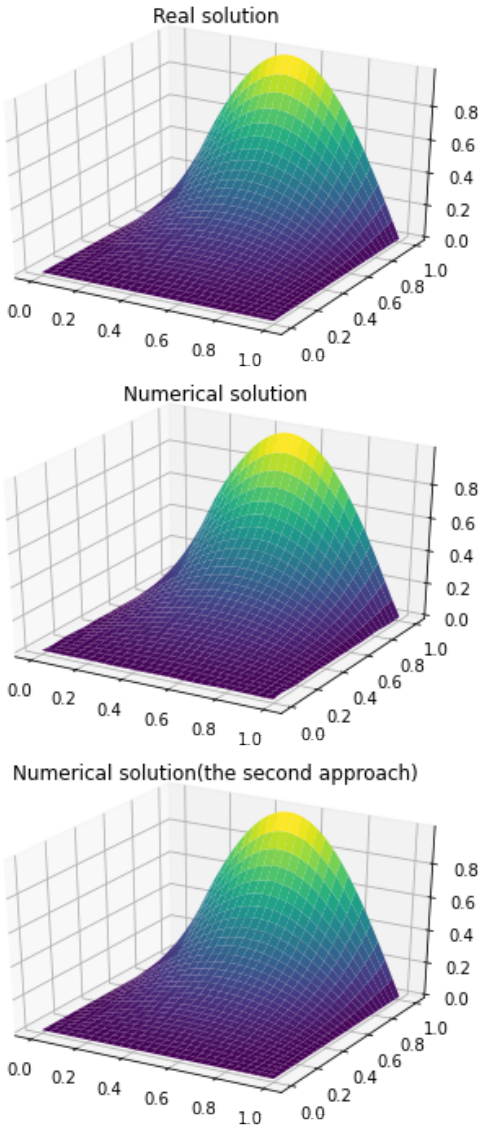


Figure 1.5: Analytic and numerical solutions are solved by the first approach (left) and the second approach (right) of example 1

- number of boundary points: 32,
- number of outer points: 8192,
- magnification factor: $\epsilon = 0.1$,
- learning rate: 0.1,
- number of epochs =1000,
- layers: 1 hidden layer with size of 100,
- loss function after training: 0.0003839250421151519.

The number of boundary and outer points	Error (Frobenius norm)
$n = 16, m = 64$	0.2461622809658327
$n = 32, m = 64$	0.3182853004347026
$n = 32, m = 128$	0.21974143048835057
$n = 16, m = 128$	0.18014076148115063
$n = 16, m = 512$	0.10919759728510069
$n = 16, m = 2048$	0.07990843843706859
$n = 32, m = 8192$	0.04762830527750495

Table 1.1: The effect of the number of boundary points and outer points (the first approach)

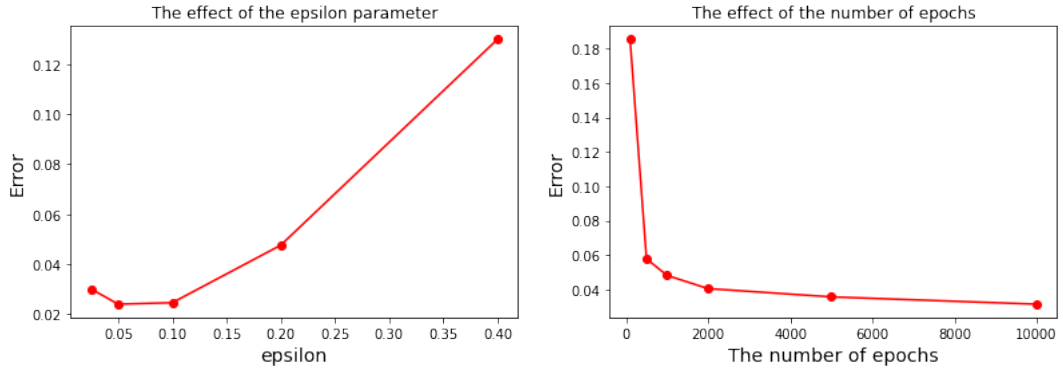


Figure 1.6: The effect of ϵ and the number of epochs(the first approach)

The role of the number of boundary points and source points is important. In Table 1.1, we run on different values of n(the number of boundary points) and m (the number of boundary points) with $\epsilon = 0.2$. We observe that the more outer points we generate, the more accuracy we get for the Frobenius norm. The number of outer points should exceed the number of boundary points because we can approximate the boundary values more accurately using a long linear combination of fundamental solutions. Note that the solution given by the neural network is not unique, so we may get results far away from the true solution but have small losses. Luckily, in this example, the divergence between the two solutions is insignificant.

The distance between two squares can change the error slightly, but the numerical solution is close to the analytical solution in general. According to the Figure 1.6 (left), ϵ should be between 0.05 and 0.1 to get an optimal numerical solution. Increasing the number of epochs will also decrease the error (see Figure 1.6 -right).

Now we turn to the second approach. We want to plot the whole function in this example to compare it to the first method. We expect to get the value of every

Collocation points	True solution	$(100, 100)$	$(400, 400)$	$(1000, 1000)$	$(1000, 100)$	$(100, 1000)$
$(0.2, 0.2)$	0.034124	0.03115696	0.0338244	0.03507766	0.03429693	0.03419725
$(0.5, 0.5)$	0.199268	0.2008082	0.20086792	0.20041794	0.19841382	0.19588123
$(0.7, 0.7)$	0.311947	0.3144576	0.3115651	0.3125463	0.3116747	0.3061932
$(0.2, 0.7)$	0.226643	0.22474924	0.22370611	0.22674346	0.22672111	0.2225104
$(0.7, 0.2)$	0.046969	0.04574379	0.04649694	0.04676051	0.04662241	0.04642019

Table 1.2: The different choices of the number of boundary points and outer points, $\epsilon = 0.15$(the second approach)

Collocation points	True solution	$\epsilon = 0.1$	$\epsilon = 0.15$	$\epsilon = 0.2$	$\epsilon = 0.025$
$(x, y) = (0.2, 0.2)$	0.034124	0.0347167	0.03429693	0.03295259	0.03433963
$(x, y) = (0.5, 0.5)$	0.199268	0.19874202	0.19841382	0.2012808	0.19995686
$(x, y) = (0.7, 0.7)$	0.311947	0.3124724	0.3116747	0.3126797	0.3119331
$(x, y) = (0.2, 0.7)$	0.226643	0.22637717	0.22672111	0.22672312	0.2267626
$(x, y) = (0.7, 0.2)$	0.046969	0.04597505	0.04662241	0.04752169	0.04735678

Table 1.3: The different choices of magnification factor (the second approach), $n = 1000, m = 100$

grid point precisely, not only the network in general. It enhances considerably the complexity of the neural network and the time to train the model. It is an obstacle when tuning the parameters, so our technique is to train the network to learn the value in one grid point. Then we apply these parameters in the neural network to all other points. Figure 1.5(right) illustrates the numerical solution of the second approach with 320 boundary points, 60 source points and $\epsilon = 0.025$.

However, the purpose of using both approaches is not to estimate every point on the domain. It measures specified source points without using discretization on this domain (i.e.the mesh-free method). In Table 1.2, we use different values including the number of boundary points and the number of outer points. It shows that the second method performs moderately well even though the number of boundary points is smaller or larger or equal to the number of outer points.

Similar to the first method, the loss decreases if the auxiliary boundary is closer to the original boundary $\partial\Omega$ (or ϵ is smaller) which can be seen in Table 1.3.

Position of inner point	Analytical solution	Approach 2	Approach 1
$(x, y) = (0.2, 0.2)$	0.034124989219717516	0.03417236	0.03415351
$(x, y) = (0.5, 0.5)$	0.19926840766919335	0.1992962	0.19928508
$(x, y) = (0.7, 0.7)$	0.311947830115096	0.31381178	0.31188422
$(x, y) = (0.2, 0.7)$	0.22664336509760868	0.22663322	0.22663559
$(x, y) = (0.7, 0.2)$	0.04696901819829289	0.04670608	0.04691111

Table 1.4: The approximation of some interior points using both methods.

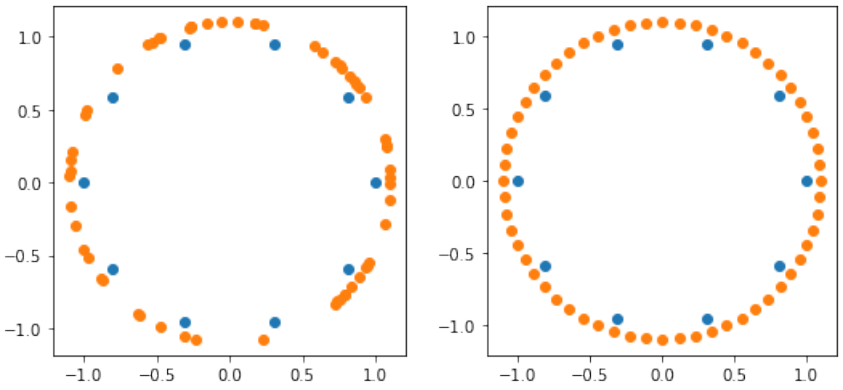


Figure 1.7: Nodal distribution of boundary and outer points.

1.3.2 Dirichlet boundary condition on the unit circle

Let us consider the Laplace equation (1.1.1) on the unit circle $\Omega = \{x, y | x^2 + y^2 \leq 1\}$, where the boundary condition is given with $u(x, y) = e^x \cos y$ if $x^2 + y^2 = 1$. We generate n junctions in the unitary circle in random direction (left) and uniformly (right) in Figure 1.7. We have observed that only uniform nodal distribution leads to an accurate approximation, large deviation between the distances of the nodal points become to large computational error.

In Figure 1.8 ,we use 20 boundary points, 600 outer points, $\epsilon = 0.1$, epoch $= 2500$. The loss after training neural network is 0.001734, and the error with respect to Frobenius norm is 0.04012585269.

We also use the second method to evaluate the numerical solution at some points: $(0, 0); (-0.5, -0.5); (0, 0.5); (0.6, -0.7)$ and can be illustrated in Table 1.5.

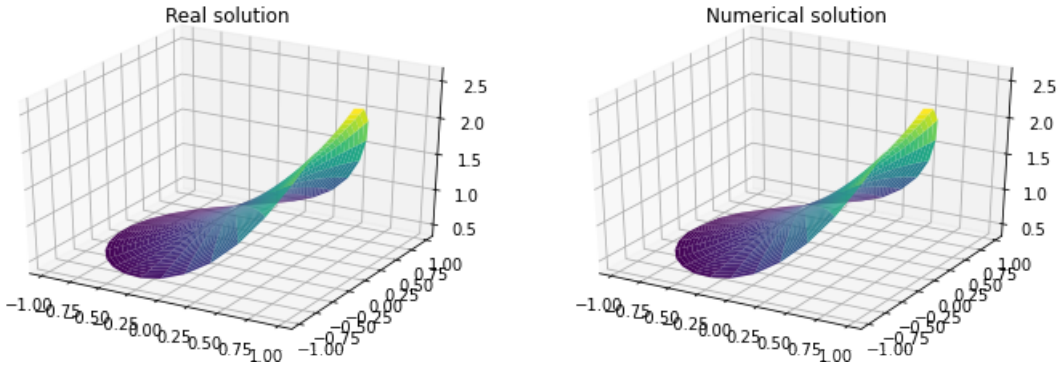


Figure 1.8: The analytical and numerical solution on the unit circle

Collocation points	True solution	n=m=100, $\epsilon = 0.1$	n=m=100, $\epsilon = 0.15$
$(x, y) = (0, 0)$	1	0.9903376	1.0000083
$(x, y) = (-0.5, -0.5)$	0.53228073	0.5215831	0.53341687
$(x, y) = (0, 0.5)$	0.87758256	0.86393476	0.8772203
$(x, y) = (0.6, -0.7)$	1.3936333	1.3846822	1.3980508

Table 1.5: The numerical solution of specific points on the unit circle (the second approach).

1.3.3 Dirichlet boundary conditions on the epitrochoid

We consider the Laplace equation 1.1.1 on the epitrochoid, where the boundary points have the position of

$$\left(\sqrt{(a+b)^2+1-2(a+b)\cos\left(\frac{a\theta}{b}\right)}\cos\theta, \sqrt{(a+b)^2+1-2(a+b)\cos\left(\frac{a\theta}{b}\right)}\sin\theta\right).$$

In this example, we choose $a = 4, b = 1$, the value on boundary is $u(x, y) = e^x \cos y$. Figure 1.9 is obtained by using 60 boundary points, 90 outer points, $\epsilon = 0.1$ and 10000 epochs with the first approach.

1.3.4 Case study for Neumann type boundary conditions

Due to the non-uniqueness of the solution in the problem containing only Neumann type boundary conditions, we investigate only the shape of numerical solutions. The program gives different results each time we implement, so we only show the best results. For the first approach, we run on the unit square.

Example 2. Consider in rectangular domain $\Omega = (0, 1) \times (0, 1)$ using Cartesian

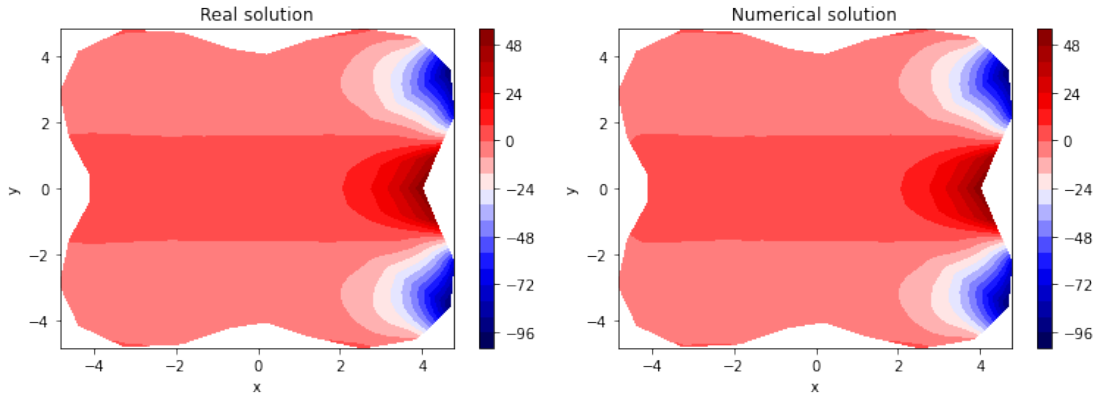


Figure 1.9: The analytical and numerical solution on epitrochoid

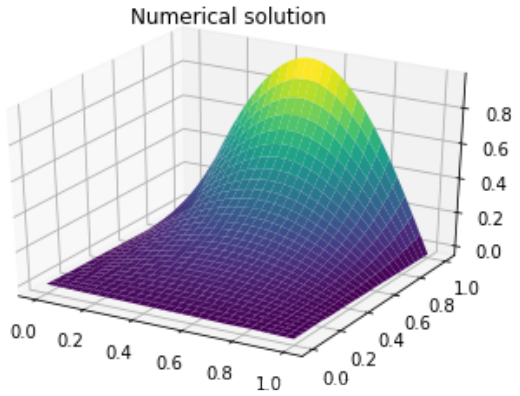


Figure 1.10: The numerical solution of the Neumann problem using the first approach.

coordinates

$$\begin{cases} \Delta u = 0 & \text{for } (x,y) \in \Omega \\ \dfrac{\partial}{\partial y}u(x,0) = 2\dfrac{\pi}{e^{\pi}-e^{-\pi}}\sin(\pi x), & \\ \dfrac{\partial}{\partial y}u(x,1) = \dfrac{\pi}{e^{\pi}-e^{-\pi}}(e^{\pi}+e^{-\pi})\sin(\pi x) & \text{for } 0<x<1 \\ \dfrac{\partial}{\partial x}u(0,y) = \dfrac{\pi}{e^{\pi}-e^{-\pi}}(e^{\pi y}-e^{-\pi y}), & \\ \dfrac{\partial}{\partial x}u(1,y) = \dfrac{-\pi}{e^{\pi}-e^{-\pi}}(e^{\pi y}-e^{-\pi y}) & \text{for } 0<y<1 \end{cases} \tag{1.3.2}$$

The function u satisfying these equations is the same in Example 1, up to an arbitrary additive constant term. Figure 1.10 displays a result we got, where the error is equal to 0.5968903065215189.

For the second approach, we run on the unit circle (see section 4.2), the Neumann conditions is given. The analytical solution is $u(x) = e^x \cos y + c$. In this example, we try to get the numerical solution is close to the analytical solution with $c = 0$.

Collocation points	$(0,0)$	$(-0.5,-0.5)$	$(0,0.5)$	$(0.6,-0.7)$
Analytical solution	1	0.53228073	0.87758256	1.39363332
Numerical solution	0.9999979	0.53227895	0.87758255	1.3936331

Table 1.6: The numerical solution at some points for the Neumann boundary value problem using the second approach.

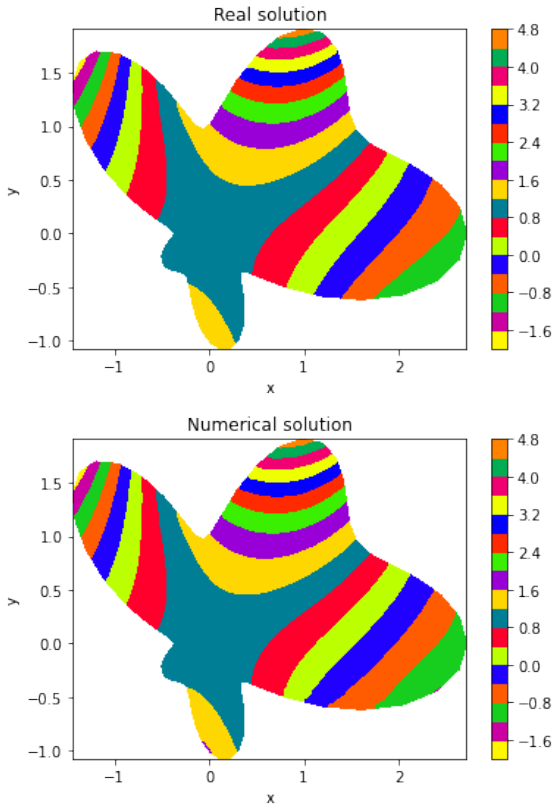


Figure 1.11: The analytical and numerical solution on Amoeba-like domain

1.3.5 Mixed boundary value problem on an Amoeba-like domain

Let us consider the Laplace equation 1.1.1 on the Amoeba-like domain, where the boundary points have the position

$$\left(e^{\sin\theta}\sin^2(2\theta)+e^{\cos\theta}\cos^2(2\theta)\right)(\cos\theta,\sin\theta)\,.$$

For $0\leq\theta<\pi$, Dirichlet boundary conditions are given, while for $0\leq\theta<2\pi$ we prescribe Neumann boundary conditions such that the analytic solution is $u(x,y)=\cos(x)\cosh(y)+\sin(x)\sinh(y)$.

Figure 1.11 is obtained by using 60 boundary points, 90 outer points, $\epsilon=0.1$ and 10000 epochs. We have also implemented the second method for this problem. Using

Collocation points	$(0,0)$	$(1,1)$	$(2,0)$	$(-1,1)$
Analytical solution	1	1.82262773	-0.41614683	-0.15516768
Numerical solution	1.0056722	1.823955	-0.43739626	-0.15932003

Table 1.7: The numerical solution of some points on the mixed boundary problem using the second approach

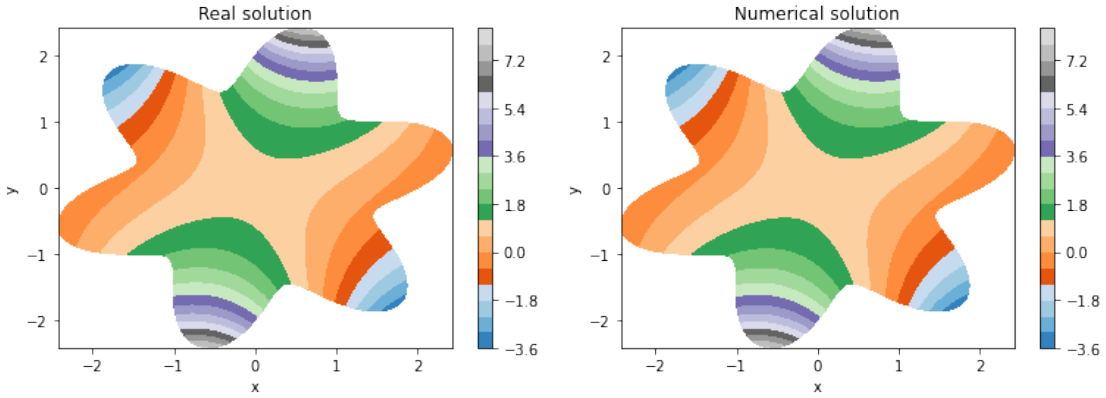


Figure 1.12: The analytical and numerical solution on flower-like domain

50 boundary points and 200 outer points, $\epsilon = 0.1$ with 90000 epochs, the result of some points is showed in Table 1.7.

1.3.6 Mixed boundary condition problem on a flower-shaped domain

Let consider the Laplace equation 1.1.1 on the flower-like domain, where the boundary points have the position

$$\left(2 + \frac{1}{2}\sin(k\theta)\right)(\cos x, \sin x)\,,$$

where k is a given integer. The boundary conditions in the domain In a model problem with $k = 6$, Dirichlet boundary condition was used for $0 \leq \theta < \pi$ and Neumann boundary condition for $0 \leq \theta < 2\pi$ such that the analytic solution is $u(x, y) = \cos(x)\cosh(y) + \sin(x)\sinh(y)$. Figure 1.12 is obtained by using 60 boundary points, 90 outer points, $\epsilon = 0.1$ and 10000 epochs.

1.3.7 Different neural networks and comparisons.

Table 1.8 and 1.9 illustrates the results after implementing in different set up of neural networks for both methods. The more complex neural networks we use in the

Neural network	Error
Linear Regression, no hidden layer, epochs = 1000	0.048206
Neural network, 1 hidden layer of size 100, epochs = 1000	0.026957
Neural network, 1 hidden layer of size 500, epochs = 1000	0.026286
Neural network, 2 hidden layers of size 100, epochs = 1000	0.022334
CNN with 2 fully connected layers, epochs = 10000	0.017556

Table 1.8: Different neural network structures for Example 1.

first approach, the more accuracy we get with the same number of epochs. But also note that if we use complex neural networks, there are more unknown parameters that cost the computation time. For example 1, we recommend using the neural network with one hidden layer of "small" size. The second approach, otherwise, does not affect by the structure of the neural network. We can achieve a reasonably good numerical solution using a simple network with no hidden layer by increasing the number of epochs.

To sum up, both methods are required a large enough number of outer points because both use the approximation (1.1.4). In practice, if the number of exterior points is more than 30 and the ratio between the number of boundary and outer points is small (<1), we will get a reliable numerical solution. However, for the second approach, a large enough number of boundary points has to be used to guarantee that the approximation (1.2.1) or (1.2.2) has a small error. It avoids the error term from approximation (1.1.4) directly and works for any ratio between the number of boundary and outer points. Hence, the second method is applicable for the problem which generating the equidistant outer points ineffectively, while the first method is an incredibly good estimation when the auxiliary domain is formed effectively.

In a nutshell, with the optimal choice of all parameters, both methods operate accurately with low error on the "smooth" boundary curve (see Table 1.4).

Neural network	$(0, 0)$	$(1, 1)$	$(2, 0)$	$(-1, 1)$
Linear Regression, no hidden layer, epochs = 60000	1.001674	1.822747	-0.4326400	-0.1545439
Neural network, 1 hidden layer of size 100, epochs = 90000	0.998864	1.820469	-0.4302049	-0.1552322
Neural network, 2 hidden layers of size 100, epochs = 90000	1.000284	1.821235	-0.4321503	-0.1545273
CNN with 2 fully connected layers, epochs = 300000	1.002132	1.820958	-0.4312253	-0.1556389
Analytical solution	1	1.8226277	-0.4161468	-0.1551676

Table 1.9: Different neural network structures for Amoeba-like domain problem

Chapter 2

Method of fundamental solutions for Helmholtz equations

2.1 Solving Helmholtz-type equations

Let us consider the two types of boundary homogeneous Helmholtz-type equation, including the monotone type

$$\begin{cases} \Delta^2 u - k_1^2 u = 0 \quad \text{on } \Omega \\ u|_{\Gamma_1} = g \quad \text{and} \quad \frac{\partial u}{\partial \mathbf{n}}\big|_{\Gamma_2} = f. \end{cases} \tag{2.1.1}$$

and the oscillatory type

$$\begin{cases} \Delta^2 u + k_2^2 u = 0 \quad \text{on } \Omega \\ u|_{\Gamma_1} = g \quad \text{and} \quad \frac{\partial u}{\partial \mathbf{n}}\big|_{\Gamma_2} = f, \end{cases} \tag{2.1.2}$$

where $\Omega \subset \mathbb{R}^n$ is an open set and $\partial\Omega$ consists of two disjoint parts, Γ_1 and Γ_2, such that $\partial\Omega = \Gamma_1 \cup \Gamma_2$.

The fundamental solutions of the modified (2.1.1) and the original (2.1.2) Helmholtz problems, respectively are given by

$$\phi_-(x,y) = \begin{cases} \frac{1}{2\pi} K_0(k_1\|x-y\|) & \text{for} \quad d=2 \\ \dfrac{e^{-k_1\|x-y\|}}{4\pi\|x-y\|} & \text{for} \quad d=3 \end{cases} \tag{2.1.3}$$

and

$$\phi_+(x,y) = \begin{cases} \frac{i}{4} H_0^{(1)}(k_2\|x-y\|) & \text{for} \quad d=2 \\ \dfrac{e^{-ik_2\|x-y\|}}{4\pi\|x-y\|} & \text{for} \quad d=3, \end{cases} \tag{2.1.4}$$

where K_0 denotes the modified Bessel function of second kind of order zero and H_0^1 denotes the Hankel function of the first (See the Appendix A). In practice, the function u is a real-valued function. Hence, we can assume the fundamental solution of the oscillatory type problem is a real-valued function, namely

$$\phi_+(x,y) = \begin{cases} \frac{-1}{4} Y_0(k_2\|x-y\|) & \text{for} \quad d=2 \\ \dfrac{\cos(-k_2\|x-y\|)}{4\pi\|x-y\|} & \text{for} \quad d=3, \end{cases} \tag{2.1.5}$$

where Y_0 is the Bessel function of the second kind of order zero kind of order zero. Using the method of fundamental solution (MFS), we estimate the solution as a data-dependent linear combination

$$u(x) \approx \sum_{j=1}^{m} a_j \phi_{c_j}(x), \tag{2.1.6}$$

where $\phi(x)$ is the fundamental solution for the corresponding problem.
From this, we construct the neural network design to find the coefficient $\{a_j\}_{j=1}^m$ using the Dirichlet input-output training data:

$$\left[\left(\phi_{c_1}(x_i), \cdots, \phi_{c_m}(x_i)\right) g(x_i)\right],$$

where $x_i \in \Gamma_1$. Similarly, the Neumann input-output training data includes the pairs

$$\left[\left(\frac{\partial}{\partial \mathbf{n}}\phi_{c_1}(x_i), \cdots, \frac{\partial}{\partial \mathbf{n}}\phi_{c_m}(x_i)\right) f_{\mathbf{n}}(x_i)\right],$$

where $x_i \in \Gamma_2$.

The output of training data of number j in the second method is always $\phi_{c_j}(y)$ and the input data in case of the mixed boundary value problem is

$$\left(\phi_{c_j}(x_1), \cdots, \phi_{c_j}(x_n), \frac{\partial}{\partial \mathbf{n}}\phi_{c_j}(x_1'), \cdots, \frac{\partial}{\partial \mathbf{n}}\phi_{c_j}(x_n')\right),$$

where $x_1, \cdots, x_n$ are boundary points generated on Γ_1, while $x_1', \cdots, x_n'$ are generated on Γ_2.

2.2 Numerical results

2.2.1 Solving the modified Helmholtz equation on sphere

In the first example, the three-dimensional eigenvalue problem for the Laplace operator problem is selected to divulge the competence of the proposed MFS. The monotone type Helmholtz problem is with Dirichlet boundary conditions is the following:

$$\begin{cases} \Delta^2 u - 3u = 0 & \text{on } \Omega \\ u(x,y,z) = \sinh(x+y+z) & \text{on } \partial\Omega, \end{cases} \tag{2.2.1}$$

where $\Omega = \{(x,y,z)|x^2+y^2+z^2 < 1\}$.
Before injecting the training data into the neural network, we should distribute the boundary and auxiliary boundary surface such that points are evenly distributed. We list four common distributions below.

a) The standard equiangular spherical coordinate distribution forming N_1 lines of latitude and N_2 lines of longitude.

b) There are $6N^2+6N+2$ points generated in the sphere which has positions

$$\left(\sin\left(\frac{n\pi}{2N}\right)\sin\left(\frac{k\pi}{3n}\right), \sin\left(\frac{n\pi}{2N}\right)\cos\left(\frac{k\pi}{3n}\right), \cos\left(\frac{n\pi}{2N}\right)\right)$$

where $k = 1, \cdots, 6n$ and $n = 1, \cdots N$.

c) The Fibonacci lattice is expressed as a sequence of N points with coordinates

$$\begin{aligned} x_k &= \sin(k\pi(3-\sqrt{5}))\sqrt{1-\left(1-\left(\frac{k}{N-1}\right)^2\right)^2} \\ y_k &= 1-\left(\frac{k}{N-1}\right)^2 \\ z_k &= \cos(k\pi(3-\sqrt{5}))\sqrt{1-\left(1-\left(\frac{k}{N-1}\right)^2\right)^2}, \end{aligned}$$

where k runs from 0 to $N-1$

Collocation points	Error(dis.1)	Error(dis.2)	Error(dis.3)	Error(dis.4)
$(0.6, 0.6, 0.1)$	-0.007686	0.00203109	0.00108302	-1.1682e-5
$(0.3, 0.3, 0.3)$	-0.021422	0.00585616	0.00275302	-0.0003191
$(0, 0, 0)$	-0.028143	0.00775653	0.00363088	-0.000422
$(-0.6, 0.7, -0.3)$	-0.000238	0.0002373	-0.0003359	-0.0005813

Table 2.1: Different nodal distribution leads to different error distribution

d) (Golden spiral method)

$$x_k = \left(1 - 2\frac{k}{N-1}\right)\sin(k\pi(1+\sqrt{5}))$$
$$y_k = \left(1 - 2\frac{k}{N-1}\right)\cos(k\pi(1+\sqrt{5}))$$
$$z_k = \pm\sqrt{1 - \left(1 - 2\frac{k}{N-1}\right)^2},$$

where $k = 0, \cdots, N-1$.

For comparison, we generate $256 = 16 \times 16$ boundary points and $1444 = 38 \times 38$ outer points for the standard equiangular coordinates. For the distribution in (b), 254 ($N = 6$) points was created in boundary and 1442($N = 38$) points in the pseudo-boundary. We both set 255 boundary points and 1443 outer points on Fibonacci lattice and spiral distribution. These distributions are visualized in Figure 2.1.

2.2.2 Solving modified Helmholtz equation on the cube

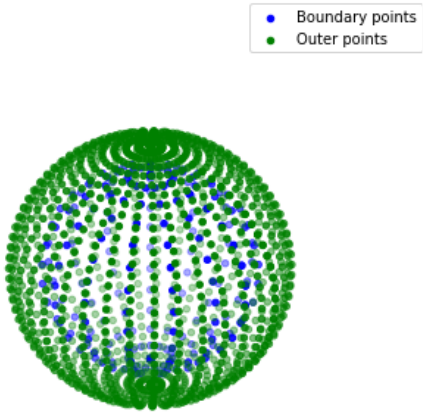
We first solve the following monotone type Helmholtz problem with Dirichlet boundary conditions:

$$\begin{cases} \Delta^2 u - 3u = 0 & \text{on } \Omega \\ u(x, y, z) = \sinh(x + y + z) & \text{on } \partial\Omega, \end{cases} \tag{2.2.2}$$

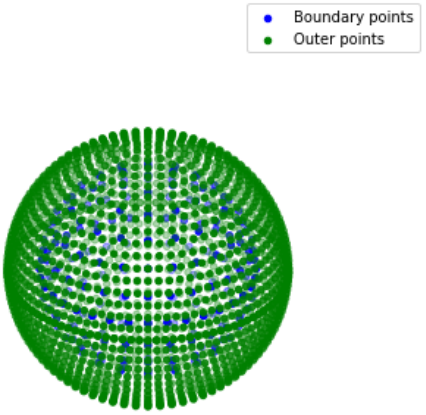
where Ω is the unit cube $(0, 1) \times (0, 1) \times (0, 1)$.

Point Distribution: Discretizing in each direction using 10 points on the boundary surface and using 40 points on the outer domain. The size of input data: 488×9128 (see Figure 2.2). Parameters: 1 hidden layer of size 100, learning rate = 0.05, epoch = 20000.

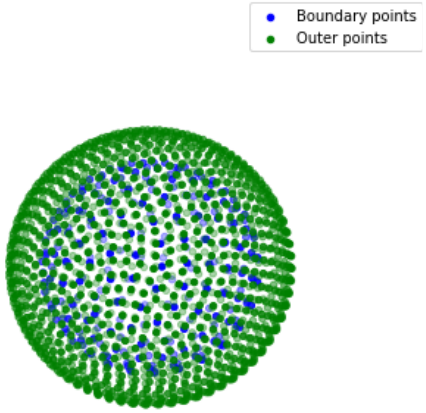
We estimate the error on some levels: $z = 0.1$ (maximum norm of error vector is 0.0162) ; $z = 0.5$ (maximum norm of error vector is 0.0378) ; $z = 0.8$ (maximum



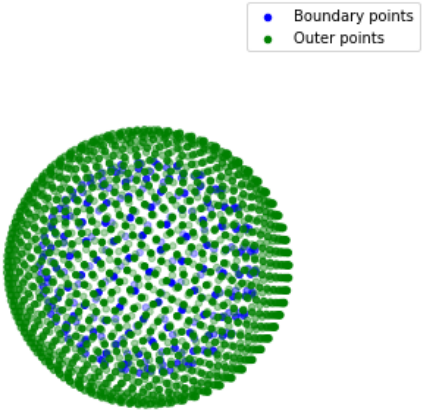
(a) Standard equiangulation



(b) Enhanced standard distribution



(c) Fibonacci lattice



(d) Golden spiral method

Figure 2.1: Nodal distributions on the sphere.

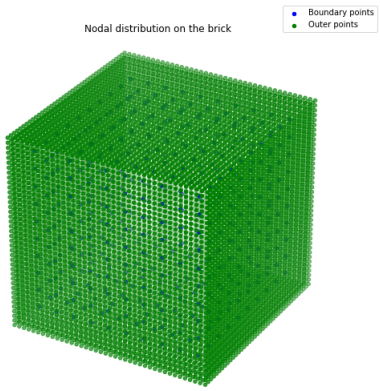


Figure 2.2: Boundary and outer points for the problem (2.2.2).

norm of error vector is 0.0268). In Figure 2.3, we show the error function on the different levels.

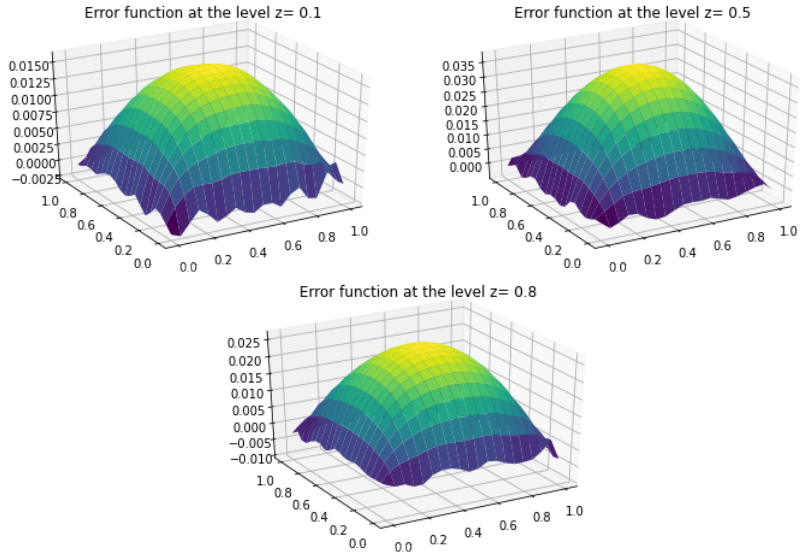


Figure 2.3: Error function on the different levels

2.2.3 Solving modified Helmholtz equation on the brick

$$\begin{cases} \Delta^2 u - \frac{3}{64}u = 0 & \text{on } \Omega \\ u(x,y,z) = \sinh(\frac{x}{8} + \frac{y}{8} + \frac{z}{8}) & \text{on } \partial\Omega \end{cases} \tag{2.2.3}$$

We consider the same above problem but on the brick of size $8 \times 5 \times 3.5$. Step size for boundary domain is: 0.5 m and stepsize for outer domain is 0.125 m, distance between two domains is 0.3m. (See Figure 2.4). The error function for the points $(x, y, 1)$ is illustrated in the Figure 2.5

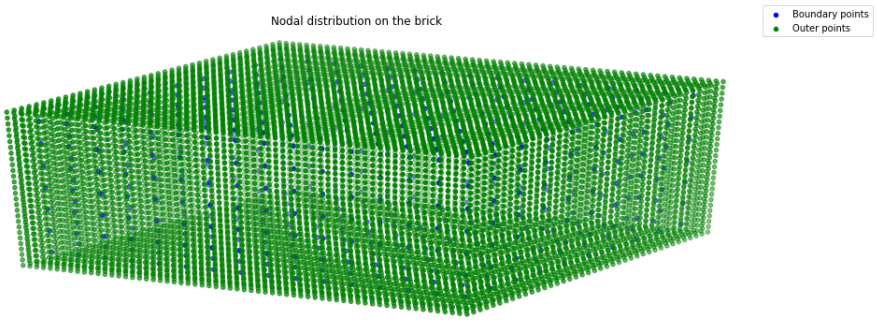


Figure 2.4: Computation domain for the brick problem.

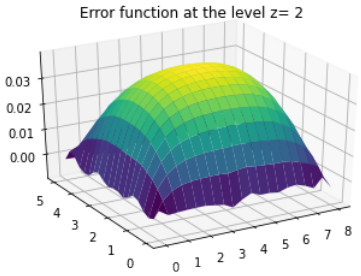


Figure 2.5: Error function on the level $z = 1$.

Chapter 3

Method of particular solutions

I partly follow in this chapter the recent work [4], where the original idea was presented. According to some papers: [25] , [26], [27] , [28], we summarize the main ideas here and complete it with some remarks and observations. The corresponding numerical experiments were implemented myself and using the neural networks approach to execute.

3.1 The method for Poisson equation

Let us consider the Poisson equation

$$\begin{cases} \Delta u = f \quad \text{in} \quad \Omega \\ u = g \quad \text{on} \quad \partial\Omega \end{cases} \tag{3.1.1}$$

for the unknown function $u \in H^1(\Omega)$ on the domain $\Omega \subset \mathbb{R}^d$ with Dirichlet boundary conditions given by $g \in H^{\frac{1}{2}}(\partial\Omega)$.
We decompose the solution of the equation in (3.1.1) into two parts:

$$u = u_p + u_l,$$

where u_p is the particular solution

$$\Delta u_p = f \tag{3.1.2}$$

and u_l is the solution of the Laplace equation

$$\begin{cases} \Delta u_l = 0 \quad \text{in} \quad \Omega \\ u_l = g - u_p \quad \text{on} \quad \partial\Omega. \end{cases} \tag{3.1.3}$$

To solve equation (3.1.3), we use the algorithm (MFS) in Chapter 1.
To find the particular solution representation of equation (3.1.2), we use radial basis function interpolation technique. The right hand side function of the Laplace operator can be represented as linear combination of radial basis functions (RBFs)

$$f(x) = \sum_{i=1}^{n} a_i \varphi(\|x - c_i\|) \tag{3.1.4}$$

The collocation points $\{c_i\}_{i=1}^n$ are selected such that the function f is well-approximated on the domain Ω. Substituting $x = c_i$ into equation (3.1.4), we obtain the system of linear equations

$$\begin{bmatrix} \varphi(c_1 - c_1) & \varphi(c_2 - c_1) & \cdots & \varphi(c_n - c_1) \\ \varphi(c_1 - c_2) & \varphi(c_2 - c_2) & & \varphi(c_n - c_2) \\ \vdots & & \ddots & \\ \varphi(c_1 - c_n) & \varphi(c_2 - c_n) & \cdots & \varphi(c_n - c_n) \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} f(c_1) \\ f(c_2) \\ \vdots \\ f(c_3) \end{bmatrix} \tag{3.1.5}$$

If we choose the collocation points well, the system of linear equations system delivers a unique set of coefficients $\{a_i\}_{i=1}^n$, which gives obviously an approximation of u_p as follows:

$$u_p(x) = \sum_{i=1}^{n} a_i \psi(\|x - c_i\|). \tag{3.1.6}$$

Here ψ is defined implicitly as

$$-\Delta\psi = \varphi \tag{3.1.7}$$

There are several possibilities to choose the radial basis function ϕ as shown in Table 3.1. The calculation for the corresponding ψ was proposed in [29]

Theorem 3.1.1.

- *If the RBF is thin plate splines, the particular solution for* (3.1.7) *is*

$$\psi(r) = \begin{cases} \dfrac{r^{2n+2}\log r}{4(n+1)^2} - \dfrac{r^{2n+2}}{4(n+1)^3} & \text{in } \mathbb{R}^2 \\ \dfrac{(2n)! r^{2n+2}}{(2n+2)!} & \text{in } \mathbb{R}^3. \end{cases}$$

- *If the RBF is inverse multiquadratics* $\phi(r) = \dfrac{1}{\sqrt{r^2 + c^2}}$, *the particular solution*

Type of RBF	function
linear	$\varphi(r) = r$
ad-hoc	$\varphi(r) = 1 + r$
cubic	$\varphi(r) = r^3$
thin plate splines	$\varphi_n(r) = \begin{cases} r^{2n} \log r & \text{in } \mathbb{R}^2 \\ r^{2n-1} & \text{in } \mathbb{R}^3 \end{cases}$
multiquadric	$\varphi_c(r) = (r^2 + c^2)^{(2k+1)/2}$
compactly supported	$\varphi_\lambda(r) = \begin{cases} \left(1 - \frac{r}{\lambda}\right)^2 & \text{if } r \le \lambda \\ 0 \quad \text{if } r > \lambda. \end{cases}$

Table 3.1: Common types of radial basis functions.

for 3.1.7 is

$$\psi(r) = \begin{cases} \sqrt{r^2 + c^2} - c \log(c + \sqrt{r^2 + c^2}) & \text{in } \mathbb{R}^2 \\ \dfrac{r^2 + c^2}{2} + \dfrac{c^2}{2r} \log\left(r + \dfrac{r^2 + c^2}{c}\right) - c & \text{in } \mathbb{R}^3. \end{cases}$$

- *If the BPF is multiquadratics $\phi(r) = \sqrt{r^2 + c^2}$, the particular solution for 3.1.7 is*

$$\psi(r) = \begin{cases} \dfrac{4c^2 + r^2}{9}\sqrt{r^2 + c^2} - \dfrac{c^3}{3} \log(c + \sqrt{r^2 + c^2}) & \text{in } \mathbb{R}^2 \\ \left(\dfrac{5c^2}{24} + \dfrac{r^2}{12}\right)\sqrt{r^2 + c^2} + \dfrac{c^4 \left[\ln(r + \sqrt{r^2 + c^2}) - \ln c\right]}{8r} - \dfrac{c^3}{3} & \text{in } \mathbb{R}^3. \end{cases}$$

3.2 The method for modified Helmholtz equations

Let us consider the modified Helmholtz equation

$$\begin{cases} \Delta u - k^2 u = f & \text{in } \Omega \\ u = g & \text{on } \partial\Omega. \end{cases} \tag{3.2.1}$$

A similar method can be applied to solve this equation. We use the approximation

$$u(x) = \sum_{i=1}^{n} a_i \psi(\|x - c_i\|) + \sum_{j=1}^{m} b_j \phi(\|x - d_j\|). \tag{3.2.2}$$

The fundamental solution for the modified Helmholtz equation is

$$\phi(r) = \frac{e^{-kr}}{4\pi r}.$$

The particular solution ψ satisfies

$$\Delta\psi - k^2\psi = \varphi,$$

where φ is the radial basis function.

If we choose the radial basis function as the general spline basis function of order n, we obtain

$$\varphi_n(r) = \begin{cases} r^{2n}\log r & \text{in } \mathbb{R}^2 \\ r^{2n-1} & \text{in } \mathbb{R}^3. \end{cases} \tag{3.2.3}$$

According to [30], the corresponding three-dimensional particular solution can be written as

$$\psi(r) = A\frac{\sinh(kr)}{r} + B\frac{\cosh(kr)}{r} + \sum_{i=-1}^{n} C_k r^i.$$

To guarantee the smoothness at $r = 0$, we choose coefficient A, B, C to have

$$\psi_{n+1}(r) = \frac{(2n)!\cosh(kr)}{rk^{2n+2}} - \sum_{k=0}^{n} \frac{(2n)!}{(2k)!}\frac{r^{2k-1}}{k^{2n-2k+2}} \tag{3.2.4}$$

For instance,

$$\psi_1(r) = \frac{\cosh(kr)}{rk^2} - \frac{1}{rk^2} \quad , \quad \psi_2(r) = \frac{2(\cosh(kr)-1)}{rk^4} - \frac{r}{k^2}.$$

Remark 3.2.1.

- The particular solution of the general spline basis function for modified Helmholtz equation in 2D can be given as

$$-\frac{1}{k^2}\sum_{i=1}^{n}\left(\frac{\Delta}{k^2}\right)^i r^{2n}\log(r) - \frac{(2n)!!}{k^{2n+2}}K_0(kr).$$

- For the Helmholtz equation $(\Delta + k^2)\psi = \phi$, the particular solution of the general two-dimensional spline basis function is

$$-\frac{1}{k^2}\sum_{i=1}^{n}\left(\frac{\Delta}{k^2}\right)^i r^{2n}\log(r) - \frac{(-1)^n(2n)!!}{k^{2n+2}}K_0(kr),$$

while in three space dimensions, we have

$$-\frac{1}{k^2}\sum_{i=1}^{n}\left(\frac{\Delta}{k^2}\right)^i r^{2n-1} - \frac{2(-1)^n(2n)!!}{k^{2n+2}}\frac{e^{-kr}}{r}.$$

3.3 An alternative method for Helmholtz equation

We can consider the Helmholtz equation as the implicit Poisson equation

$$\begin{cases} \Delta u = k^2 u + f \quad \text{in} \ \ \Omega \\ u = g \quad \text{on} \ \ \partial\Omega. \end{cases} \tag{3.3.1}$$

Using the radial basis function and the fundamental solution for the Poisson equation, we represent the solution as

$$u(x) = \sum_{i=1}^{n} a_i \psi(\|x - c_i\|) + \sum_{j=1}^{m} b_j \phi(\|x - d_j\|), \tag{3.3.2}$$

where n, m are the number of inner and outer points we generate. The fundamental solutions and particular solution we use is the same in the case of the Poisson equation. These were rather discussed for the Helmholtz equation in the Chapter 2. Taking the Laplace operator and using the definition of particular solution and fundamental solution, we get

$$\Delta u(x) = \sum_{i=1}^{n} a_i \varphi(\|x - c_i\|). \tag{3.3.3}$$

Substituting into the equation (3.3.1) , for all $x \in \Omega$ we have

$$\sum_{i=1}^{n} a_i \left(\varphi(\|x - c_i\|) - k^2 \psi(\|x - c_i\|)\right) + \sum_{j=1}^{m} b_j \left(-k^2 \phi(\|x - d_j\|)\right) = f(x). \tag{3.3.4}$$

If we choose $x = c_k$ $(k = 1, \cdots, n)$, we obtain system of n equations

$$\sum_{i=1}^{n} a_i \left(\varphi(\|c_k - c_i\|) - k^2 \psi(\|c_k - c_i\|)\right) + \sum_{j=1}^{m} b_j \left(-k^2 \phi(\|c_k - d_j\|)\right) = f(c_k) \tag{3.3.5}$$

We obtain also p equations using (3.3.2) if we take the value on p boundary points e_l $(l = 1, \cdots, p)$

$$\sum_{i=1}^{n} a_i \psi(\|e_l - c_i\|) + \sum_{j=1}^{m} b_j \phi(\|e_l - d_j\|) = g(e_l). \tag{3.3.6}$$

We inject both equation 3.3.5 and 3.3.6 into the linear neural network to approximate the coefficients a_i and b_j. In details, the training data includes $n + p$ training input-

output.

- Input: A vector of size $n+m$: $[(\varphi(\|c_k - c_i\|) - k^2\psi(\|c_k - c_i\|), -k^2\phi(\|c_k - d_j\|)]$, Output: $f(c_k)$
- Input: A vector of size $n+m$: $[\psi(\|e_l - c_i\|), \phi(\|e_l - d_j\|)]$, Output: $g(e_l)$

To compute the numerical solution, we inject the input $[\psi(\|x - c_i\|), \phi(\|x - d_j\|)]$ into the trained neural network.

Remark 3.3.1. In practice, we are using the compactly supported RBFs in both 2D and 3D

$$\phi_\lambda(r) = \begin{cases} \left(1 - \frac{r}{\lambda}\right)^2 & \text{if } r \leq \lambda \\ 0 \quad \text{if } r > \lambda \end{cases}$$

The corresponding particular solution are

$$\psi_\lambda(r) = \begin{cases} \frac{r^4}{16\lambda^2} - \frac{2r^3}{9\lambda} + \frac{r^2}{4} & \text{if } r \leq \lambda \\ \frac{13\lambda^2}{144} + \frac{\lambda^2}{12}\log\left(\frac{r}{\lambda}\right) & \text{if } r > \lambda \end{cases} \quad \text{in 2D}$$

$$\psi_\lambda(r) = \begin{cases} \frac{r^4}{20\lambda^2} - \frac{r^3}{6\lambda} + \frac{r^2}{6} & \text{if } r \leq \lambda \\ \frac{\lambda^2}{12} + \frac{\lambda^3}{30r} & \text{if } r > \lambda. \end{cases} \quad \text{in 3D}$$

3.4 The method of particular solutions for the wave equation

We wil also use the method of particular solutions for the following wave equation:

$$\begin{cases} \Delta u = \frac{1}{c^2}\frac{\partial^2 u}{\partial t^2} & \text{in } \Omega \\ u(x,y,z,t) = g(t) & \text{on } \partial\Omega \\ u\Big|_{t=0} = I_1(x,y,z), \quad \frac{\partial u}{\partial t}\Big|_{t=0} = I_2(x,y,z) & \text{in } \Omega. \end{cases}$$

For the numerical solution, we discretize the time-interval into uniform time-steps of length δt and use the notation

$$u(x,y,z,n\delta t) = u^n.$$

We apply the so-called Houbolt method to approximate second order derivative with respect to time:

$$\frac{\partial^2 u^n}{\partial t^2} \approx \frac{1}{\delta t^2}\left(2u^n - 5u^{n-1} + 6u^{n-2} - u^{n-3}\right). \tag{3.4.1}$$

Hence, at each time level, we are solving the modified Helmholtz equation

$$\begin{cases} \Delta u^n - \dfrac{2}{c^2\delta t^2}u^n = \dfrac{1}{c^2\delta t^2}\left(-5u^{n-1} + 6u^{n-2} - u^{n-3}\right) & \text{in } \Omega \\ u^n = g(n\delta t) & \text{on } \partial\Omega. \end{cases}$$

Note that to solve the modified Helmholtz equation, we only have to take the value of f at the collocations points and the value of g at the boundary points.
Since it is a three-step method, we use compute first three numerical solution by first-order explicit Euler method to initiate the computation with (3.4.1)

$$\begin{aligned} u^0 &= u\Big|_{t=0} \\ u^1 &= u\Big|_{t=0} - \delta t\frac{\partial u}{\partial t}\Big|_{t=0} \\ u^2 &= u\Big|_{t=0} - 2\delta t\frac{\partial u}{\partial t}\Big|_{t=0}. \end{aligned}$$

The following algorithm summarizes the method we use t o solve the wave equation

Algorithm 1 MFS-MPS algorithm to solve the wave equation

Subprogram `Helmholtz-solver()`

Input: k RHS function vector $[f(c_1), f(c_2), \cdots, f(c_n)]$ Boundary function vector $[g(e_1), g(e_2), \cdots, g(e_p)]$ **return** numerical solution

Procedure `Main-loop()`

Initialization $u(0), u(1), u(2)$ **for** *i= 3 to* $T/\delta t$ **do**

  $k = \frac{\sqrt{2}}{c\delta t}$, RHS-vector $= \frac{1}{c^2\delta t^2}\left(-5u(i-1) + 6u(i-2) - u(i-3)\right)$,

  boundary-vector $= [g(e_l, i\delta t)]_{l=1,\cdots,p}$ $u(i) =$ `Helmholtz-solver(`*k,RHS-vector, boundary-vector*`)`

end

return $u(T/\delta t)$

3.5 Numerical experiments

Example 3.5.1. In an attempt to test the algorithm, we consider the following Poisson equation in the domain of a unit 3-dimensional cube

$$\begin{cases} \Delta u = -2 \quad \text{in} \quad \Omega \\ u(x,y,z) = -\frac{1}{3}(x^2+y^2+z^2) \quad \text{on} \quad \partial\Omega. \end{cases} \tag{3.5.1}$$

We use 150 boundary points and 1350 outer points. For the radial basis function we chose the multiquadric function given with

$$\varphi(r) = \sqrt{r^2+c^2}. \tag{3.5.2}$$

The particular solution corresponding to the above multiquadric function is

$$\psi(r) = \begin{cases} \frac{c^3}{3} & \text{if} \quad r = 0 \\ \left(\frac{5c^2}{24} + \frac{r^2}{12}\right)\sqrt{r^2+c^2} + \frac{c^4\left[\ln(r+\sqrt{r^2+c^2})-\ln c\right]}{8r} & \text{if} \quad r \neq 0. \end{cases} \tag{3.5.3}$$

Table 3.2 shows the absolute error at some points :$(0,0,0); (0.5,0.5,0.5); (0.1,0.2,0.3); (0.15,0.95,0.5); (0.73,0.86,0.97)$ with different choice of parameter c in mulitiquadrarics function. According the results, $c = 3$ is the optimal choice for this problem.

x	0	0.5	0.1	0.15	0.73
y	0	0.5	0.2	0.95	0.86
z	0	0.5	0.3	0.5	0.97
$u(x,y,z)$	0	-0.25	-0.0466	-0.3916	-0.7378
c=1	1.62e-4	7.62e-5	7.33e-4	2.12e-3	5.34e-4
c=3	1.16e-4	7.63e-5	4.28e-5	1.23e-3	3.52e-3
c=5	9.13e-4	2.86e-4	4.9e-3	5.84e-3	9.79e-3
c=7	2.96e-3	2.29e-4	1.4e-3	1.08e-3	9.45e-3
c=9	1e-1	6.2e-4	5.26e-2	2.41e-2	8.61e-2

Table 3.2: Absolute error in approximation to $u(x,y,z)$ of Poisson equation.

Example 3.5.2. We consider the Helmholtz equation on a unit 3D cube

$$\begin{cases} \Delta u - k^2 u = (1-k^2)(e^x+e^y+e^z) \quad \text{in} \quad \Omega \\ u(x,y,z) = e^x+e^y+e^z \quad \text{on} \quad \partial\Omega. \end{cases} \tag{3.5.4}$$

In this example, we set $k = \sqrt{3}$ and use 1000 collocation points, 1350 outer points and 150 boundary points. The radial basis functions are general spline basis function(3.2.3) and the corresponding particular solution are the finite sum of 3.2.4. Table 3.3 displays the error estimation in maximum norm of some specific points with the choices for particular solutions. If the finite sum with bigger n for particular solution, the more accuracy we obatain for numerical solution.

x	0	0.5	0.3	0.15	0.25
y	0	0.5	0.4	0.95	0.85
z	0	0.5	0.5	0.5	0.94
$u(x, y, z)$	3	4.946163	4.490404	5.396265	6.183653
$n = 2$	7.99e-7	2.582e-1	2.45e-1	3.74e-2	4.047 e-2
$n = 3$	1.34e-6	8.518e-2	9.848e-2	1.14e-2	3.431e-3

Table 3.3: Absolute error in approximation to $u(x, y, z)$ for Helmholtz equation.

Example 3.5.3. We are using the same example as Example 3.5.2 and the radial basic functions are multiquadrics function with different value of c.

x	0	0.5	0.3	0.15	0.25
y	0	0.5	0.4	0.95	0.85
z	0	0.5	0.5	0.5	0.94
$u(x, y, z)$	3	4.946163	4.490404	5.396265	6.183653
$c = 1$	4.882e-2	3.315e-2	9.002e-2	6.384e-2	9.963e-3
$c = 3$	0.6045	0.3071	0.5323	0.0523	0.3614

Table 3.4: Absolute error in approximation to $u(x, y, z)$ for Helmholtz equation

Chapter 4

Error analysis

This chapter is an extended version of [1] where I contribute the implementation part.

4.1 Recent results

Katsurada and Okamoto ([31]), and Fairweather and Karageorghis ([32]) derived an error bound for the MFS type numerical solution of the Dirichlet - Laplace equation (1.1.1) on the two-dimensional disk. They have assumed that the boundary data g is analytic, solution is analytic and furthermore that u can be extended (continued) to be harmonic. Under these condtions, they have established the following error estimation:

$$\|u - u_M\|_{L^\infty(\Omega)} \leq C \left(\frac{\rho}{R}\right)^M, \tag{4.1.1}$$

where R is the radius of a circle including the outer points. This shows that the MFS is exponentially convergent with respect to increasing M, or R.

Betcke ([11]) obtained the same results for Helmholtz operator $\Delta + k^2 I$ and Balakrishnan and Ramachandran ([33]); Barnett and Betcke ([11]); Bogomolny ([34]) and for the modified Helmholtz operator $\Delta - k^2 I$.

The estimate 4.1.1 was proved by Katsurada ([35], [8]) for domains with an an analytical Jordan curve as a boundary.

In case u is not analytically continuable to the whole plane, but rather only up to

an extension $B(0; r_0)$ Kitagawa ([36], [37]) proved:

$$\|u - u_M\|_{L^\infty(\Omega)} \leq \|u\|_{L^\infty(\partial B(0,r_0))}$$
$$\left(\frac{2}{1-\frac{\rho}{R}}\right)\left[(1+A(R,p))\left(\frac{\rho}{r_0}\right)^{M/3} + 4\left(\frac{\rho}{R}\right)^{M/3}\right]$$

where $A(R, \rho)$ is some constant between 1 and 2. The price to pay for this excellent exponential convergence is that the condition number of the coefficient matrix of the resulting linear system of equations grows exponentially with respect to M.

4.2 Our results

For our approach, we recall the single layer representation. According to this, the solution $u(x)$ of (1.1.1) can be given by

$$u(x) = \int_{\partial\Omega} \phi(x-y)\mathcal{G}(y)\,\mathrm{d}y. \tag{4.2.1}$$

We extend the boundary to avoid the singularity of the integral. We denote $\Omega' \supset \overline{\Omega}$ be a extended domain containing pseudo boundary domain of Ω and we rewrite 4.2.1 formula as

$$u(x) = \int_{\partial\Omega'} \phi(x-y)\mathcal{G}(y)\,\mathrm{d}y. \tag{4.2.2}$$

In practice, we can put the following assumptions on the domain Ω'

- The domain Ω' is a closed triangular Lipschitz surface with triangles $\{T_n\}_{n=1}^N$
- Let $\{y_n\}_{n=1}^N$ are the centroids of the corresponding triangles $\{T_n\}_{n=1}^N$. Then we assume
$$\sup_{x\in\partial T_j} |x - y_j| \leq h \;\forall\; j = 1, 2, \cdots, N$$
- There exist a positive number d such that
$$\inf_{x\in\Omega, y\in\Omega'} |x - y| \geq d$$

Fixing the points y_j of the triangles, we define the numerical solution with parameter h as follows:

$$u_h(x) = \sum_{j=1}^{N} \int_{T_j} \phi(x-y_j)\mathcal{G}(y)\,\mathrm{d}y = \sum_{j=1}^{N} \int_{T_j} \mathcal{G}(y)\,\mathrm{d}y\phi(x-y_j). \tag{4.2.3}$$

In this way, the numerical solution can be represented by a linear combination of fundamental solutions, where the linear coefficients are $a_j = \int_{T_j} \mathcal{G}(y)\,\mathrm{d}y$.

Theorem 4.2.1. *Under the conditions 4.2, the MFS converges with second order in maximum norm*

$$|u(x) - u_h(x)| \leq Ch^2$$

where $C > 0$ is independent of the step size h.

Proof. Comparing (4.2.2) and (4.2.3), we have

$$\begin{aligned} |u(x) - u_h(x)| &= \left| \sum_{j=1}^{N} \int_{T_j} \phi(x-y) - \phi(x-y_j)\mathcal{G}(y)\,\mathrm{d}y \right| \\ &\leq \sum_{j=1}^{N} \int_{T_j} |\phi(x-y) - \phi(x-y_j)||\mathcal{G}(y)|\,\mathrm{d}y. \end{aligned}$$

Using the Cauchy–Schwarz inequality, we obtain

$$|u(x) - u_h(x)| \leq \sum_{j=1}^{N} \sqrt{\int_{T_j} |\phi(x-y) - \phi(x-y_j)|^2\,\mathrm{d}y \int_{T_j} |\mathcal{G}(y)|^2\,\mathrm{d}y}.$$

Using the formula for the fundamental solution of three-dimensional Laplace operator

$$\begin{aligned} \int_{T_j} |\phi(x-y) - \phi(x-y_j)|^2\,\mathrm{d}y &= \int_{T_j} \left| \frac{1}{|x-y|} - \frac{1}{|x-y_j|} \right|^2 \mathrm{d}y \\ &= \int_{T_j} \left| \frac{|x-y| - |x-y_j|}{|x-y||x-y_j|} \right|^2 \mathrm{d}y \\ &\leq \int_{T_j} \left| \frac{|y-y_j|}{|x-y||x-y_j|} \right|^2 \mathrm{d}y \qquad \text{[triangular inequality]} \\ &\leq \int_{T_j} \frac{|y-y_j|^2}{d^4}\,\mathrm{d}y \qquad \text{[minimum distance assumptions]} \\ &\leq \int_{B_j} \frac{|y-y_j|^2}{d^4}\,\mathrm{d}y. \end{aligned}$$

In the last inequality, B_j denotes the open disk centered at y_j with radius h and lies in the same plane with T_j.

We can calculate the integral

$$\int_{B_j} \frac{|y-y_j|^2}{d^4}\,\mathrm{d}y = \frac{\pi}{2d^4}h^4.$$

To sum up, we have

$$|u(x) - u_h(x)| \leq \sum_{j=1}^{N} \sqrt{\frac{\pi}{2d^4} h^4 \int_{T_j} |\mathcal{G}(y)|^2 \, \mathrm{d}y} = \frac{\sqrt{\pi}}{\sqrt{2}d^2} \|\mathcal{G}\|_{L_2(\partial\Omega_1)} h^2.$$

Here $C = \frac{\sqrt{\pi}}{\sqrt{2}d^2} \|\mathcal{G}\|_{L_2(\partial\Omega)}$ depends on Ω and triangulation and independent of the parameter h. □

Theorem 4.2.2. *Under the conditions 4.2, the MFS converges with second order in the H_1-norm, i.e., we have*

$$|u(x) - u_h(x)|_{H_1(\Omega)} \leq Ch^2,$$

where $C > 0$ is independent of the step size h.

Proof.

$$|u(x) - u_h(x)|_{H_1(\Omega)} \leq \sqrt{\int_{\Omega} \sup_{x\in\Omega} |u - u_h|^2 + \sup_{x\in\Omega} |\nabla(u - u_h)|^2}.$$

Using Theorem 4.2.1, we have

$$\sup_{x\in\Omega} |u - u_h|^2 \leq C_1 h^4.$$

Therefore, we have to prove

$$\sup_{x\in\Omega} |\nabla(u - u_h)|^2 \leq C_2 h^4.$$

Indeed, we can prove all partial derivatives have the same property

$$\sup_{x\in\Omega} |\partial_k(u - u_h)|^2 \leq C_k h^4$$

for all $k = 1, 2, 3$.

Similar to the estimation of $|u - u_h|$, we have:

$$|\partial_k(u(x) - u_h(x))|^2 \leq \sum_{j=1}^{N} \sqrt{\int_{T_j} |\partial_k(\phi(x - y) - \phi(x - y_j))|^2 \, \mathrm{d}y \int_{T_j} |\mathcal{G}(y)|^2 \, \mathrm{d}y}.$$

The first integral term can be estimated as

$$
\begin{aligned}
&|\partial_k(\phi(x-y)-\phi(x-y_j))| = \left|\frac{2(x_k-y_k)}{|x-y|^3} - \frac{2(x_k-(y_j)_k)}{|x-y_j|^3}\right| \\
&= 2\left|\frac{(x_k-y_k)|x-y_j|^3-(x_k-(y_j)_k)|x-y|^3}{|x-y|^3|x-y_j|^3}\right| \\
&\le 2\left|\frac{(x_k-y_k)|x-y_j|^3-(x_k-y_k)|x-y|^3}{|x-y|^3|x-y_j|^3}\right| + 2\left|\frac{(x_k-y_k)|x-y|^3-(x_k-(y_j)_k)|x-y|^3}{|x-y|^3|x-y_j|^3}\right| \\
&= 2\left|\frac{(x_k-y_k)(|x-y_j|^3-|x-y|^3)}{|x-y|^3|x-y_j|^3}\right| + 2\left|\frac{(x_k-y_k)-(x_k-(y_j)_k)}{|x-y_j|^3}\right| \\
&\le 2\frac{|x-y|(|x-y_j|-|x-y|)(|x-y_j|^2+|x-y_j||x-y||x-y|^2)}{|x-y|^3|x-y_j|^3} + 2\frac{|y-y_j|}{|x-y_j|^3} \\
&\le 2|y-y_j|\left(\frac{1}{|x-y|^2|x-y_j|}+\frac{1}{|x-y||x-y_j|^2}+\frac{1}{|x-y_j|^3}+\frac{1}{|x-y_j|^3}\right) \\
&\le \frac{8|y-y_j|}{d^3}
\end{aligned}
$$

Inserting this to the previous expression, we have

$$
|\partial_k(u(x)-u_h(x))|^2 \le \sum_{j=1}^{N}\sqrt{\frac{32\pi}{d^6}h^4\int_{T_j}|\mathcal{G}(y)|^2\,\mathrm{d}y} = \frac{4\sqrt{2}\pi}{d^3}\|\mathcal{G}\|_{L_2(\partial\Omega_1)}h^2,
$$

as stated in the theorem. □

4.3 Numerical experiments

The method was tested on the domain $(-1,1)^3$ $B_1(1)$ using the analytic solution $u(x,y,z) = e^{x^2-y^2}sin(2xy)z$. The outer points $\{y_j\}_{j=1}^N$ were placed on the enlargement of $\partial\Omega$ centered at origin with scale factor $1+\alpha = 1.15$and $1+\alpha = 1.2$, respectively. A surface grid was applied here to distribute them quasi-uniformly. For a visualization, see Figure 4.1

In the first series of experiments, we took $N = M$ such that a complete linear system was solved numerically. In the vicinity of the boundary, the computations are unreliable due to the large values of the functions ϕ_j here. Therefore, in the test, we have computed the norms $\sup|u(x)-u_h(x)|$ and $\sup|\nabla u(x)-\nabla u_h(x)|$ sampled on a quasi uniform grid $\partial\Omega'$,in, which consists of 6970 points in the domain $(-0.5,0.5)^3 \setminus B_{0.5}(0.5)$. The corresponding results are shown in Table 4.1. Whenever the functions $\{\phi_j\}_{j=1}^N$ have the approximation property given in Theorem 4.2.1, the determination of the optimal coefficients $\{aj\}_{j=1}^N$ would require in general, a linear

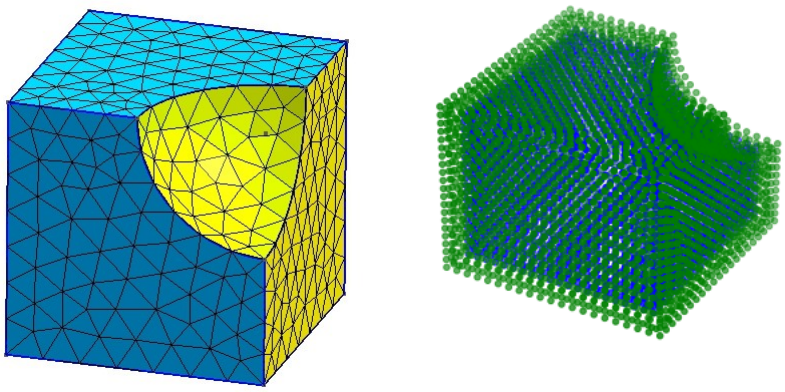


Figure 4.1: The computational domain and outer points with $\alpha = 0.15$.

Table 4.1: Error estimation when $N = M$ in the maximum norm with $\alpha = 0.15$ (up) and $\alpha = 0.2$ (down).

$N = M$	h	$\sup \lvert u(x) - u_h(x) \rvert$	$\sup \lvert \nabla u(x) - \nabla u_h(x) \rvert$
206	0.4	1.10e-2	6.63e-2
473	$0.4/\sqrt{2}$	2.21e-3	1.84e-2
735	0.2	1.15e-3	9.74e-3
1620	$0.2/\sqrt{2}$	1.28e-4	9.19e-4
2819	0.1	7.46e-5	3.83e-4
5899	$0.1/\sqrt{2}$	2.83e-6	1.82e-5
11052	0.05	2.03e-7	1.11e-6

$N = M$	h	$\sup \lvert u(x) - u_h(x) \rvert$	$\sup \lvert \nabla u(x) - \nabla u_h(x) \rvert$
206	0.4	5.56e-3	4.82e-2
473	$0.4/\sqrt{2}$	1.40e-3	9.59 e-3
735	0.2	3.72e-4	1.28e-3
1620	$0.2/\sqrt{2}$	7.58e-5	3.97 e-4
2819	0.1	5.71e-5	2.28e-4
5899	$0.1/\sqrt{2}$	2.91e-7	1.46e-6
11052	0.05	8.43e-9	4.48e-8

system of size $N \times N$. To save computational time and avoiding badly conditioned problems, it was advised to take less boundary points. The underdetermined linear system was solved then in the least square-sense. The corresponding results are shown in Table 4.1.

For computing the L_1, L_2, H_1- norm error we use 5 points Gaussian quadrature approximation for each tetrahedron. For example, to measure the computational error in the L_1-norm, we have

$$|u(x) - u_h(x)|_{L1} = \sum_{j=1}^{N} Vol(T_j) \cdot \int_{T_j} |u(x) - u_h(x)| dx$$

$$\approx Vol(T_j) \cdot \sum_{j=1}^{N} \sum_{i=1}^{5} s_i |u((w_j)_i) - u_h((w_j)_i)|,$$

Table 4.2: Error estimation with different (N, M) in different norms($\alpha = 0.2$)

(M, N)	Maximum norm	$L1$-norm	$L2$-norm	$H1$-norm	Running time
(140,384)	1.01e-2	1.46e-3	1.65e-3	6.05e-3	0.36s
(140,735)	4.47e-3	3.98e-4	6.28e-4	3.20e-3	0.5s
(140,2819)	1.72e-3	1.31e-4	2.07e-4	1.45e-3	0.61s
(140,11052)	1.43e-3	1.25e-4	1.92e-4	1.34e-3	1.52s
(384,735)	2.08e-3	3.29e-4	4.03e-4	1.48e-3	0.5s
(384,2819)	3.23e-4	2.02e-5	2.67e-5	1.65e-4	0.78s
(384,11052)	1.91e-4	3.28e-5	4.01e-5	1.62e-4	1.3s
(735,2819)	4.92e-5	5.43e-6	7.39e-6	4.29e-5	1.01s
(735,11052)	6.26e-5	1.01e-5	1.28e-5	3.98e-5	2.92s
(2819,11052)	1.37e-6	6.55e-8	1.17e-7	6.15e-7	22.7s

where

$$
\begin{aligned}
w_1 &= (1/4, 1/4, 1/4, 1/4) \\
w_2 &= (1/2, 1/6, 1/6, 1/6) \\
w_3 &= (1/6, 1/2, 1/6, 1/6) \\
w_4 &= (1/6, 1/6, 1/2, 1/6) \\
w_5 &= (1/6, 1/6, 1/6, 1/2)
\end{aligned}
$$

and we have used the weights

$$(s_1, s_2, s_3, s_4) = (-4/5, 9/20, 9/20, 9/20, 9/20).$$

In both cases, on the grid, we have experienced an even higher convergence rate as suggested in Theorem 4.2.1. Also, Table 4.2 confirms the earlier observation that choosing $M > N$ is beneficial to reduce the computing time for a certain level of accuracy. In all cases, the computations were carried out using an Intel 5i processor with 4 GB memory.

Chapter 5

Adaptive finite difference methods using artificial neural networks

This chapter continues the work in [22]. The term "adaptive" methods here is comprehended as adjusting the choice of free parameters (not as adaptive in time as usual). The application of these adaptive finite difference schemes is to build up an optimal model publicity while the training process is conducted at the center(offline training).

5.1 Motivations

The motivation to improve the finite difference methods comes from the challenge when solving a stiff problem for ODE's. It requires extremely small step sizes: unless, the scheme becomes unstable. For example, let us consider the Dashquist test problem with $\lambda = -25$:

$$u'(t) = -25u(t) \quad t > 0, \qquad u(0) \text{ is given.} \tag{5.1.1}$$

The analytical solution of it is $u(t) = e^{-25t} \to 0$ when $t \to \infty$.
We can use a simple numerical method, e.g., the explicit Euler method with a large step size $\Delta t = 1/10$ to estimate the solution at final time $T = 1$. In Figure 5.2, we can see that the numerical solution is oscillatory because it does not satisfy the stability condition $\lambda \Delta t < 1$. Also, if we change the step size to be $\Delta t < 1/25$ then the scheme becomes stable.

Questions: Is there any method which can solve the problem with the same step

size using a simple formula like the forward Euler method?
In general, if there is a given numerical scheme to solve a specific problem, can we enhance the precision of the method with the same step size?

5.2 Description of a neural network-based method for solving ODEs

The aim of the method is to enhance the accuracy of some known schemes by using the neural networks to minimize the loss without changing the step size. An appropriate neural network architecture can be defined as the set of neural networks, where each sub-neural net simulates one-step in the loop of the numerical scheme. The output of such a numerical solution at a specific time corresponds to the output of such a sub-neural net and this also serves as the input of the next sub-neural net. For a schematic picture, see Figure 5.1).
We consider a general initial-value problem for an ODE

$$U'(t) = f(U(t), t), \quad t \in [0, T]; \qquad U(0) \text{ is given.}$$

For the sake of simplicity, we discretize the time into N uniform intervals corresponding to N sub neural networks. On each sub-network j, we use a set of weight parameters w_{j-1}.
$U_{n,NN}$ denotes the output of the nth sub-network in the consecutive approximation to $U(0)$. This can be recognized as the numerical solution at time $t = n\Delta t$ where $\Delta t = T/N$. In this way, formally, for all $j = 0, \cdots, N-1$

$$U_{j+1,NN} = NNO(w_j, U_{j,NN}).$$

Pre-trained weights can be adapted from an existing numerical scheme and coincide on every sub-network $w_0 \equiv \cdots \equiv w_{N-1}$.
If the baseline approximation we use is sufficiently accurate, the initial weights are close to the optimal points that can save time for training.
Training data set can be generated mainly by two ways

- Using the exact or analytical data. It is not applicable for all problems, but it can be utilized for testing the model.

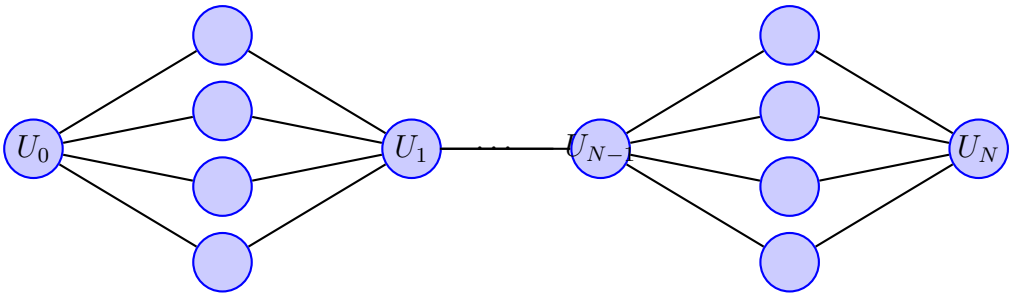


Figure 5.1: Neural network architecture for ODE

- Using high-resolution data. We may use the same method with smaller step size or higher order method.

The number of training data equals to the number of generated initial values $U(0)$. For the specific problem, the model for generating initial value is different.
To train the networks, we propose two types of training: all-in-one and one-by-one. In all-in-one training, we compute the loss function on the whole network and train once. The loss function usually define as

$$L(w_0, \cdots, w_{N-1}) = \sum_i \sum_n |U^i_{n,\mathrm{NN}} - U^i_{n,\mathrm{exact}}|^2.$$

We sometimes also use the modified loss function

$$L(w_1, \cdots, w_{N-1}) = \sum_i \sum_n \left| \frac{U^i_{n,NN} - U^i_{n,exact}}{U_{n,exact}} \right|^2.$$

This training was proposed in [22] and leads to a really fast computation. However, it has its disadvantage that the loss function is well-optimized only on the a few first weights and stuck at the local minimum on the last weights. It can be explained by the dependence of numerical solutions on the weights at different times. While w_{N-1} appears only in the loss of the last sub-network $|\sum_i U^i_{N,\mathrm{NN}}(t) - U^i_{N,\mathrm{exact}}(t)|^2$, the first weight w_1 appears in every loss components.

In the one-by-one approach, we move stepwise from the first sub-network the last one. It means that after training the first sub-network, we get the first optimal weights w_1 and the output U_1 which is the input for the second sub-network, and the loss for second sub-network only depends on the weights w_2. In general, we define

the loss and the time step as

$$\begin{aligned}\mathcal{L}_j(w_j) &= \sum_i |U^i_{j,NN} - U^i_{j,exact}|^2 \\ w_j &= \text{argmin}_j \mathcal{L}_j \\ U_{j+1} &= NNO(w_j, U_j).\end{aligned}$$

The optimization algorithm may varied including standard gradient descent, stochastic gradient descents, quasi Newton methods and Nelder–Mead Simplex algorithm.

The test data is generated in the same way as the training data set. Using this, we can evaluate the model by one of three metrics: Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE).

5.3 Numerical Experiments for ODEs

5.3.1 A motivating example

The explicit Euler step for the model problem (5.1.1) can be given as

$$u_{n+1} = u_n + \Delta t f(u_n) = (1 - 25\Delta t)u_n.$$

According to this formula, we can generalize geometric growth in the form

$$u_{n+1} = g_n u_n.$$

As initial value choose 50 random numbers from the interval $[0, 1]$. With all-by-one training, the loss function is

$$L(g_1, \cdots, g_{10}) = \sum_{i=1}^{10} \sum_{n=1}^{10} \left| \frac{u^i_{n,NN} - u^i_{n,exact}}{u^i_{n,exact}} \right|^2.$$

Note that we divide with $u^i_{n,exact}$ in each components, due to $u^i_{n,exact} \to 0$ when $t \to \infty$. By using the BFGS algorithm, the optimized coefficient can be seen in Table 5.1. Comparing to the one-by-one training, it converges to the global minimum is $g_0 = \cdots = g_9 = e^{-25\Delta t} \approx 0.08208$.

Table 5.1: The optimized coefficients obtained by all-in-one and one-by-one training.

	$g_0 = g_1 = g_2$	g_3	g_4	g_5	g_6	g_7	g_8	g_9
all-in-one	8.208e-02	8.211e-02	-1.635e-03	-4.033	-1.670e-03	-4.034	-1.664e-03	-4.0470
one-by-one	8.208e-02	8.208e-02	8.208e-02	8.208e-02	8.208e-02	8.208e-02	8.208e-02	8.208e-02

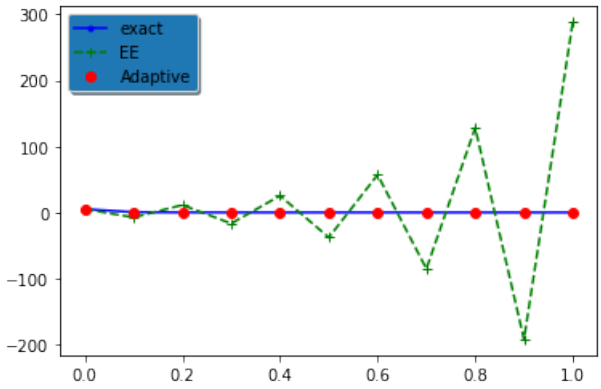


Figure 5.2: Analytical solution and numerical solutions solving by explicit Euler method and the adaptive method of the motivation example with the initial value $u_0 = 5$.

5.3.2 System of ODEs

Example 5.3.1. Let us consider the following initial values problem for a second order ODE:

$$\begin{cases} u''(t) + c^2 u(t) = 0 \quad t \in (0,1), \\ u(0) = u_0, \\ u'(0) = 0 \end{cases} .$$

The analytical solution for this problem is $u(t) = u_0 \cos(ct)$.

If we set $v(t) = -cv(t)$, we reduce the second order ODE to the system of ODEs

$$U(t)' = \begin{pmatrix} u \\ v \end{pmatrix}' = F\left(U\right) = \begin{pmatrix} -cv \\ cu \end{pmatrix}.$$

We start from the BDF2 method as a baseline method

$$\frac{3}{2}U_n - 2U_{n-1} + \frac{1}{2}U_{n-2} = \Delta t F(U_n).$$

The general form of this method - depending on the parameter g_n - is the following:

$$(1+g_n)U_n - (1+2g_n)U_{n-1} + g_n U_{n-2} = \Delta t F(U_n) \qquad n \geq 2. \tag{5.3.1}$$

In case of BDF2, we have $g_n = 0.5$ for every step. This parameter may effect the

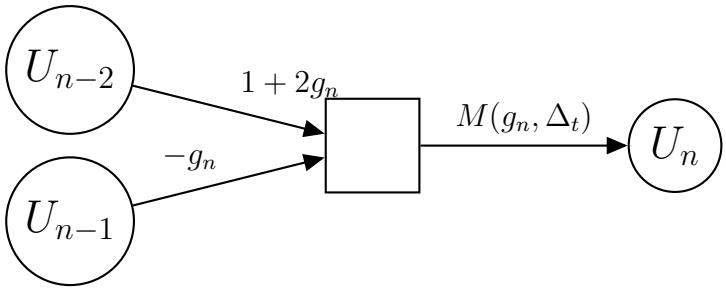


Figure 5.3: The neural network structure for one-step ODE.

performance in the stiff problems. The artificial neural network can modify the parameter g_i suitably for each time step. For simplicity, we only discretize the domain with 3 uniform time-intervals and step size $\Delta t = 1/3$.

In this example, we specify $F(U) = \begin{bmatrix} 0 & -c \\ c & 0 \end{bmatrix} U$ and rewrite (5.3.1) as

$$U_n = \frac{1}{(1+g_n)^2 + (c\Delta t)^2} \begin{bmatrix} 1+g_n & -c\Delta t \\ c\Delta t & 1+g_n \end{bmatrix} ((1+2g_n)U_{n-1} - g_n U_{n-2}).$$

The ANN structure is described in Figure 5.3, where $M(g_n, \Delta_t)$ is a linear operator defined as

$$M(g_n, \Delta_t) = \frac{1}{(1+g_n)^2 + (c\Delta t)^2} \begin{bmatrix} 1+g_n & -c\Delta t \\ c\Delta t & 1+g_n \end{bmatrix}$$

. In the implementation, we can simply define the one-step function, with inputs g_n, U_{n-1}, U_{n-2}

To generate appropriate training data, we utilize the initial data u_0^i, where $i = 1, \cdots, I$ are random points in the domain $[0, 1]$, with I the number of training data. We don't apply the same method on the finer grid; instead, we use the exact solution

$$U^i_{0,\text{exact}}(t) = (u_0^i \cos(ct), u_0^i \sin(ct)) \qquad i = 1, \cdots, I.$$

We define the loss function as

$$L(g_2, g_3) = \sum_{i=1}^{I} \sum_{n=2}^{3} |U^i_{n,\text{NN}} - U^i_{n,\text{exact}}|^2.$$

The gradient descent method for this example is a standard full-batch gradient descent.

The method can perform with more accuracy if the ANN structure have more free parameters. Namely, let a_n, b_n, d_n be the coefficients in the following expression to

approximate U_n' :

$$a_n U_n + b_n U_{n-1} + d_n U_{n-2} = \Delta t U_n' = \Delta t F(U_n) \qquad n \geq 2.$$

We can rewrite this as

$$U_n = \frac{1}{a_n^2 + (c\Delta t)^2} \begin{bmatrix} a_n & -c\Delta t \\ c\Delta t & a_n \end{bmatrix} (-b_n U_{n-1} - d_n U_{n-2}).$$

For training, we use $c = 50$, $I = 10$ and $U_1 = (u_0 \cos(\Delta t), u_0 \sin(\Delta t))$. The trained coefficients can be seen in Table 5.2.

Table 5.2: Optimized weights obtained by different techniques.

Techniques	Weights for U_2 approximation	Weights for U_3 approximation
all-in-one	$g_2 = -2.507$	$g_3 = -6.745$
one-by-one (1 parameter)	$g_2 = -2.356$	$g_3 = -15.586$
one-by-one (multiple parameters)	$(a_2, b_2, d_2) = (2.096, 2.397, -4.580)$	$(a_3, b_3, d_3) = (-2.757, 6.360, -15.534)$

We take randomly an initial value from the test set and plot the numerical solutions by using different techniques in Figure 5.4.

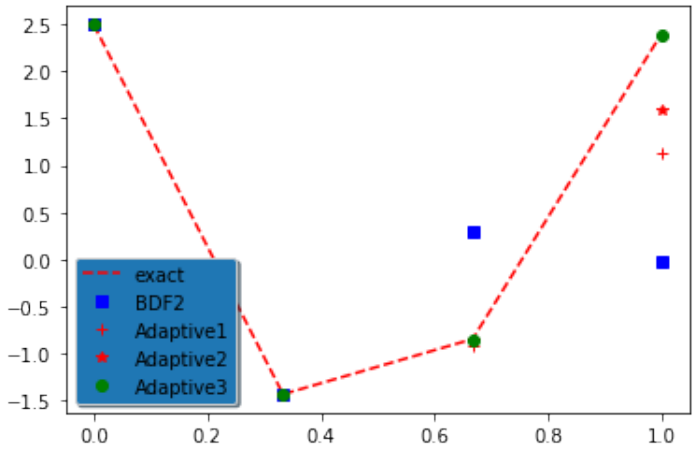


Figure 5.4: Comparison of numerical solutions to the analytical solution: (Adaptive1) all-in-one training, (Adaptive2) one-by-one training with one parameter, (Adaptive3) one-by-one training with more parameters

5.3.3 Case study for a non-linear ODE

Example 5.3.2. We will investigate the numerical solution of the initial value problem

$$\begin{aligned} \frac{dy}{dt} &= y^2 - y^3 \\ y(0) &= \delta \\ 0 < t &< \frac{2}{\delta}, \end{aligned}$$

which is a simple model of flame propagation ([38]).

In practice, if a match is set to light, the ball of flame grows rapidly until it reaches a critical size. Then it remains at that size because the amount of oxygen being consumed by the combustion in the interior of the ball balances the amount available through the surface.

The scalar variable $y(t)$ represents the radius of the ball. The y^2 and y^3 terms come from the surface area and the volume. The critical parameter is the initial radius, δ , which is "small". We seek the solution over a length of time that is inversely proportional to δ.

In this example, we use the step size $\Delta t = 5$ with 50 intervals. The baseline method is RK4 and we replace the fixed coefficients of RK4 by free parameters to obtain general explicit RK methods with 4 stages. We suppose

$$\begin{aligned} y_{n+1} &= y_n + h\left(b_1k_1 + b_2k_2 + b_3k_3 + b_4k_4\right), \\ t_{n+1} &= t_n + h \end{aligned}$$

for $n = 0, 1, 2, 3, ..., 39$, where

$$\begin{aligned} k_1 &= f(t_n, y_n), \\ k_2 &= f\left(t_n + \frac{h}{2}, y_n + ha_{21}k_1\right), \\ k_3 &= f\left(t_n + \frac{h}{2}, y_n + ha_{31}k_1 + ha_{32}k_2\right), \\ k_4 &= f\left(t_n + h, y_n + ha_{41}k_1 + ha_{42}k_2 + ha_{43}k_3\right). \end{aligned}$$

The set of weights for each sub-network includes 10 parameter $(b_1, b_2, b_3, b_4, a_{ij}|4 \geq j > i \geq 1)$. We try to find the optimized weights starting from the values $(1/6, 1/3, 1/3, 1/6, 1/2, 0, 1/2, 0, 0, 1)$, such that numerical solutions are close to training data which is obtained by decreasing the step size 100 times. The initial values are 100 random numbers from $[0, 0.01]$.

We have used the analytical solution

$$y(t) = \frac{1}{(W(ae^{a-t}) + 1)},$$

where $a = \frac{1}{\delta} - 1$ and W denotes the Lambert function, which is the solution to $W(z)e^{W(z)} = z$.

Figure 5.5 illustrates the analytical and numerical solutions solving RK4 and adaptive RK4 methods.

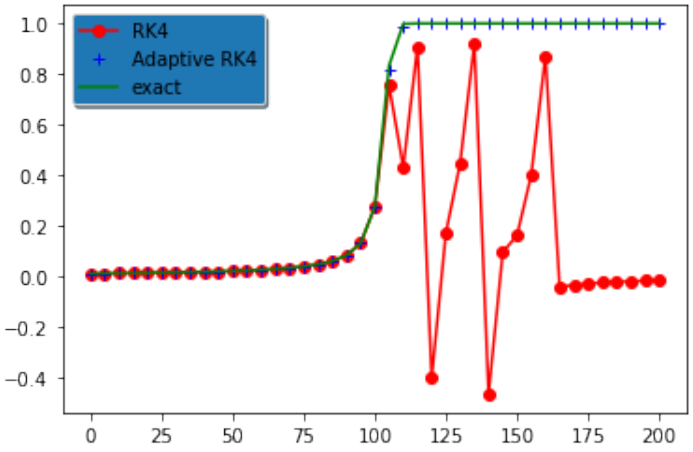


Figure 5.5: Numerical solutions and the analytical solution for flame propagation problem

5.4 Case study for the Burgers equation

Let us consider an initial value problem for the general conservation law

$$\begin{cases} \partial_t u(t,x) + (\partial_x f(u))(t,x) = 0 & (t,x) \in (0,T) \times \mathbb{R} \\ u(x,0) = u_0(x) \end{cases} \tag{5.4.1}$$

Taking $f(u) = \frac{u^2}{2}$ gives the Burgers equation, which is widely investigated in the numerical studies. In the general model, u is the conserved quantity and $f(u)$ denotes its flux.

5.4.1 Numerical schemes as a neural net

Based on the finite volume method, we are starting with a general conservative scheme

$$U^n_{j+1} = U^n_j - \frac{\Delta t}{\Delta x}(\hat{f}^n_{j+\frac{1}{2}} - \hat{f}^n_{j-\frac{1}{2}}),$$

where

- we consider a uniform tessellation of the spatial domain (here $\mathbb{R}$) into cells I_j,
- U_j^n is the cell-average of u in I_j, which serves as unknowns in the discretized system,
- $\hat{f}^n_{j+\frac{1}{2}}$ is the estimation of the flux on the right-hand boundary point of I_j.

The essence of the different approaches is an appropriate approximation of the numerical flux. We start from the so-called Lax–Friedrichs scheme, which is given by

$$\hat{f}^n_{j+\frac{1}{2}} = \frac{1}{2}\left(f(U_j^n) + f(U_{j+1}^n)\right) - \frac{1}{2}\max\{f'(U_j), f'(U_{j+1})\}(U_{j+1}^n - U_j^n)$$
$$\hat{f}^n_{j-\frac{1}{2}} = \frac{1}{2}\left(f(U_{j-1}^n) + f(U_j^n)\right) - \frac{1}{2}\max\{f'(U_{j-1}), f'(U_j)\}(U_j^n - U_{j-1}^n).$$

We can represent the Lax–Friedrichs scheme in the form of neural network approximately in Figure 5.6.

- The maximum can be represented as a sum of two ReLU functions

$$|a| = \mathrm{ReLU}(a) + \mathrm{ReLU}(-a)$$

- The maximum function can be rewrite as

$$\max(a, b) = a + \mathrm{ReLU}(b - a)$$

- The product operator and $x \to x^2$ mapping can be represented by $\epsilon-$ approximation neural nets. In practice, we can use artificial neural networks (the function itself) instead to make it less complicated.

5.4.2 Adaptive method

Weights determination. For each sub-network, we introduce new weights w_1, w_2, w_3, w_4, which assign possibly different weights to define new numerical flux functions as

$$\hat{f}^n_{j+\frac{1}{2}} = w_1\left(f(U_j^n) + f(U_{j+1}^n)\right) - w_2\max\{f'(U_j), f'(U_{j+1})\}(U_{j+1}^n - U_j^n)$$
$$\hat{f}^n_{j-\frac{1}{2}} = w_3\left(f(U_{j-1}^n) + f(U_j^n)\right) - w_4\max\{f'(U_{j-1}), f'(U_j)\}(U_j^n - U_{j-1}^n).$$

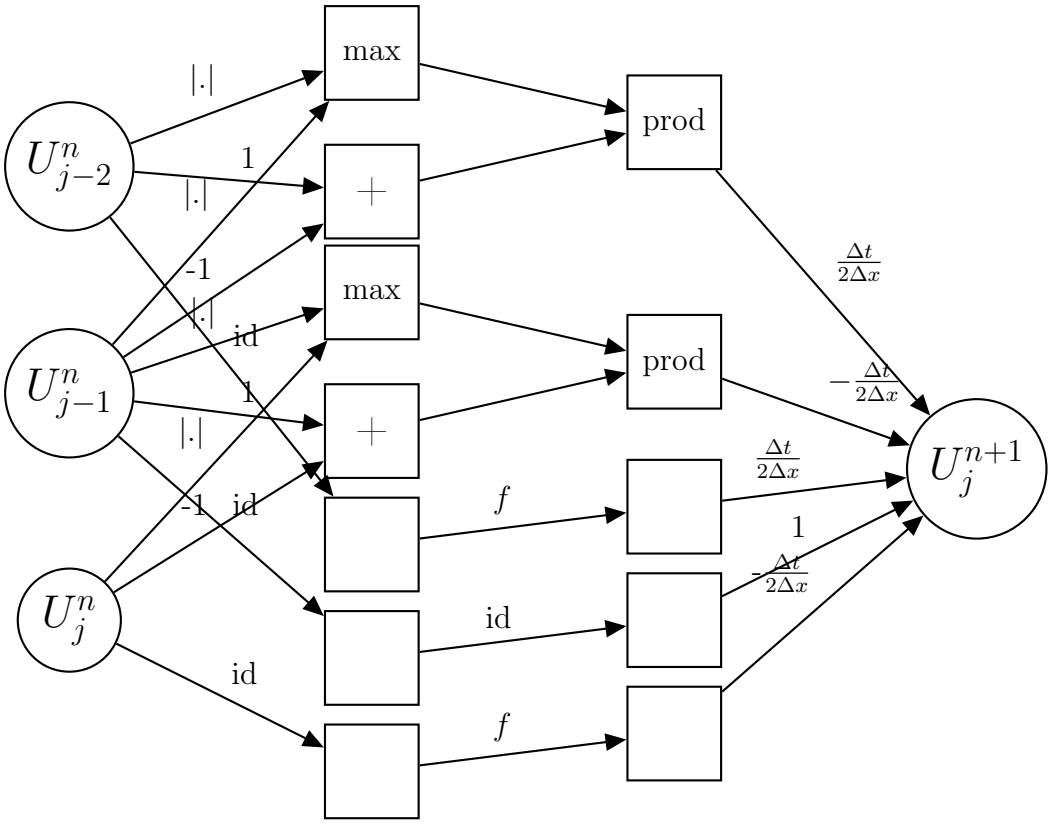


Figure 5.6: The neural network structure for one-step Lax-Friedrichs scheme

Then in each step, we apply

$$U^n_{j+1} = U^n_j - \frac{\Delta t}{\Delta x}\left[w_1\left(f(U^n_j) + f(U^n_{j+1})\right) - w_3\left(f(U^n_{j-1}) + f(U^n_j)\right)\right.$$
$$\left. -w_4 \max\{f'(U_{j-1}), f'(U_j)\}(U^n_j - U^n_{j-1}) - w_2 \max\{f'(U_j), f'(U_{j+1})\}(U^n_{j+1} - U^n_j)\right].$$

The initial training data We use the family of the functions below as initial functions in the training data

$$u_0(x) = \begin{cases} \frac{1}{2^\alpha}(x+1)^\alpha & \text{if } -1 \le x \le 1 \\ 1 & \text{if } 1 < x \le 2 \end{cases},$$

where α is chosen randomly 25 times in the interval $[0; 2]$. We use the same Lax–Friedrichs methods but in the finer grid: 10 times finer in both the time and the spatial coordinates.

The test example we are using is $u_0(x) = H(x)$, where $H(x)$ is the Heaviside function. The exact solution is

$$u(t,x) = \begin{cases} 1 & \text{if } t < x, \\ x & \text{if } 0 < x < t, \\ 0 & \text{if } x < 0. \end{cases}$$

However, we do not use it for comparison because we are not always can find an analytical solution for arbitrary initial solution. Figure 5.7 shows how close the trained numerical solution can become to the numerical solution on the the finer resolution.

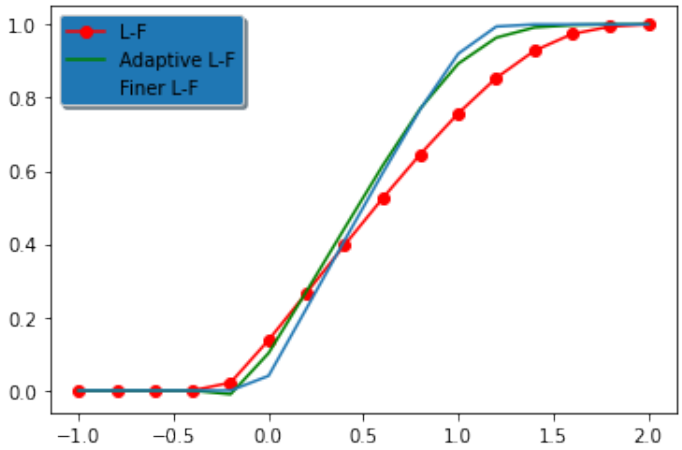


Figure 5.7: The numerical solutions solved in coarse grid, fine grid by L-F scheme and by neural network.

To make it more obvious, we calculate the ratio

$$R(x) = \frac{|u_{LF}(x) - u_{\text{finer}}(x)|}{|u_{\text{Adaptive LF}}(x) - u_{\text{finer}}(x)|} \tag{5.4.2}$$

of the pointwise errors. This shows how much the computational error can be reduced if the adaptive approach is applied. For the numerical results, see Figure 5.8.

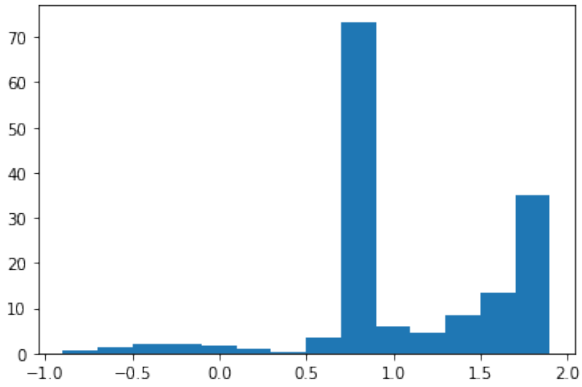


Figure 5.8: The ration of errors in (5.4.2).

Appendix A

Special functions

A.1 Bessel functions

Bessel functions([39]) are analytical solutions $y(x)$ of Bessel's differential equation

$$x^2\frac{d^2y}{dx^2} + x\frac{dy}{dx} + \left(x^2 - \alpha^2\right)y = 0.$$

Bessel functions of the first kind of order α, denoted as $J_\alpha(x)$, diverge as x tends to zero and has a form of series expansion:

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

where $\Gamma(z)$ is the gamma function.

The Bessel functions of the second kind of order α, denoted by $Y_\alpha(x)$, that have a singularity at the origin are also the solutions of the Bessel's equation.
For non-integer α, $Y_\alpha(x)$ is related to the first kind Bessel functions$J_\alpha(x)$ by the formula

$$Y_\alpha(x) = \frac{J_\alpha(x)\cos(\alpha\pi) - J_{-\alpha}(x)}{\sin(\alpha\pi)}.$$

In the case of integer order n, the function is defined by taking the limit as a non-integer α tends to n:

$$Y_n(x) = \lim_{\alpha\to n} Y_\alpha(x).$$

A.2 Modified Bessel functions

The first and second kind of modified Bessel function of order α(or occasionally the hyperbolic Bessel functions), denoted by $I_\alpha(x)$ and $K_\alpha(x)$ are the two linearly independent solutions to the modified Bessel's equation:

$$x^2\frac{d^2y}{dx^2} + x\frac{dy}{dx} - \left(x^2 + \alpha^2\right) y = 0.$$

It is an extension of the Bessel functions for complex arguments x, and an important special case is that of a purely imaginary argument. The modified Bessel functions of the first and second kind and are represented as follow:

$$I_\alpha(x) = i^{-\alpha} J_\alpha(ix) = \sum_{m=0}^{\infty} \frac{1}{m!\,\Gamma(m+\alpha+1)} \left(\frac{x}{2}\right)^{2m+\alpha},$$
$$K_\alpha(x) = \frac{\pi}{2}\frac{I_{-\alpha}(x) - I_\alpha(x)}{\sin\alpha\pi}.$$

A.3 Hankel functions

The Hankel functions of the first and second kind of order α , $H_\alpha^{(1)}(x)$ and $H_\alpha^{(2)}(x)$, respectively are two linearly independent solutions to Bessel's equation:

$$H_\alpha^{(1)}(x) = J_\alpha(x) + iY_\alpha(x),$$
$$H_\alpha^{(2)}(x) = J_\alpha(x) - iY_\alpha(x).$$

The following expression shows the relation between the Hankel functions and the modified Bessel functions:

$$K_\alpha = \begin{cases} \frac{\pi}{2} i^{\alpha+1} H_\alpha^{(1)}(ix) & -\pi < \arg x \leq \frac{\pi}{2} \\ \frac{\pi}{2}(-i)^{\alpha+1} H_\alpha^{(2)}(-ix) & -\frac{\pi}{2} < \arg x \leq \pi \end{cases}.$$

Bibliography

[1] Hieu T. Hoang, Ferenc Izsak, and Gabor Maros. "Method of fundamental solutions revisited: convergence in the energy norm for three-dimensional domains". In: *CAMWA(submitted)*. 2022.

[2] V. D. Kupradze and M. A. Aleksidze. "An approximate method of solving certain boundary-value problems". In: *Soobšč. Akad. Nauk Gruzin. SSR* 30 (1963), pp. 529–536.

[3] Alexander H.D. Cheng and Yongxing Hong. "An overview of the method of fundamental solutions—Solvability, uniqueness, convergence, and stability". In: *Eng. Anal. Bound. Elem.* 120 (2020), pp. 118–152. ISSN: 0955-7997. DOI: 10.1016/j.enganabound.2020.08.013.

[4] D. L. Young, M. H. Gu, and C. M. Fan. "The time-marching method of fundamental solutions for wave equations". In: *Eng. Anal. Bound. Elem.* 33.12 (2009), pp. 1411–1425. ISSN: 0955-7997. DOI: 10.1016/j.enganabound.2009.05.008. URL: https://doi.org/10.1016/j.enganabound.2009.05.008.

[5] Omar Askour et al. "Method of fundamental solutions and high order algorithm to solve nonlinear elastic problems". In: *Engineering Analysis with Boundary Elements* 89 (2018), pp. 25–35. ISSN: 0955-7997. DOI: 10.1016/j.enganabound.2018.01.007.

[6] Lu Lu et al. "DeepXDE: A deep learning library for solving differential equations". In: *SIAM Review* 63 (Feb. 2021), pp. 208–228.

[7] Yiorgos-Sokratis Smyrlis. "Density results with linear combinations of translates of fundamental solutions". In: *J. Approx. Theory* 161.2 (2009), pp. 617–633. ISSN: 0021-9045. DOI: 10.1016/j.jat.2008.11.018.

[8] Masashi Katsurada. "Asymptotic error analysis of the charge simulation method in a Jordan region with an analytic boundary". In: *J. Fac. Sci. Univ. Tokyo Sect. IA Math.* 37.3 (1990), pp. 635–657. ISSN: 0040-8980.

[9] Yiorgos-Sokratis Smyrlis and Andreas Karageorghis. "Numerical analysis of the MFS for certain harmonic problems". In: *M2AN Math. Model. Numer. Anal.* 38.3 (2004), pp. 495–517. ISSN: 0764-583X. DOI: 10.1051/m2an:2004023.

[10] Xin Li. "Convergence of the method of fundamental solutions for Poisson's equation on the unit sphere". In: *Adv. Comput. Math.* 28.3 (2008), pp. 269–282. ISSN: 1019-7168. DOI: 10.1007/s10444-006-9022-3.

[11] A. H. Barnett and T. Betcke. "Stability and convergence of the method of fundamental solutions for Helmholtz problems on analytic domains". In: *J. Comput. Phys.* 227.14 (2008), pp. 7003–7026. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2008.04.008.

[12] Ji Lin, Wen Chen, and Fuzhang Wang. "A New Investigation into Regularization Techniques for the Method of Fundamental Solutions". In: *Math. Comput. Simul.* 81.6 (2011), 1144–1152. ISSN: 0378-4754. DOI: 10.1016/j.matcom.2010.10.030.

[13] Yiorgos-Sokratis Smyrlis. "The method of fundamental solutions: a weighted least-squares approach". In: *BIT* 46.1 (2006), pp. 163–194. ISSN: 0006-3835. DOI: 10.1007/s10543-006-0043-6.

[14] P. P. Carvalho et al. "Some new results for geometric inverse problems with the method of fundamental solutions". In: *Inverse Probl. Sci. Eng.* 29.1 (2021), pp. 131–152. ISSN: 1741-5977. DOI: 10.1080/17415977.2020.1782398.

[15] I.E. Lagaris, A. Likas, and D.I. Fotiadis. "Artificial neural networks for solving ordinary and partial differential equations". In: *IEEE Transactions on Neural Networks* 9.5 (1998), pp. 987–1000. DOI: 10.1109/72.712178.

[16] M. Raissi, P. Perdikaris, and G.E. Karniadakis. "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations". In: *Journal of Computational Physics* 378 (2019), pp. 686–707. ISSN: 0021-9991. DOI: https://doi.org/

10.1016/j.jcp.2018.10.045. URL: https://www.sciencedirect.com/science/article/pii/S0021999118307125.

[17] Kailai Xu and Eric Darve. "Physics constrained learning for data-driven inverse modeling from sparse observations". In: *Journal of Computational Physics* 453 (2022), p. 110938. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2021.110938. URL: https://www.sciencedirect.com/science/article/pii/S0021999121008330.

[18] Aditi S. Krishnapriyan et al. "Characterizing possible failure modes in physics-informed neural networks". In: *NeurIPS*. 2021.

[19] Haoxiang Huang, Yingjie Liu, and Vigor Yang. "Neural Networks with Inputs Based on Domain of Dependence and A Converging Sequence for Solving Conservation Laws, Part I: 1D Riemann Problems". In: Sept. 2021.

[20] Ben Stevens and Tim Colonius. "FiniteNet: A Fully Convolutional LSTM Network Architecture for Time-Dependent Partial Differential Equations". In: Feb. 2020.

[21] Yohai Bar-Sinai et al. "Learning data-driven discretizations for partial differential equations". In: *Proceedings of the National Academy of Sciences* 116.31 (2019), pp. 15344–15349. DOI: 10.1073/pnas.1814058116. URL: https://www.pnas.org/doi/abs/10.1073/pnas.1814058116.

[22] Siddhartha Mishra. "A machine learning framework for data driven acceleration of computations of differential equations". In: *ArXiv* abs/1807.09519 (2018).

[23] Izsak Domonkos Haffner ; Ferenc. "Solving the Laplace equation by using neural networks". In: *Conference on Developments in Computer Science : Budapest, Hungary,* Eötvös Loránd University, Faculty of Informatics. 2021, pp. 143–146.

[24] Stefan A. Sauter and Christoph Schwab. *Boundary element methods*. Vol. 39. Springer Series in Computational Mathematics. Springer-Verlag, Berlin, 2011, pp. xviii+561. ISBN: 978-3-540-68092-5. DOI: 10.1007/978-3-540-68093-2.

[25] Guangming Yao. *Local radial basis function methods for solving partial differential equations*. Thesis (Ph.D.)–The University of Southern Mississippi. ProQuest LLC, Ann Arbor, MI, 2010, p. 174. ISBN: 978-1124-25108-0. URL: http://gateway.proquest.com/openurl?url_ver=Z39.88-2004&rft_val_fmt=info:ofi/fmt:kev:mtx:dissertation&res_dat=xri:pqdiss&rft_dat=xri:pqdiss:3424894.

[26] Michael A. Golberg, Ching-Shyang Chen, and Sriganesh R. Karur. "Improved multiquadric approximation for partial differential equations". In: *Engineering Analysis With Boundary Elements* 18 (1996), pp. 9–17.

[27] M. A. Golberg and C. S. Chen. "The method of fundamental solutions for potential, Helmholtz and diffusion problems". In: *Boundary integral methods: numerical and mathematical aspects*. Vol. 1. Comput. Eng. WIT Press/Comput. Mech. Publ., Boston, MA, 1999, pp. 103–176. DOI: 10.1007/BF01200068. URL: https://doi.org/10.1007/BF01200068.

[28] M. D. Buhmann. *Radial basis functions: theory and implementations*. Vol. 12. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, Cambridge, 2003, pp. x+259. ISBN: 0-521-63338-9. DOI: 10.1017/CBO9780511543241. URL: https://doi.org/10.1017/CBO9780511543241.

[29] C. S. Chen, Y. C. Hon, and R. A. Schaback. *Scientific computing with radial basis functions*. Tech. Rep., Department of Mathematics, University of Southern Mississippi, Hattiesburg. 2005. URL: http://num.math.uni-goettingen.de/schaback/SCwRBF.pdf.

[30] Angel Muleshkov, MA Golberg, and C. Chen. "Particular solutions of Helmholtz-type operators using higher order Poly harmonic splines". In: *Computational Mechanics* 23 (Jan. 1999), pp. 411–419. DOI: 10.1007/s004660050420.

[31] M. Katsurada and H. Okamoto. "The collocation points of the fundamental solution method for the potential problem". In: *Computers & Mathematics with Applications* 31.1 (1996), pp. 123–137. ISSN: 0898-1221. DOI: https://doi.org/10.1016/0898-1221(95)00186-3. URL: https://www.sciencedirect.com/science/article/pii/0898122195001863.

[32] Graeme Fairweather and Andreas Karageorghis. "The method of fundamental solutions for elliptic boundary value problems". In: vol. 9. 1-2. Numerical treatment of boundary integral equations. 1998, pp. 69–95. DOI: 10.1023/A:1018981221740. URL: https://doi.org/10.1023/A:1018981221740.

[33] K. Balakrishnan and P. A. Ramachandran. "The method of fundamental solutions for linear diffusion-reaction equations". In: *Math. Comput. Modelling* 31.2-3 (2000), pp. 221–237. ISSN: 0895-7177. DOI: 10.1016/S0895-7177(99)00233-2. URL: https://doi.org/10.1016/S0895-7177(99)00233-2.

[34] Alexander Bogomolny. "Fundamental solutions method for elliptic boundary value problems". In: *SIAM J. Numer. Anal.* 22.4 (1985), pp. 644–669. ISSN: 0036-1429. DOI: 10.1137/0722040. URL: https://doi.org/10.1137/0722040.

[35] Masashi Katsurada. "A mathematical study of the charge simulation method. II". In: *J. Fac. Sci. Univ. Tokyo Sect. IA Math.* 36.1 (1989), pp. 135–162. ISSN: 0040-8980.

[36] Takashi Kitagawa. "On the numerical stability of the method of fundamental solution applied to the Dirichlet problem". In: *Japan J. Appl. Math.* 5.1 (1988), pp. 123–133. ISSN: 0910-2043. DOI: 10.1007/BF03167903. URL: https://doi.org/10.1007/BF03167903.

[37] Takashi Kitagawa. "Asymptotic stability of the fundamental solution method". In: *Proceedings of the International Symposium on Computational Mathematics (Matsuyama, 1990)*. Vol. 38. 1-3. 1991, pp. 263–269. DOI: 10.1016/0377-0427(91)90175-J. URL: https://doi.org/10.1016/0377-0427(91)90175-J.

[38] Lawrence Shampine and Skip Thompson. "Stiff systems". In: *Scholarpedia* 2 (Jan. 2007), p. 2855. DOI: 10.4249/scholarpedia.2855.

[39] Wikipedia contributors. *Bessel function — Wikipedia, The Free Encyclopedia*. [Online; accessed 27-April-2022]. 2022. URL: https://en.wikipedia.org/wiki/Bessel_function.

List of Figures

1.1 Inner (blue), boundary (green) and outer (brown) points. 8
1.2 Neural network structure of the first approach 9
1.3 Neural network structure of the second approach 12
1.4 The outer and boundary points for example 1 14
1.5 Analytic and numerical solutions are solved by the first approach (left) and the second approach (right) of example 1 15
1.6 The effect of ϵ and the number of epochs(the first approach) 16
1.7 Nodal distribution of boundary and outer points. 18
1.8 The analytical and numerical solution on the unit circle 19
1.9 The analytical and numerical solution on epitrochoid 20
1.10 The numerical solution of the Neumann problem using the first approach. 20
1.11 The analytical and numerical solution on Amoeba-like domain 21
1.12 The analytical and numerical solution on flower-like domain 22

2.1 Nodal distributions on the sphere. 29
2.2 Boundary and outer points for the problem (2.2.2). 30
2.3 Error function on the different levels 30
2.4 Computation domain for the brick problem. 31
2.5 Error function on the level $z = 1$. 31

4.1 The computational domain and outer points with $\alpha = 0.15$. 46

5.1 Neural network architecture for ODE 50
5.2 Analytical solution and numerical solutions solving by explicit Euler method and the adaptive method of the motivation example with the initial value $u_0 = 5$. 52
5.3 The neural network structure for one-step ODE. 53

5.4 Comparison of numerical solutions to the analytical solution: (Adaptive1) all-in-one training, (Adaptive2) one-by-one training with one parameter, (Adaptive3) one-by-one training with more parameters 54
5.5 Numerical solutions and the analytical solution for flame propagation problem 56
5.6 The neural network structure for one-step Lax-Friedrichs scheme . . 58
5.7 The numerical solutions solved in coarse grid, fine grid by L-F scheme and by neural network. 59
5.8 The ration of errors in (5.4.2). 59

List of Tables

1.1 The effect of the number of boundary points and outer points (the first approach) . . . 16
1.2 The different choices of the number of boundary points and outer points, $\epsilon = 0.15$(the second approach) . . . 17
1.3 The different choices of magnification factor (the second approach), $n = 1000, m = 100$ . . . 17
1.4 The approximation of some interior points using both methods. . . . 18
1.5 The numerical solution of specific points on the unit circle (the second approach). . . . 19
1.6 The numerical solution at some points for the Neumann boundary value problem using the second approach. . . . 21
1.7 The numerical solution of some points on the mixed boundary problem using the second approach . . . 22
1.8 Different neural network structures for Example 1. . . . 23
1.9 Different neural network structures for Amoeba-like domain problem 24

2.1 Different nodal distribution leads to different error distribution 28

3.1 Common types of radial basis functions. . . . 34
3.2 Absolute error in approximation to $u(x, y, z)$ of Poisson equation. . . 39
3.3 Absolute error in approximation to $u(x, y, z)$ for Helmholtz equation. 40
3.4 Absolute error in approximation to $u(x, y, z)$ for Helmholtz equation . 40

4.1 Error estimation when $N = M$ in the maximum norm with $\alpha = 0.15$ (up) and $\alpha = 0.2$ (down). . . . 46
4.2 Error estimation with different (N, M) in different norms($\alpha = 0.2$) . . 47

5.1 The optimized coefficients obtained by all-in-one and one-by-one training. . . . 52

5.2 Optimized weights obtained by different techniques. 54