LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-SR-870166

Final Report: Non-Intrusive Parallel-in-Time Solvers for Partial Differential Equations

D. Vallejo, T. Le, B. Li, M. Luo

October 4, 2024

Disclaimer

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

Research in Industrial Projects for Students

Sponsor

Lawrence Livermore National Laboratory

Final Report

Non-Intrusive Parallel-in-Time Solvers for Partial Differential Equations

Student Members

Daniel Agraz Vallejo, *CETYS University*

Tran Duy Anh Le, *University of Rochester*

Bryan Li, *University of California, Berkeley*

Margaret Luo (Project Manager), *University of California, San Diego*

Academic Mentor

Jean-Michel Maldague

Sponsoring Mentors

Rob Falgout

Rui Peng Li

Wayne Mitchell

Daniel Osei-Kuffuor

Date: August 20, 2024

This project was jointly supported by Lawrence Livermore National Laboratory and NSF Grant DMS 1925919.

Abstract

Many time-dependent problems and simulations are often modeled using Partial Differential Equations. Traditional modeling approaches that use sequential time-stepping are reaching a bottleneck in optimizing efficiency. The Center of Applied Science and Computing at Lawrence Livermore National Laboratory extensively works on parallelizing these algorithms to leverage the increasing computational power from the growing number of processors in computer hardware. In particular, they aim to design non-intrusive algorithms that can generalize to a variety of problems and sizes without requiring additional information from or modifications on the original problems. Multigrid Reduction in Time (MGRIT) is a parallel-in-time algorithm that is designed to be non-intrusive. This project focuses on increasing the efficiency of MGRIT by approximating the coarse-grid operator using machine learning approaches as a means to find the most non-intrusive, or general, solution.

Acknowledgments

As a research team, we would like to express our gratitude to all institutions and experts who provided invaluable support for the development of this project.

First and foremost, we would like to thank the Institute of Pure and Applied Mathematics (IPAM) for organizing the summer program Research in Industrial Projects for Students (RIPS), which gathered talented individuals passionate about science and gave us the opportunity to apply mathematics to real-world problems. We extend our thanks to our academic mentor Jean-Michel Maldague for his support and guidance on the mathematical background required for the project and his useful feedback throughout the program.

We also wish to acknowledge Lawrence Livermore National Laboratory (LLNL) in conjunction with the Center of Applied Scientific Computing (CASC), for their trust and cooperation in this program. Finally, we would like to extend our appreciation to our industry sponsoring mentors Robert Falgout, Wayne Mitchell, Daniel Osei-Kuffour, and Rui Peng Li for sharing their knowledge, providing guidance, and supporting the completion of this project.

Contents

Abstract **3**

Acknowledgments **5**

1 Introduction **13**
1.1 Lawrence Livermore National Laboratory . . . 13
1.2 Motivation . . . 13
1.3 Report Overview . . . 14

2 Mathematical Background **15**
2.1 Discretization with finite difference . . . 15
2.2 Relaxation Methods . . . 16
2.3 Multigrid Methods . . . 17
2.4 Multigrid reduction in time (MGRIT) . . . 19

3 Machine Learning approach **21**
3.1 Neural Network Architecture . . . 21
3.2 The Loss Functions . . . 21

4 Results **23**
4.1 Loss Landscapes . . . 23
4.2 Optimal Stencils . . . 28
4.3 Neural Network integrated into PyMGRIT . . . 29
4.4 Commentary on Data . . . 30

5 Summary and Further Research **33**
5.1 Conclusion . . . 33
5.2 Future Directions . . . 33

A Analyses of Loss Functions **37**

B Abbreviations **41**

Selected Bibliography Including Cited Works **43**

List of Figures

1.1 50 Years of Microprocessor Trend Data 14

2.1 Multigrid Hierarchy 18
2.2 Multigrid V-Cycle 19
2.3 Coarse and fine time grid 20

4.1 Loss Landscape of the loss functions 24
4.2 Loss Landscape of the loss functions (cont) 25
4.3 Loss Landscape of $\mathcal{L}_1$, with and without the constant vector 26
4.4 Loss Landscape of $\mathcal{L}_2$, with and without the constant vector 26
4.5 Loss Landscape of $\mathcal{L}_{3,5}$ and $\mathcal{L}_{3,10}$, with and without the constant vector 27
4.6 Loss Landscape of $\mathcal{L}_{3,20}$ and $\mathcal{L}_{3,100}$, with and without the constant vector 28

5.1 $\mathcal{L}_1$ with periodic boundary condition, $\epsilon = 0.01$ 34

List of Tables

4.1 Loss function minimizers and PyMGRIT performance. 29
4.2 Comparison between neural network output loss to re-discretization and PyMGRIT performance 30

Chapter 1

Introduction

1.1 Lawrence Livermore National Laboratory

Lawrence Livermore National Laboratory (LLNL), a federally funded research institution in California, is one of three national laboratories under the National Nuclear Security Administration and it is operated in partnership by Lawrence Livermore National Security.

For over 70 years, LLNL has leveraged science and technology to make the world a safer place. The institution conducts mission driven research in the areas of nuclear and multi-domain deterrence, and seeks to assure the safety and reliability of the national nuclear stockpile, threat preparedness, climate and energy security.

The laboratory's expertise in science and engineering, along with its leading experimental capabilities and world-class research, are achieving milestones to address some of society's greatest challenges. Later this year, LLNL will deploy "El Capitan", a parallel supercomputer capable of performing 2 exaFLOPS (10^{18} floating point operations per second), making it the fastest supercomputer in the world.

This project is overseen by the Center of Applied Scientific Computing (CASC) branch at LLNL. Within the branch, they have various groups conducting scientific research in computational physics, computer science and applied mathematics on problems critical to national security.

CASC applies the power of high performance computing (HPC) and the efficiency of modern computational methods to the realms of stockpile stewardship, cyber and energy security, and knowledge discovery for intelligence applications. Moreover, CASC focuses on increasing the simulation fidelity and resolution of multi-physics and multi-scale models through the usage of advanced numerical methods and efficient algorithms. Furthermore, they create computing tools and programming environments that support extreme-scale computing.

1.2 Motivation

Over the past two decades, there has been a trend that depicts the current hardware design overview. While the number of transistors and logical cores per processor has continued to increase, clock speeds have remained stagnant, in other words, adding more cores no longer boosts processor speed as shown in Figure 1.1. As a consequence, sequential time marching algorithms have reached a bottleneck that fail to fully utilize the potential of modern computers.

This limitation motivates the need for algorithmic research aimed on enhancing concurrency to improve the convergence of iterative solutions. In particular, the development of non-intrusive

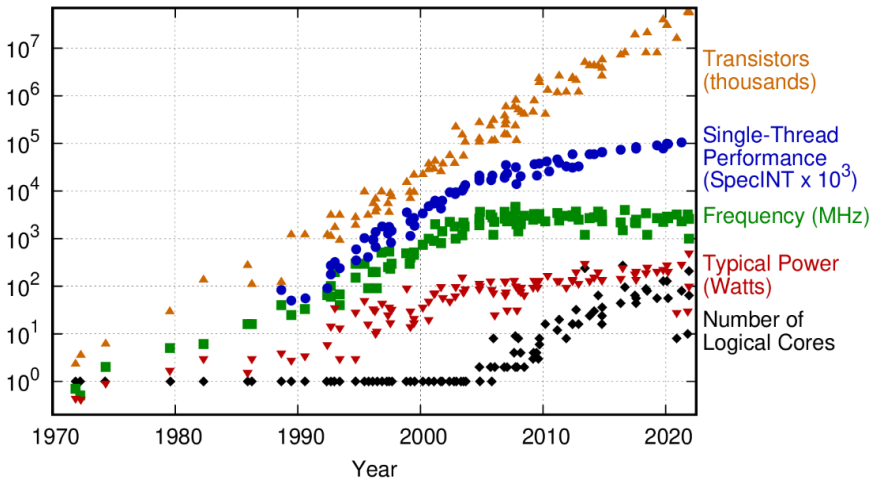


Figure 1.1: Since 2005, we have seen a continuous growth in the number of logical cores in processors. However, the processors clock speed have not been improved at all since then. This exemplifies the current situtation of hardware limitations. Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K.

algorithms is a priority. Non-intrusive algorithms can generalize to a wide array of problems and various problem sizes without requiring additional information from or modifications on the original problems. An intrusive algorithm, unlike its counterpart, demands significant effort and explicit changes to the code. While it offers greater efficiency and faster convergence, it requires experts on the code team. By taking a non-intrusive approach, we aim to generalize the time stepping routine the algorithm employs, so it is problem agnostic.

1.3 Report Overview

The remainder of this report consists of chapter 2 detailing mathematical concepts relevant to this project, then chapter 3 providing a comprehensive overview of our algorithmic developments on the neural network focusing on our loss functions. Next, we illustrate our observations and testing results in chapter 4. Finally, we conclude with chapter 5 by summarizing our advancements and recommendations for future directions of this problem.

Chapter 2

Mathematical Background

2.1 Discretization with finite difference

Finite difference methods (FDM) are numerical techniques used to approximate derivatives by using differences between function values at discrete points. We will describe this method of discretization in detail for the 1D steady-state diffusion equation.

$$-u_{xx} = f \tag{2.1}$$

where $u = u(x)$ for $0 \leq x \leq 1$ is unknown, $f = f(x)$ is given, and $u_{xx} = \frac{d^2u}{dx^2}$.

To solve this differential equation numerically, we employ discretization methods which convert continuous models into a discrete counterpart by dividing the domain into a finite set of points.

Let N be the number of partitions in space. For any $i = 0, \ldots, N$, let $x_i = hi$, $\delta x = \frac{1}{N}$ and $u_i \approx u(x_i)$. Note that for $x \in [0, 1]$ we have

$$u'(x_i) = \lim_{\mu \to 0} \frac{u(x_i + \mu) - u(x)}{\mu} \approx \frac{u(x_i + \delta x) - u(x_i)}{\delta x}$$

for sufficiently small δx. Then from averaging forward and backward difference, we have

$$u''(x_i) \approx \frac{u'(x_i + \frac{\delta x}{2}) - u'(x_i - \frac{\delta x}{2})}{\delta x}.$$

Applying the approximation for u' we get

$$\begin{aligned} -u''(x_i) &\approx -\frac{u(x_i + \delta x) - 2u(x_i) + u(x_i - \delta x)}{\delta x^2} \\ &= \frac{-u(x_{i+1}) + 2u(x_i) - u(x_{i-1})}{\delta x^2} \\ &\approx \frac{-u_{i+1} + 2u_i - u_{i-1}}{\delta x^2} \end{aligned}$$

We can rewrite this as a linear system $A\mathbf{u} = \mathbf{f}$:

$$\frac{1}{\delta x^2} \begin{pmatrix} 2 & -1 & 0 & \cdots & 0 \\ -1 & 2 & -1 & & \vdots \\ 0 & -1 & \ddots & & 0 \\ \vdots & & & & -1 \\ 0 & \cdots & 0 & -1 & 2 \end{pmatrix} \begin{pmatrix} u_0 \\ u_1 \\ \vdots \\ u_N \end{pmatrix} = \begin{pmatrix} f_0 \\ f_1 \\ \vdots \\ f_N \end{pmatrix}$$

Next we can consider the application of this method for the *advection-diffusion equation*, a linear partial differential equation (PDE) that is used to describe the behaviour of a scalar field such as the concentration of a pollutant or solvent in a liquid, that varies with both space and time.

The general form of the 1D advection-diffusion equation for $u\colon [0,1]_x \times [0,1]_t \to \mathbb{R}$ is given by

$$u_t - \varepsilon u_{xx} + a u_x = f, \tag{2.2}$$

where $\varepsilon \in [0, \infty)$, $a = a(x)$, $f = f(x,t)$. Let x and t represent our space and time dimensions, respectively. We divide the interval into evenly-spaced grid nodes with $S+1$ as the number of spatial divisions and $T+1$ as the number of time divisions. A uniform discretization sets

$$x_i = \delta x i, \quad \delta x = \frac{1}{S}$$

and

$$t_j = \delta t j, \quad \delta t = \frac{1}{T}$$

for any $i = 0, \ldots, S$ and $j = 0, \ldots, T$. We assume that δt and δx are chosen such that the Courant–Friedrichs–Lewy stability condition (CFL condition) holds [10]. Additionally, let $u_i^j \approx u(x_i, t_j)$. We can employ a finite difference scheme, assuming a is positive, to obtain the following discretizations:

$$\begin{aligned} u_t &\approx \frac{u_i^{j+1} - u_i^j}{\delta t} \\ -u_{xx} &\approx \frac{-u_{i-1}^j + 2u_i^j - u_{i+1}^j}{\delta x^2} \\ u_x &\approx \frac{u_{i+1}^j - u_i^j}{\delta x}. \end{aligned}$$

2.2 Relaxation Methods

Relaxation methods are iterative methods to solve linear systems of the form $A\mathbf{x} = b$, given $A \in \mathrm{GL}_n(\mathbb{R})$ (invertible $n \times n$ square matrices) and $b \in \mathbb{R}^n$. In the general form for relaxation using splitting, we let A be the sum of two matrices as such:

$$A = M + N.$$

Therefore our system $A\mathbf{x} = b$ becomes

$$(M+N)\mathbf{x} = \mathbf{b},$$

and we define the recurrence

$$M\mathbf{x}_{k+1} + N\mathbf{x}_k = \mathbf{b},$$

which after isolating x_{k+1} is

$$\mathbf{x}_{k+1} = M^{-1}(\mathbf{b} - N\mathbf{x}_k).$$

By adding and subtracting $M\mathbf{x}_k$ inside the parentheses, we get

$$\mathbf{x}_{k+1} = \mathbf{x}_k + M^{-1}(\mathbf{b} - A\mathbf{x}_k)$$

and define the *residual* $r = \mathbf{b} - A\mathbf{x}_k$.

The specific relaxation methods used this project are *Jacobi* and *Gauss-Seidel*. For these methods, let us consider $A = L + D + U$ where L is strictly lower triangular, D is diagonal, and U is strictly upper triangular.

In the *Jacobi* method, $M = D$ and we have the recurrence

$$\mathbf{x}_{k+1} = \mathbf{x}_k + D^{-1}(\mathbf{b} - A\mathbf{x}_k).$$

In the *Gauss-Seidel* method, $M = L + D$ and we have the recurrence

$$\mathbf{x}_{k+1} = \mathbf{x}_k + (L + D)^{-1}(\mathbf{b} - A\mathbf{x}_k).$$

Jacobi and *Gauss-Seidel* methods do not converge for all systems. Generally, the convergence of relaxation by splitting for a system can be determined through the following analysis.

Let $\mathbf{x}^* = A^{-1}\mathbf{b}$ be the true solution, and we define the error $\mathbf{e}_{k+1}$ to be the difference between x_{k+1}, the solution from the $k+1$ iteration, and x^*, the true solution. Then we can get the following expression for the error:

$$\begin{aligned}
\mathbf{e}_{k+1} &= \mathbf{x}_{k+1} - \mathbf{x}^* \\
&= \mathbf{x}_k + M^{-1}(\mathbf{b} - A\mathbf{x}_k) - \mathbf{x}^* \\
&= \mathbf{x}_k - \mathbf{x}^* + M^{-1}(A\mathbf{x}^* - A\mathbf{x}_k) \\
&= (I - M^{-1}A)(\mathbf{x}_k - \mathbf{x}^*) \\
&= (I - M^{-1}A)\mathbf{e}_k \\
&= (I - M^{-1}A)^{k+1}\mathbf{e}_0.
\end{aligned}$$

Here, $I - M^{-1}A$ is also known as the *error propagator*. Under the assumption that the error propagator $I - M^{-1}A$ is diagonalizable, i.e. $I - M^{-1}A = PDP^{-1}$, the term $(I - M^{-1}A)^k = PD^kP^{-1}$ only converges to 0 if all eigenvalues of $I - M^{-1}A$ have magnitude less than 1. Therefore we have that if all eigenvalues of $I - M^{-1}A$ have magnitude less than 1, $x_k \to A^{-1}b$ as $k \to \infty$.

However one drawback for relaxation methods is that while they are good at correcting oscillatory errors, they are not as effective at correcting smooth errors. This means they take more iterations to correct smooth errors. This ultimately leads to the employment of Multigrid methods which attacks this weakness of relaxation methods.

2.3 Multigrid Methods

Like mentioned previously, weighted *Jacobi* and *Gauss-Seidel* methods are good at eliminating high-frequency components of error, but are significantly slower at eliminating low-frequency components. In particular, these smooth, low-frequency, errors are characterized to have small eigenmodes. Assuming A is symmetric and has zero row sum, i.e. $a_{ii} = -\sum_{j \neq i} a_{ij}$, if e is some smooth

error vector, then

$$\begin{aligned}
e^\top Ae &= \sum_i e_i(a_{ii}e_i + \sum_{j\neq i} a_{ij}e_j) \\
&= \sum_i e_i\big(\sum_{j\neq i}(-a_{ij})(e_i - e_j)\big) \\
&= \sum_{i<j} e_i(-a_{ij})(e_i - e_j) + \sum_{i>j} e_i(-a_{ij})(e_i - e_j) \\
&= \sum_{i<j} e_i(-a_{ij})(e_i - e_j) - \sum_{i<j} e_j(-a_{ji})(e_i - e_j) \\
&= \sum_{i<j} -a_{ij}(e_i - e_j)^2 \ll 1.
\end{aligned}$$

Smooth error vary slowly in the direction of "large" matrix coefficient, and thus are not corrected as efficiently in relaxation.

Multigrid methods employ a grid hierarchy in an attempt to convert low-frequency errors on the fine grid into high-frequency errors on coarser grids. This allows relaxation to be more effective than just operating on the fine grid.

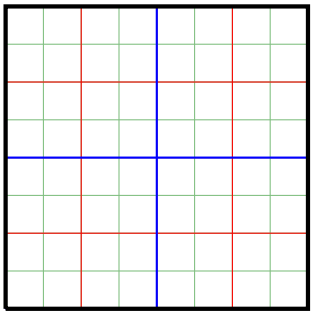


Figure 2.1: Multigrid Hierarchy

For some linear system $Au = f$ with $u^* = A^{-1}f$ as true solution. Given a guess u_0, we may compute the *residual* r as

$$r = f - Au_0 = A(u^* - u_0) = Ae,$$

where $e_0 = u^* - u_0$ is the *error*. Next, we solve the equation $Ax = r$, where the true solution is e_0. Then, given some approximation x to e_0, we can update our approximation to $u^* = u_0 + e_0$ with $u_1 = u_0 + x$.

Hereafter, we apply a restriction matrix to Ae that aims to downsample the residual error to a coarser grid. Thanks to this restriction step we are able to transition from a fine grid to a coarser one. We can restrict our linear system until we reach a defined coarsest level. The following example restricts values from a fine grid to a coarser grid with a coarsening factor $m = 2$.

$$\begin{pmatrix} u_0 \\ u_2 \\ u_4 \\ \vdots \\ u_{N-2} \\ u_N \end{pmatrix} = \begin{pmatrix} 1 & & & & & & & \\ 0 & 0 & 1 & & & & & \\ & & 0 & 0 & 1 & & & \\ & & & & \ddots & & & \\ & & & & 0 & 0 & 1 & \\ & & & & & & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} f_0 \\ f_1 \\ \vdots \\ f_N \end{pmatrix}$$

After reaching the coarsest level, we start performing an interpolation process that seeks to correct the error in the coarse grid. As we have restricted to coarser and coarser grids during the solution process, now we need to interpolate back to the original problem space. Below is an example of a matrix that interpolates from a coarse grid with $m = 2$ to a fine grid.

$$\begin{pmatrix} 1 & & & & & & \\ \frac{1}{2} & \frac{1}{2} & & & & & \\ & 1 & & & & & \\ & \frac{1}{2} & \frac{1}{2} & & & & \\ & & 1 & & & & \\ & & \ddots & & & & \\ & & \frac{1}{2} & \frac{1}{2} & & & \\ & & & 1 & & & \\ & & & \frac{1}{2} & \frac{1}{2} & & \\ & & & & 1 & & \end{pmatrix} \begin{pmatrix} u_0 \\ u_2 \\ u_4 \\ \vdots \\ u_{N-2} \\ u_N \end{pmatrix} = \begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \\ f_4 \\ \vdots \\ f_{N-2} \\ f_{N-1} \\ f_N \end{pmatrix}$$

The general multigrid algorithm includes different types of cycles, such as V-cycle, F-cycle, and W-cycle, that determine the order in which the restriction and interpolation steps are applied.

In the V-cycle we recursively solve the residual equation at each level and apply correction. Starting from the finest grid (the green grid), we relax then restrict our estimate to the next finest grid (the red grid). Then, we relax and restrict recursively until we hit the coarsest grid. At that point, we recursively relax and then interpolate our estimate to the next finest grid until we are back at the finest grid, at which point we have completed one multigrid V-cycle. The steps in the V-cycle can be summarized in Figure 2.2.

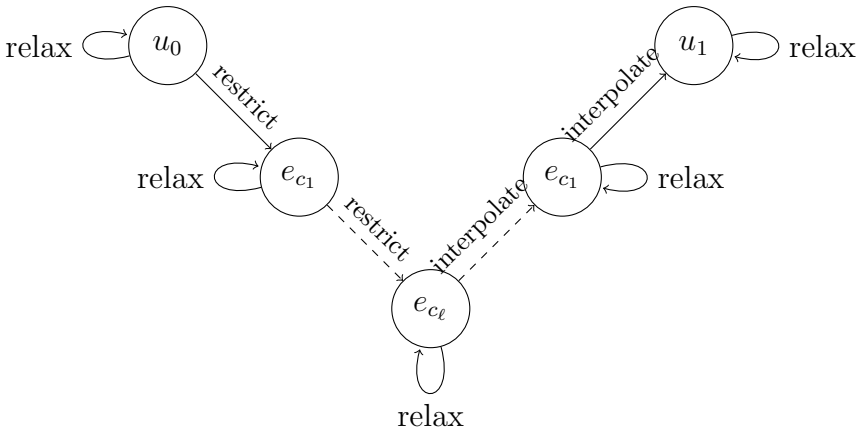


Figure 2.2: Multigrid V-Cycle

2.4 Multigrid reduction in time (MGRIT)

The multigrid reduction in time (MGRIT) algorithm is a parallel-in-time approach for solving time dependent problems that is designed to be as non-intrusive as possible on existing codes [7].

Consider the 1D diffusion equation

$$u_t - u_{xx} = f$$

with the following discretization using forward euler on a uniform space-time mesh with spacing δx and δt

$$u_t \approx \frac{u_i^{j+1} - u_i^j}{\delta t}, \quad -u_{xx} \approx \frac{-u_{i-1}^j + 2u_i^j - u_{i+1}^j}{\delta x^2}$$

where $u_i^j \approx u(x_i, t_j)$.

Let $\mathbf{u}^j \approx \begin{bmatrix} u_0^j & u_1^j & \cdots & u_N^j \end{bmatrix}^T$. We have the following update rule

$$\mathbf{u}^j = \Phi \mathbf{u}^{j-1} + \delta t \mathbf{f}^{j-1}$$

where $\Phi = I - \delta t L$ and L is the discretization for $-u_{xx}$

$$L = \frac{1}{\delta x^2} \begin{pmatrix} 2 & -1 & 0 & \cdots & 0 \\ -1 & 2 & -1 & & \vdots \\ 0 & -1 & \ddots & & 0 \\ \vdots & & & & -1 \\ 0 & \cdots & 0 & -1 & 2 \end{pmatrix}$$

and has the stencil

$$L \sim \frac{1}{\delta x^2} \begin{bmatrix} -1 & 2 & -1 \end{bmatrix}.$$

Thus Φ's stencil can also be written as

$$\phi \sim \begin{bmatrix} \beta & (1-2\beta) & \beta \end{bmatrix}, \quad \beta = \frac{\delta t}{\delta x^2} \leq \frac{1}{2}. \tag{2.3}$$

Performing a *von Neumann stability analysis* on this problem results in the constraint $\beta \leq 1/2$. For the technical details of this analysis, see [10] §9.6.

Using the partition in Figure 2.3, we let $t_k = k\delta t$ for the fine time grid and $T_j = mj\delta t$ for the coarse time grid with coarsening factor m. Then to update each coarse time grid, we solve the following system

$$A_\Delta \mathbf{u} = \begin{pmatrix} I & & & & \\ -\Phi^m & I & & & \\ & -\Phi^m & I & & \\ & & \ddots & \ddots & \\ & & & -\Phi^m & I \end{pmatrix} \begin{pmatrix} u_0 \\ u_m \\ u_{2m} \\ \vdots \\ u_N \end{pmatrix} = \mathbf{f}.$$

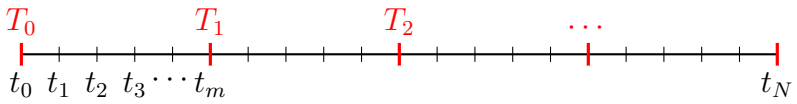


Figure 2.3: Coarse and fine time grid

Φ^m is known as the ideal *coarse grid operator*. However, as we take a banded matrix to a high power, the matrix gets denser as bandwidth increases. Φ^m becomes expensive to compute with and offers no parallel speedup, as it is computationally equivalent to computing m fine time steps. Therefore, we want to utilize machine learning techniques to find a practical (sparse) coarse operator Ψ that approximates the action of the ideal operator.

Chapter 3

Machine Learning approach

In this project, we approached the problem of approximating the ideal coarse grid operator Φ using machine learning techniques. The main motivation behind the choice is the potential for machine learning to provide the most general solutions, aligning well with our goal of developing non-intrusive algorithms. In this section we will give a detailed overview of components of the Neural Network model we have worked with.

3.1 Neural Network Architecture

We are using a Feedforward Neural Network (FNN). The input is a stencil $\phi \in \mathbb{R}^3$ along with the coarsening factor $m \in \mathbb{N}$ (the exponent) for the matrix Φ, as in Equation (2.3), and the output is a stencil $\psi \in \mathbb{R}^3$ for the matrix Ψ, which approximates Φ^m.

3.2 The Loss Functions

Our goal is to approximate the ideal coarse-grid timestepping operator

$$T = \begin{bmatrix} I & & & \\ -\Phi^m & I & & \\ & \ddots & \ddots & \\ & & -\Phi^m & I \end{bmatrix}$$

with a sparser matrix

$$A(v) = \begin{bmatrix} I & & & \\ -\Psi & I & & \\ & \ddots & \ddots & \\ & & -\Psi & I \end{bmatrix}$$

where Ψ is sparse (e.g. tridiagonal) and v are coefficients of A. For a good approximation, we want A to be "spectrally equivalent" to T.

Definition 1 (Spectral Equivalence) *Let A_k and B_k be sequences of symmetric matrices in $\mathbb{R}^{n\times n}$. Then, we say A_k and B_k are spectrally equivalent if the spectrum of $A_k^{-1}B_k$ are contained within the interval (c_1, c_2), where c_1, c_2 are constants close to 1 that do not depend on k.*

We recall that the "spectral norm" of a matrix $A \in \mathbb{R}^{n\times n}$ is given by the two following equivalent definitions:

1. The induced operator norm, considering the Euclidean norm on both the domain and codomain, given by the following formula

$$\|A\| = \sup_{\substack{v\in\mathbb{R}^n \\ v\neq 0}} \frac{\|Av\|_{\ell^2}}{\|v\|_{\ell^2}}.$$

2. The largest magnitude of eigenvalue of A, denoted $\rho(A) = \max\{|\lambda| \mid \det(A - \lambda I) = 0\}$.

Thus, A_k and B_k being spectrally equivalent is equivalent to $\|I - A_k^{-1}B_k\|$ being uniformly bounded. And we have

$$\|I - A_k^{-1}B_k\| \leq \|A_k - B_k\|\|A_k^{-1}\| = \frac{\|A_k - B_k\|}{\sigma_{min}A_k} \geq \frac{\|A_k - B_k\|}{\|A_k\|}$$

where σ_{min} is the smallest singular value fo A. We can now see that a natural loss function to minimize is based on this spectral norm. Given some vectors y_k, we can then attempt to minimize

$$\mathcal{L}_1 = \sum_k w_k \frac{\|Ty_k - A(v)y_k\|^2}{\|Ty_k\|^2} \tag{3.1}$$

where $w_1, \ldots, w_k$ are the list of weights. We can simply this further,

$$\frac{\|Ty_k - A(v)y_k\|^2}{\|Ty_k\|^2} = \frac{\sum_r (T_r y_k - A(v)_r y_k)^2}{\sum_r (T_r y_k)^2} \leq \sum_r \frac{(T_r y_k - A(v)_r y_k)^2}{(T_r y_k)^2}$$

where $T_r, A(v)_r$ denote row r of T and $A(v)$ respectively. By focusing on one row, and letting $y_k(i)$ be the restriction of y_k to the coarse time point i, we obtain a simplified expression

$$\mathcal{L}_1 = \sum_{k=1}^{K} \frac{((\Phi^m)_r y_k - \Psi_r y_k(i))^2}{(1 - (\Phi^m)_r y_k(i))^2}. \tag{3.2}$$

Our second idea for a loss function comes from Ru Huang et. al. [8], where we first compute the largest K (normalized) eigenvectors v_k of the ideal operator Φ^m, and then construct the loss function

$$\mathcal{L}_2 = \sum_{k=1}^{K} \|\Phi^m v_k - \Psi v_k\|^2. \tag{3.3}$$

Our third idea for a loss function follows similarly from the second loss function. If (λ_k, v_k) are normalized eigenpairs of the matrix Ψ, arranged in descending order of eigenvelues, then we construct the following loss function

$$\mathcal{L}_{3,M} = \sum_{k=1}^{K} \frac{(M-1)\|(\Psi - \Phi^m)v_k\|_{\ell_2}^2}{1 + (M-1)|1 - \lambda_k^m|^2} \tag{3.4}$$

where M is chosen to be a sufficiently large number or $M = \frac{N}{m} + 1$ where N is a large number and m is the coarsening factor. For more information, see Appendix A.

Chapter 4

Results

For further exploration and testing, we used the same Neural Network structure with the different loss functions introduced in section 3.2. We observed various characteristics and evaluated their performance at minimizing loss and achieving convergence in PyMGRIT. Ultimately, we want to understand how well our loss functions capture operators that lead to convergence in PyMGRIT.

For implementation, we modified Equation (3.2) slightly by adding ϵ in the denominator to avoid dividing by zero,

$$\mathcal{L}_1 = \sum_{k=1}^{K} \frac{((\Phi^m)_r y_k - \Psi_r y_k(i))^2}{(1 - (\Phi^m)_r y_k(i))^2 + \epsilon}. \tag{4.1}$$

Below are the five loss functions we tested closely on.

1. $\mathcal{L}_1$, Equation (4.1), where y_k are random vectors with an additional constant vector of all 1's.
2. $\mathcal{L}_1$, Equation (4.1), where y_k are largest eigenvectors of Φ.
3. $\mathcal{L}_2$, Equation (3.3), where y_k are largest eigenvectors of Φ.
4. $\mathcal{L}_{3,M}$, Equation (3.4), where y_k are largest eigenvectors of Φ. We test with $M = 5, 10, 20, 100$.
5. $\mathcal{L}_{3,\text{Hybrid}}$, Equation (3.4), where y_k are largest eigenvectors of Φ. We test with $M = \frac{1000}{m} + 1$ where m is the coarsening factor. We only consider this loss function for model training.

We also considered the following modifications of adding the constant vector of all 1's to the test vector list for $\mathcal{L}_1$ with eigenvectors, $\mathcal{L}_2$ and $\mathcal{L}_3$.

4.1 Loss Landscapes

To know whether our loss functions have a clear minimum to be reached, we want to look at the loss landscapes of them, i.e. the loss of every stencil returned by the loss function for a specific Φ and m.

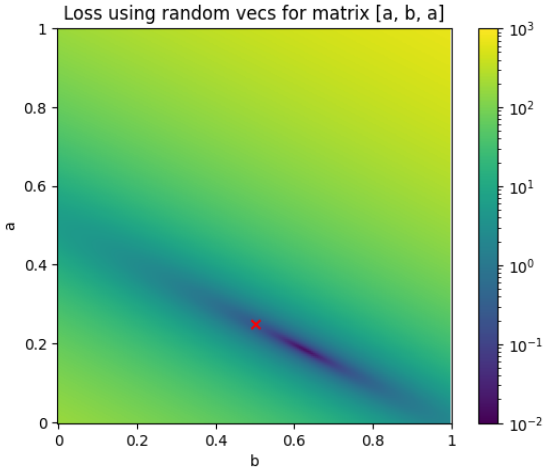
Consider the problem set up with

$$\delta x = \frac{1}{16}, \quad \delta t = \frac{1}{4096}, \quad \beta = \frac{1}{16}, \quad m = 4.$$

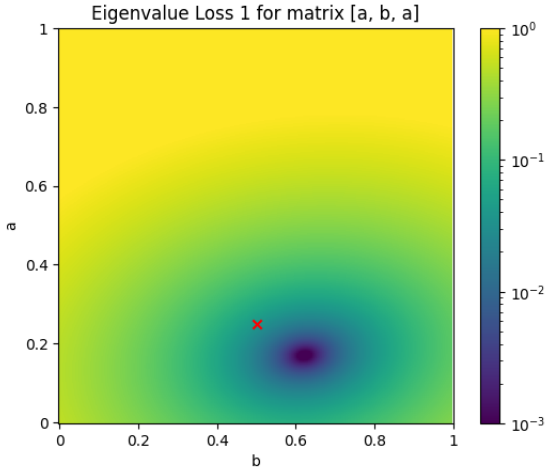
Note that this choice of δx and δt will satisfy the CFL condition, as mentioned in Chapter 2. The stencil for Φ is $\phi = \begin{bmatrix} 0.0625 & 0.8750 & 0.0625 \end{bmatrix}$. And the re-discretization stencil, the current approximation for Φ^m used in MGRIT obtained by discretizing again, is $\begin{bmatrix} 0.2500 & 0.5000 & 0.2500 \end{bmatrix}$.

We restrict the approximation Ψ to have stencil of the form $\begin{bmatrix} a & b & a \end{bmatrix}$ and plot the loss at (b, a) for each loss function. We also plot the re-discretization stencil, which we know converges well in PyMGRIT, in each loss function to see how "good" the loss functions consider the re-discretization stencil to be. This helps us gain an understanding of how good low loss corresponds to convergence in PyMGRIT.

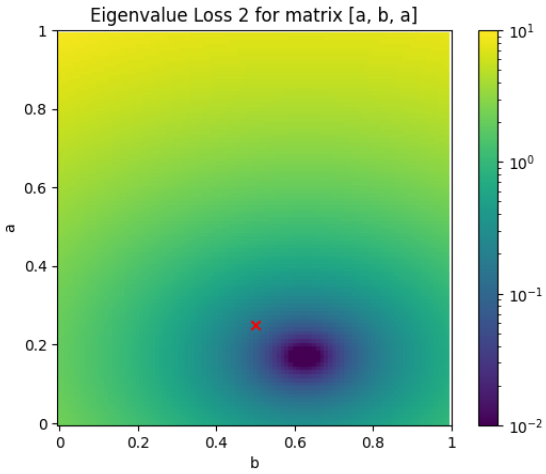
For Figures 4.1 and 4.2, $(b, a) \in [0, 1] \times [0, 1]$ with resolution 200×200, i.e. each interval is divided into 200 points. We have b on the x-axis, a on the y-axis, and the red x marks the re-discretization stencil. Each loss landscape evaluates the corresponding loss function with respect to the stencil $\begin{bmatrix} a & b & a \end{bmatrix}$, and the darker the region is, the smaller the value of the loss function is at that point.



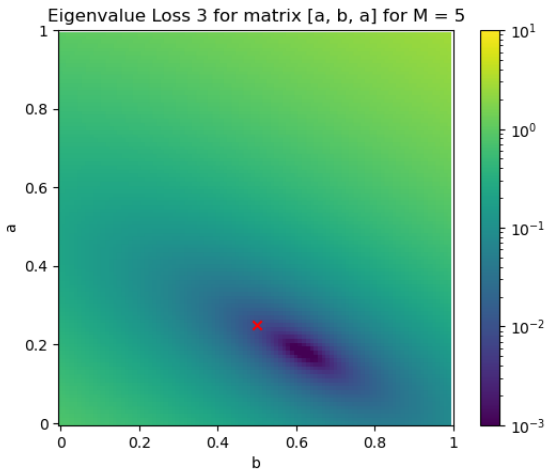
(a) $\mathcal{L}_1$, Equation (4.1), with Rand and Const vector, $\epsilon = 0.01$



(b) $\mathcal{L}_1$, Equation (4.1), with EVecs, $\epsilon = 0.01$

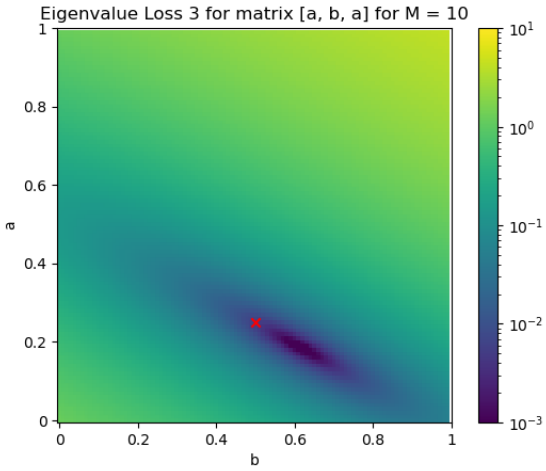


(c) $\mathcal{L}_2$, Equation (3.3), with Evecs

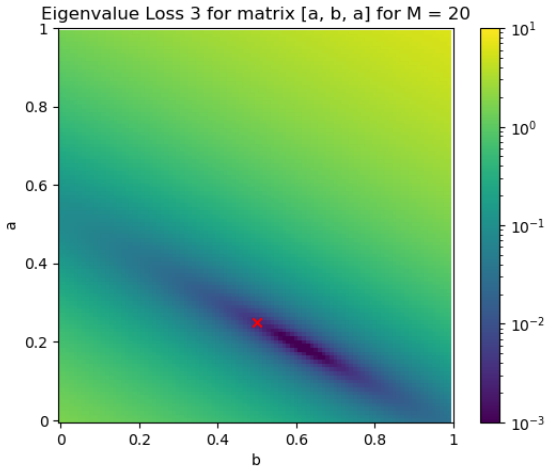


(d) $\mathcal{L}_{3,5}$, Equation (3.4), with Evecs

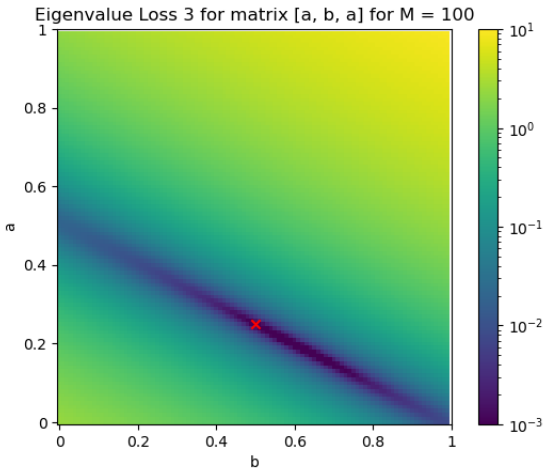
Figure 4.1: Loss Landscape of the loss functions



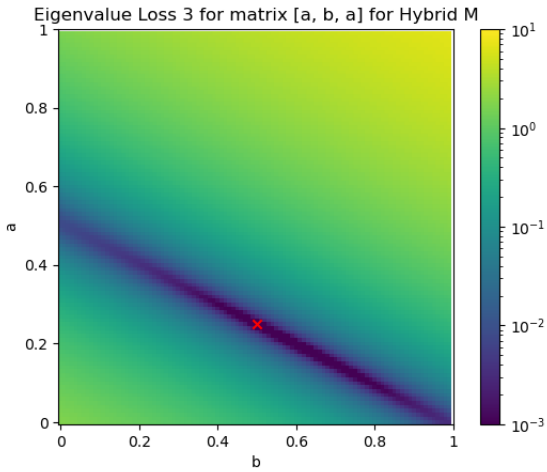
(a) $\mathcal{L}_{3,10}$, Equation (3.4), with Evecs



(b) $\mathcal{L}_{3,20}$, Equation (3.4), with Evecs



(c) $\mathcal{L}_{3,100}$, Equation (3.4), with Evecs

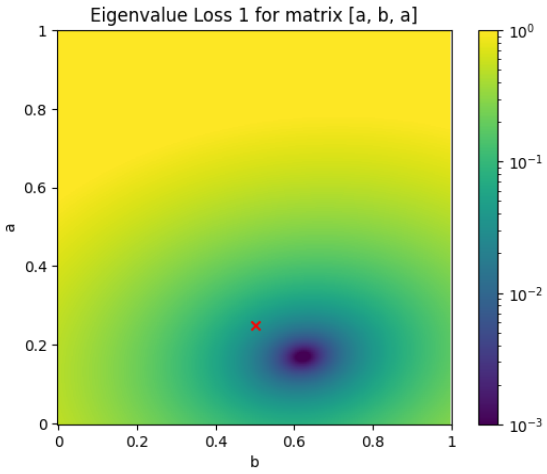


(d) $\mathcal{L}_{3,\text{Hybrid}}$, Equation (3.4), with Evecs

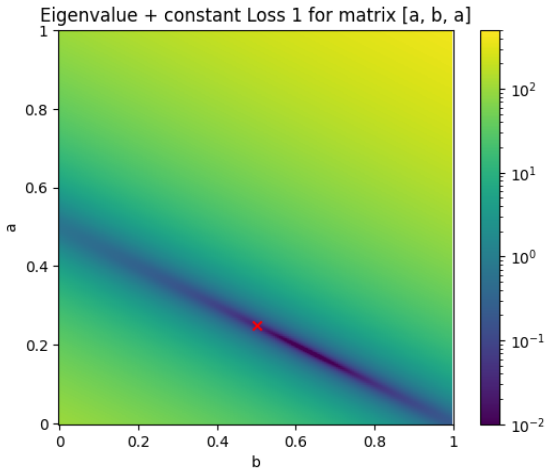
Figure 4.2: Loss Landscape of the loss functions (cont)

In Figures 4.1 and 4.2, we can see the each loss function does have a single clear minimum, which is the dark purple region, and that the landscape is convex. Note that the re-discretization stencil falls near but not precisely on the minimum of the loss landscape for all loss function. For the hybrid training, note that we only work on a fix coarsening factor so we have $M = \frac{1000}{4} + 1 = 251$. We see that as we increase M, the strip gets streched out via the region $2a + b \approx 1$.

To test further, we added the constant vector to the three loss functions using eigenvectors as test vectors to see how that affected the loss landscape. The results are shown in Figure 4.3 for $\mathcal{L}_1$.



(a) $\mathcal{L}_1$, Equation (4.1), with Evecs, $\epsilon = 0.01$



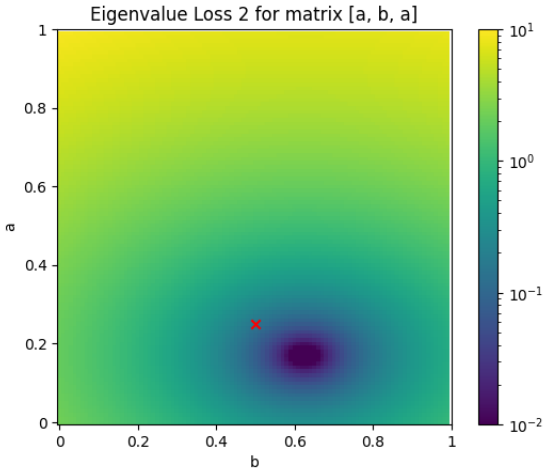
(b) $\mathcal{L}_1$, Equation (4.1), with Evecs and Const, $\epsilon = 0.1$

Figure 4.3: Loss Landscape of $\mathcal{L}_1$, Equation (4.1), with eigenvectors vs. eigenvectors and the constant vector

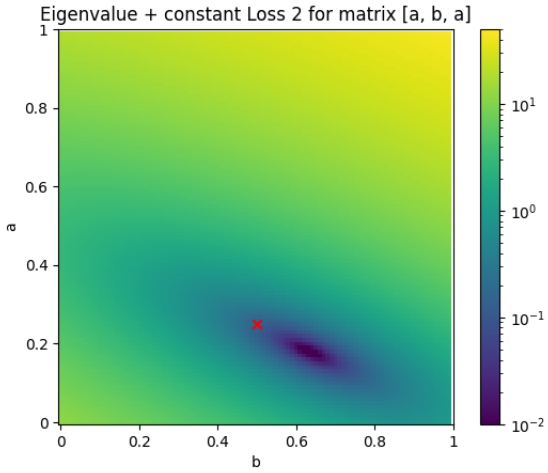
Note that $\epsilon = 0.1$ for $\mathcal{L}_1$ with both eigenvectors and constant vector, this is because turning down ϵ causes the loss to blow up and thus the model cannot be trained. So to keep the model consistent for both observing properties and training, $\epsilon = 0.1$ for this set of test vectors for $\mathcal{L}_1$ for all tests.

Here we can observe that when we add the constant vector, the graph gets stretched out. The minimum is no longer a circular area but resembles a line, and it traces the region where $2a+b \approx 1$. The addition of the constant vector allows the loss function to distinguish stencil with entries which add up to one. Note that the re-discretization stencil also has entries which sum up to one, so the property of summing up to one could be desirable to obtain stencils that converge well in PyMGRIT.

By looking at Figures 4.4, 4.5, and 4.6, a similar effect can be observed for adding the constant to $\mathcal{L}_2$ and $\mathcal{L}_3$ as well, although to a lesser extent.

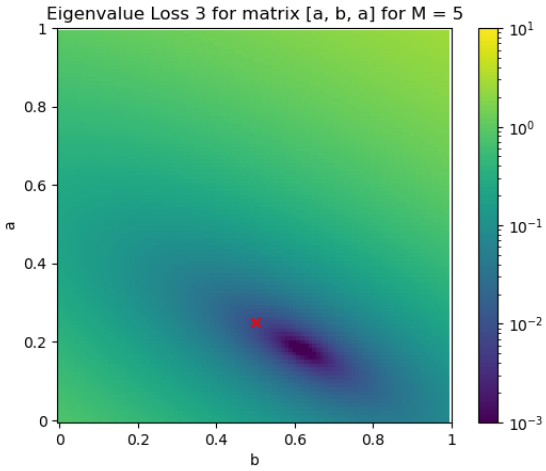


(a) $\mathcal{L}_2$, Equation (3.3), with Evecs

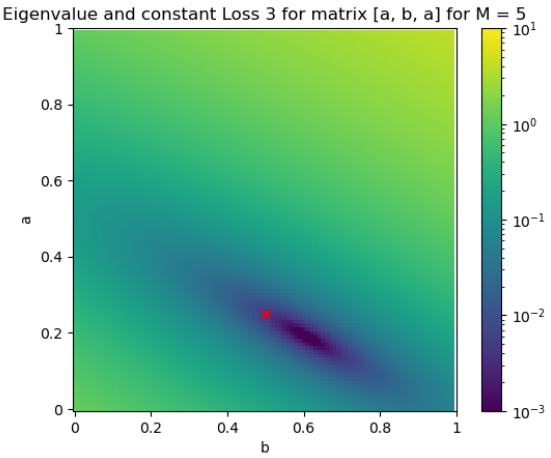


(b) $\mathcal{L}_2$, Equation (3.3), with Evecs and Const

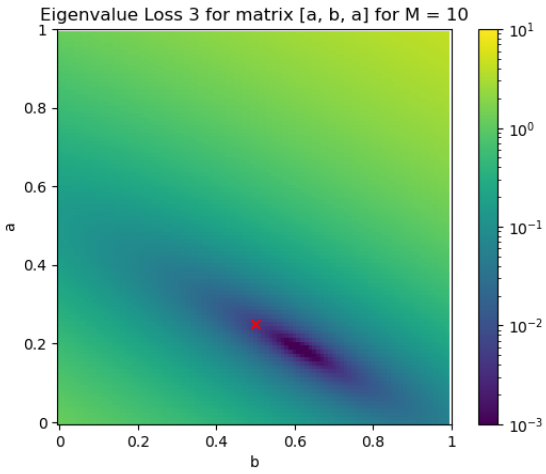
Figure 4.4: Loss Landscape of $\mathcal{L}_2$, Equation (3.3), with eigenvectors vs. eigenvectors and the constant vector



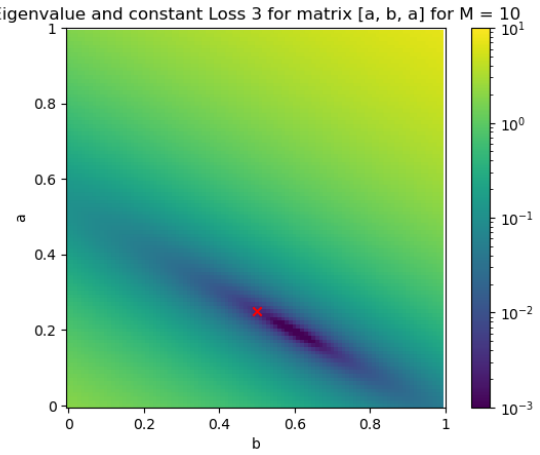
(a) $\mathcal{L}_{3,5}$, Equation (3.4), with Evecs



(b) $\mathcal{L}_{3,5}$, Equation (3.4), with Evecs and Const

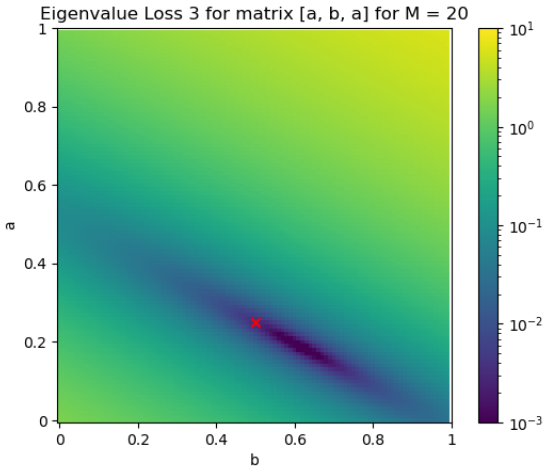


(c) $\mathcal{L}_{3,10}$, Equation (3.4), with Evecs

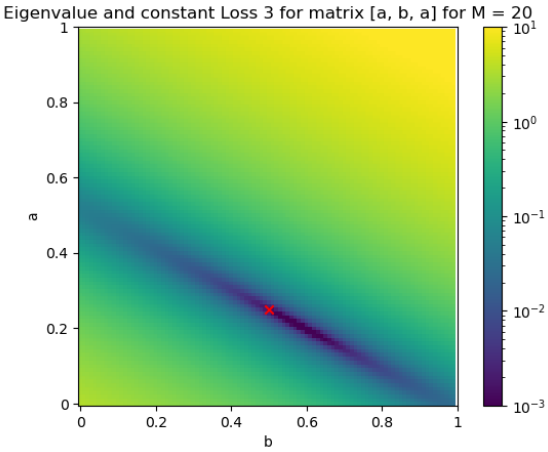


(d) $\mathcal{L}_{3,10}$, Equation (3.4), with Evecs and Const

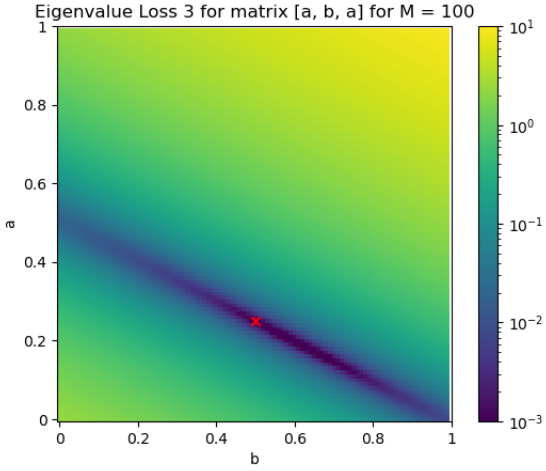
Figure 4.5: Loss Landscape of $\mathcal{L}_{3,5}$ and $\mathcal{L}_{3,10}$, (3.4), with eigenvectors vs. eigenvectors and the constant vector



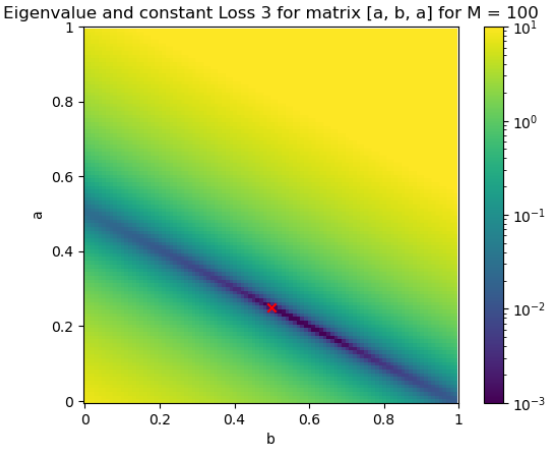
(a) $\mathcal{L}_{3,20}$, Equation (3.4), with Evecs



(b) $\mathcal{L}_{3,20}$, Equation (3.4), with Evecs and Const



(c) $\mathcal{L}_{3,100}$, Equation (3.4), with Evecs



(d) $\mathcal{L}_{3,100}$, Equation (3.4), with Evecs and Const

Figure 4.6: Loss Landscape of $\mathcal{L}_{3,20}$ and $\mathcal{L}_{3,100}$, Equation (3.4), with eigenvectors vs. eigenvectors and the constant vector

4.2 Optimal Stencils

We want to test how well these loss functions actually correspond to MGRIT convergence. To do this, we first compute the minimizers of these losses, which we also refer to as the optimal stencil, and then use that stencil instead of the re-descritization stencil in a 2-level MGRIT solver.

To generate optimal stencils for our loss functions, we used the `scipy` package `scipy.optimize.minimize` function, assuming a symmetric stencil.

Then, to test these, we run PyMGRIT on an inhomogenous heat equation on the domain $[0,1] \times [0,1]$ with periodic boundary conditions in space, given by

$$\begin{cases} u_t - u_{xx} &= -\sin(t)\sin^2(\pi x) - 2\pi^2 \cos(t)(\cos^2(\pi x) - \sin^2(\pi x)) \\ u(0,t) &= u(1,t) \\ u_x(0,t) &= u_x(1,t) \\ u(x,0) &= \sin^2(\pi x) \end{cases} . \tag{4.2}$$

We perform discretization with $n_x = 17, n_t = 4097$, and our 2-level MGRIT solver has a coarsening factor of 4. Then, we ran 10 multigrid V-cycles using the optimal stencil for each loss function to compute the coarse-grid operator, and recorded the residual after those 10 iterations, as well as the convergence factor from the 9th to 10th iteration. Note that since we are fixing m, $\mathcal{L}_{3,\text{Hybrid}}$ is $\mathcal{L}_{3,251}$.

Loss Function	Optimal Stencil			Residual	Conv. Factor
$\mathcal{L}_1$ Random	[0.1814	0.6324	0.1814]	6.60e-5	4.04e-1
$\mathcal{L}_1$ EVecs	[0.1700	0.6221	0.1700]	2.13e-2	9.40e-1
$\mathcal{L}_1$ EVecs, const.	[0.1824	0.6341	0.1824]	6.60e-6	2.82e-1
$\mathcal{L}_2$	[0.1705	0.6223	0.1705]	2.11e-2	9.59e-1
$\mathcal{L}_2$ const.	[0.1797	0.6346	0.1797]	1.55e-4	4.17e-1
$\mathcal{L}_{3,5}$	[0.1891	0.6071	0.1891]	6.82e-2	9.03e-1
$\mathcal{L}_{3,10}$	[0.2076	0.5789	0.2076]	3.93e-2	5.15e-1
$\mathcal{L}_{3,20}$	[0.2028	0.5895	0.2028]	1.63e-2	4.48e-1
$\mathcal{L}_{3,100}$	[0.2101	0.5783	0.2101]	1.71e-7	1.41e-1
$\mathcal{L}_{3,\text{Hybrid}}$	[0.2119	0.5754	0.2191]	1.65e-8	9.94e-2
$\mathcal{L}_{3,5}$, const.	[0.1977	0.5945	0.1977]	3.32e-2	7.62e-1
$\mathcal{L}_{3,10}$, const.	[0.2003	0.5925	0.2003]	8.03e-3	5.79e-1
$\mathcal{L}_{3,20}$, const.	[0.2061	0.5840	0.2061]	2.87e-3	3.47e-1
$\mathcal{L}_{3,100}$, const.	[0.2068	0.5848	0.2068]	2.37e-7	1.46e-1
$\mathcal{L}_{3,\text{Hybrid}}$, const.	[0.2085	0.5820	0.2085]	4.22e-8	1.16e-1

Table 4.1: Loss function minimizers and PyMGRIT performance.

4.3 Neural Network integrated into PyMGRIT

We are also interested in how effectively we are able to train the model to return outputs which have small loss. For this, we built a training set consisting of 16 elements, generated by the stencils with $\beta = 1/8, 1/12, 1/16, 1/24$ with coarsening factors $m = 1, 2, 3, 4$. We trained the same neural network structure on this training dataset for each loss function with minor modifications on the test vector set. Our neural network consisted of 3 hidden layers of 50 neurons each with the LeakyReLU activation function.

Our testing setup uses once again Equation (4.2) with domain $[0, 1] \times [0, 1]$, with the discretization mesh given by $n_x = 17, n_t = 2561$. This gives us $\beta = 1/10$ (note that this is not in the training set).

We compare the loss of the neural network's output for $m = 2$ to the loss of the re-discretization for each loss function to see how well the neural network trains. Additionally, we compare the PyMGRIT performance of these trained neural networks, along with re-discretization as a baseline. The PyMGRIT experiment was run with the problem as stated above, with 2-level V-cycle using a coarsening factor of $m = 2$.

	PyTorch Output			PyMGRIT Results	
Loss Function	Output Stencil	Output Loss	Re-disc. Loss	Residual	Conv. Factor
$\mathcal{L}_1$ Random	[0.131 0.743 0.131]	0.177	0.143	2.28	5.74e-1
$\mathcal{L}_1$ Evecs	[0.127 0.726 0.121]	6.75e-3	6.15e-3	1.85e-2	9.42e-1
$\mathcal{L}_1$ Evecs, const.	[0.159 0.693 0.150]	1.21e-3	4.83e-3	2.07e-4	4.03e-1
$\mathcal{L}_2$	[0.128 0.733 0.120]	4.03e-2	3.48e-2	1.13e-2	9.03e-1
$\mathcal{L}_2$ const.	[0.130 0.738 0.125]	4.15e-2	3.48e-2	1.69e-3	6.30e-1
$\mathcal{L}_{3,5}$	[0.209 0.509 0.255]	9.64e-3	8.07e-4	9.25e-2	9.86e-1
$\mathcal{L}_{3,10}$	[0.223 0.505 0.256]	7.26e-3	5.67e-4	7.84e-2	9.57e-1
$\mathcal{L}_{3,20}$	[0.239 0.485 0.266]	5.97e-3	3.40e-4	5.08e-2	8.64e-1
$\mathcal{L}_{3,100}$	[0.258 0.430 0.308]	3.45e-3	5.67e-4	9.68e-4	4.29e-1
$\mathcal{L}_{3,\text{Hybrid}}$	[0.246 0.473 0.275]	5.14e-3	2.48e-4	1.43e-2	6.57e-1
$\mathcal{L}_{3,5}$ const.	[0.159 0.664 0.165]	1.11e-3	3.62e-3	6.40e-2	9.13e-1
$\mathcal{L}_{3,10}$ const.	[0.168 0.666 0.166]	1.53e-3	2.76e-3	2.19e-8	1.67e-1
$\mathcal{L}_{3,20}$ const.	[0.196 0.619 0.199]	5.97e-3	3.40e-4	1e12	1.75
$\mathcal{L}_{3,100}$ const.	[0.241 0.514 0.254]	9.46e-3	8.26e-4	1e7	1.02
$\mathcal{L}_{3,\text{Hybrid}}$ const.	[0.243 0.420 0.334]	5.05e-2	1.47e-3	1.54e-2	6.81e-1
Re-disc.	[0.200 0.600 0.200]	—	—	7.60e-8	6.77e-2

Table 4.2: Comparison between neural network output loss to re-discretization and PyMGRIT performance

4.4 Commentary on Data

As we may observe in Table 4.1, only one minimizer did achieve better convergence than re-discretization in our PyMGRIT tests, such is the loss function $\mathcal{L}_3$ with $M = 10$ and a constant vector ($\mathcal{L}_{3,100}$ const.) leading to 2.19e-8 residual and 1.67e-1 as convergence factor, achieving the lowest convergence among the rest of loss functions.

For our experimentation, the same neural network structure was trained on different loss functions. By analyzing Table 4.2, we can note how each NN seems to do good regarding the stencil loss, returning outputs which have low loss. However, these stencils with roughly the same or lower loss than the re-discretization stencil does not lead to better convergence in PyMGRIT. This indicates to us that low loss does not directly translate to good performance in PyMGRIT, and our loss functions are not able to capture stencils which lead to convergence very accurately. Nevertheless, some level of convergence was observed for every loss function in this example.

After reviewing the results on both Table 4.1 and Table 4.2, we infer that $\mathcal{L}_{3,10}$ const., is the best performing loss function. Some possible inferences for this is (1) it employs large eigenvectors of Φ (small eigenvectors of T) as test vectors, and (2) it uses the constant vector which appears to allow the loss function to recognize stencils with entries summing up to 1 to be better. The property of sum of entries = 1 being significant in a good stencils can be observed from results of testing with and without the constant vectors for $\mathcal{L}_1$, $\mathcal{L}_2$ and $\mathcal{L}_3$ with other M values as well. We also see that as M gets larger, the residual for the third loss function decreases. Nevertheless, as we consider the constant vector, the residual from the output of $\mathcal{L}_{3,20}$ and $\mathcal{L}_{3,100}$ increase to a very large number. If we decrease the learning rate, the residual will decrease for the 2 upper loss functions.

A drawback to this loss function lies on the calculation of eigenvectors, as the matrix Φ gets more denser, it gets expensive to compute the largest eigenvector.

We also noticed that symmetric stencils tend to lead to better PyMGRIT performance. It is possible to force the neural network to return symmetric stencils by having it only output the middle value (b of $[a\ b\ a]$) and calculate $a = \frac{1-b}{2}$. While this approach may produce better results for the diffusion problem we have been testing, it has a more intrusive focus that misaligns with the purpose of the project.

Chapter 5

Summary and Further Research

5.1 Conclusion

Our ultimate goal in this project was to find a non-intrusive method to approximate the ideal coarse-time-stepping operator. The approach of using machine learning methods was chosen in hopes to ensure non-intrusiveness in the solution. We explored the relation between theorized good approximations (spectral equivalence) and empirical convergence in PyMGRIT through investigating multiple loss functions derived from the same definition. All loss functions we explored seems to have a clear minimum, and we are able to train the models to return outputs which minimized loss. In spite of that, optimal stencils given by the loss function did not correspond to best convergence in PyMGRIT.

5.2 Future Directions

5.2.1 Generalization

The application of this method to the 1D diffusion equation is only a proof-of-concept. Future direction can be to generalize this method to other types of equations, such as advection.

For other problems, a tridiagonal stencil for Ψ may not work well. For example, for the case of 1D advection, using a tridiagonal stencil for Ψ will eventually violate the CFL condition for large enough m. More explicitly, considering the 1D advection equation $u_t + u_x = 0$ with initial condition $u(x, 0) = g(x)$, we see that $g(x - t)$ is an analytical solution to this PDE. Thus we can see that the analytical domain of dependence of this PDE at point (x, t) is simply the set $\{x - t\}$.

However, considering the discretization scheme for u_t and u_x in section 2.1, we see that if $\Delta T = m\delta t > \delta x$, the numerical domain of dependence for any tridiagonal coarse grid operator does not include the anlaytical domain of dependence, violating CFL conditions.

One potential solution is using a second neural network to select the nonzero entries of Ψ, following [8], although this would require modifying the structure of our output vector.

5.2.2 Increasing the number of entries in stencil

For the diffusion problem we worked with, Ψ was hard-coded to be tridiagonal. However, for larger coarsening factors, Φ^m gets much denser, so larger stencils may result in better convergence. One

possible approach is to use a second neural network to select nonzero entries of Ψ, following Ru Huang et als' [8] paper.

5.2.3 Using bootstrapping techniques

We have focused on improving the loss function by choosing important vectors such as eigenvectors as test vectors which results in a more effective loss function than random vectors. However, computing eigenvectors may become expensive for larger problem sizes which motivates the bootstrapping method. For bootstrapping, we start with a set of random vectors as y_k (test vectors), then we test the result of the network by running MGRIT (with the trained neural net) to solve the homogeneous equation, whose solutions coincide with the error function. Then we correct the previous y_k by forming a new set $\{y_k\}_{new} = \{y_k + e_k\}$ to train the model with.

The bootstrap method is promising because it uncovers error components the current method is not correcting. When methods don't converge well, it is often that they are not correcting near null space components of A. Consider the general error propagator for linear methods $I - M^{-1}A$. Slow convergence means

$$(I - M^{-1}A)e \approx e$$

for some error e. Hence $M^{-1}Ae \approx 0$. So these components are always in the near null space of $M^{-1}A$ which are often components also in the near nullspace of A. So if we add these components to the loss function, we hope new Ψ operators returned by the model after training will be good at dealing with both original vectors and the new ones.

5.2.4 Implementing periodic boundary condition

The diffusion problem in PyMGRIT has periodic boundary conditions. However, for all loss functions we implemented with eigenvectors, the eigenvectors were computed with matrix of Dirichlet boundary condition. So one possible adjustment to potentially lead to improvement is by implementing the loss functions with periodic boundary condition. We have explored this a little and computed the loss landscape Figure 5.1 of $\mathcal{L}_1$, Equation (4.1), with periodic boundary condition and the corresponding optimal stencil: [0.1700 0.6220 0.1700]. From the results we received, there

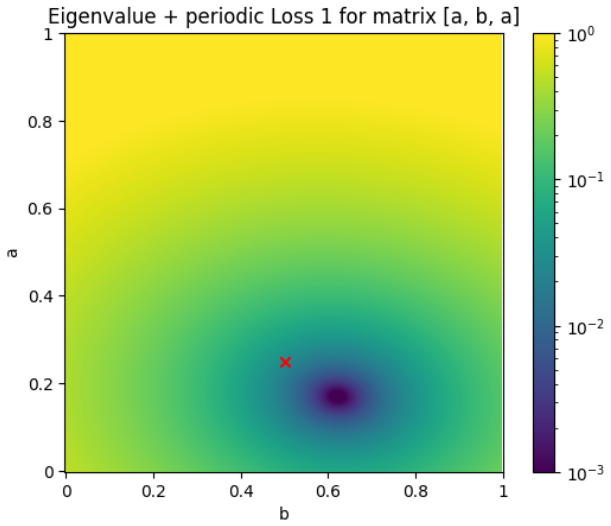


Figure 5.1: $\mathcal{L}_1$, Equation (4.1), with periodic boundary condition, $\epsilon = 0.01$

does not appear to be any significant difference from Dirichlet boundary condition. But additional in-depth testing cuold present different results. Some further testing such as implementing on other loss functions, observing the convergence of optimal stencil in PyMGRIT, and integrating trained models with periodic boundary condition with PyMGRIT could lead to better understanding.

Appendix A

Analyses of Loss Functions

The first loss function is derived by looking at a singular row of the matrix T and $A(v)$ as the row of Ψ is generated by a simple stencil. Note that from Section 3.2,

$$\frac{\|Ty_k - A(v)y_k\|_{\ell_2}^2}{\|Ty_k\|_{\ell_2}^2} = \frac{\sum_r (T_r y_k - A_r y_k)^2}{\sum_r (T_r y_k)^2} \leq \sum_r \frac{(T_r y_k - A_r y_k)^2}{(T_r y_k)^2}$$

where T_r, A_r are the r-th rows of T and $A(v)$ respectively. Hence, we could minimize a loss functional based on a representative r instead.

For some row $1 \leq r \leq Mn$, let $r = qn + d$, where $0 < d \leq n$ and $q, d \in \mathbb{N}_{\geq 0}$. Then for any $r > n$, the r-th row of $T(m)$ is

$$T_r = \left(\underbrace{0 \quad 0 \quad \dots \quad 0}_{n(q-1)\ 0\text{s}} \quad \underbrace{-\Phi_d^m}_{d\text{-th row of } -\Phi^m} \quad \underbrace{0 \quad 0 \dots \quad 0}_{d-1\ 0\text{s}} \quad \underbrace{1}_{\text{at postition } r} \quad \underbrace{0 \quad 0 \dots \quad 0}_{Mn-R\ 0\text{s}} \right) \in \mathbb{R}^{Mn}.$$

Similarly for $A(v)$, for any $r > n$,

$$A(v)_r = \left(\underbrace{0 \quad 0 \quad \dots \quad 0}_{n(q-1)\ 0\text{s}} \quad \underbrace{-\Psi_d}_{d\text{-th row of } -\Psi} \quad \underbrace{0 \quad 0 \dots \quad 0}_{d-1\ 0\text{s}} \quad \underbrace{1}_{\text{at postition } r} \quad \underbrace{0 \quad 0 \dots \quad 0}_{Mn-R\ 0\text{s}} \right) \in \mathbb{R}^{Mn}.$$

For each pair of index $i < j$, define $y_k[i][j] = \begin{pmatrix} y_{k,i} \\ y_{k,i+1} \\ \dots \\ y_{k,j} \end{pmatrix}$.Hence, we want to minimize

$$\min_{v_r} \sum_k \frac{((\Phi^m)_d y_k[n(q-1)+1][nq] - \Psi_d y_k[n(q-1)+1][nq])^2}{(y_k[r][r] - \Phi_d^m y_k[n(q-1)+1][nq])^2}$$

For each vector y_k, we define $y'_k = y_k[n(q-1)+1][nq] \in \mathbb{R}^n$ and $w_k = y_{kr} \in \mathbb{R}$. Then we want to optimize

$$\min_{v_r} \sum_k \frac{((\Phi^m)_d y'_k - \Psi_d y'_k)^2}{(w_k - (\Phi^m)_d y'_k)^2}$$

where w_k is some real number. In this loss function, we need to choose w_k to optimize the loss function. However, the distribution of the 1's entry varies between rows so we can expect to choose

a set of vectors $\{y_1', y_2', \ldots, y_k'\}$ and some "weights" $\{w_1, w_2, \ldots, w_k\}$. One of the idea for getting the first loss function is to assign all w_ks to be 1, ensuring the condition that the sum of the entries should be approximately 1.

The third loss function is derived from expanding the ℓ_2-norm of a vector. Let m be the coarsening factor and there are $N+1$ time gridpoints. From equation (3.1), suppose we have $\Phi, \Psi \in \mathbb{R}^{n\times n}$ and in each matrix T and $A(v)$, there are r rows of $[-\Phi^m, I]$ and $[-\Psi^m, I]$ respectively (where $r = \frac{N}{m}+1$), then $T, A(v) \in \mathbb{R}^{rn\times rn}$. We let $y_1, y_2, \ldots, y_K \in \mathbb{R}^{rn}$ such that for each $1 \le k \le K$,

$$y_k = \begin{pmatrix} U_{k,1} \\ U_{k,2} \\ \cdots \\ U_{k,r} \end{pmatrix}$$

where each of the U_{kj} is a block of size n. Then

$$Ty_k = \begin{pmatrix} U_{k,1} \\ U_{k,2} - \Phi^m U_{k,1} \\ \cdots \\ U_{k,M} - \Phi^m U_{k,r-1} \end{pmatrix}, \quad A(v)y_k = \begin{pmatrix} U_{k,1} \\ U_{k,2} - \Psi U_{k,1} \\ \cdots \\ U_{k,r} - \Psi U_{k,r-1} \end{pmatrix}$$

and we have

$$\|Ty_k - A(v)y_k\|_{\ell_2}^2 = \left\| \begin{pmatrix} 0 \\ (\Psi - \Phi^m)U_{k,1} \\ \cdots \\ (\Psi - \Phi^m)U_{k,r-1} \end{pmatrix} \right\|_{\ell_2}^2.$$

We see that for any vector $z \in \mathbb{R}^n$, $\|z\|_{\ell_2}^2 = \sum_i z_i^2$. Hence, by generalizing it to block vectors (y_k for example), one can show that

$$\|y_k\|_{\ell_2}^2 = \sum_{i=1}^{M} \|U_{k,i}\|_{\ell_2}^2.$$

Hence,

$$\|Ty_k - A(v)y_k\|_{\ell_2}^2 = \left\| \begin{pmatrix} 0 \\ (\Psi - \Phi^m)U_{k,1} \\ \cdots \\ (\Psi - \Phi^m)U_{k,M-1} \end{pmatrix} \right\|_{\ell_2}^2 = \|0\|^2 + \sum_{i=1}^{r-1} \|(\Psi - \Phi^m)U_{k,i}\|_{\ell_2}^2.$$

$$\|Ty_k\|_{\ell_2}^2 = \left\| \begin{pmatrix} U_{k,1} \\ U_{k,2} - \Phi^m U_{k,1} \\ \cdots \\ U_{k,M} - \Phi^m U_{k,r-1} \end{pmatrix} \right\|_{\ell_2}^2 = \|U_{k,1}\|_{\ell_2}^2 + \sum_{i=1}^{r-1} \|U_{k,i+1} - \Phi^m U_{k,i}\|_{\ell_2}^2.$$

Therefore,

$$\frac{\|Ty_k - A(v)y_k\|_{\ell_2}^2}{\|Ty_k\|_{\ell_2}^2} = \frac{\sum_{i=1}^{r-1} \|(\Psi - \Phi^m)U_{k,i}\|_{\ell_2}^2.}{\|U_{k,1}\|_{\ell_2}^2 + \sum_{i=1}^{r-1} \|U_{k,i+1} - \Phi^m U_{k,i}\|_{\ell_2}^2.}$$

Given a set of testing vectors $w_1, w_2, \ldots, w_K \in \mathbb{R}^n$, if we define $y_k = \begin{pmatrix} w_k \\ w_k \\ \cdots \\ w_k \end{pmatrix} \in \mathbb{R}^{rn}$, then we have the loss function be

$$\mathcal{L}(\Phi, \Psi, m, (w_1, \ldots, w_k)) = \sum_{k=1}^{K} \frac{(r-1)\|(\Psi - \Phi^m)w_k\|_{\ell_2}^2}{\|w_k\|_{\ell_2}^2 + (r-1)\|(I - \Phi^m)w_k\|_{\ell_2}^2}.$$

As the number of time-grid points is large, we expect r to be large enough. Hence, either we fix r to be a large number, i.e. $r = 20$, or we fix the number of time-grid points to be large, i.e. $N = 1000$ and update $r = \frac{N}{r} + 1$ during the training.

Note that if $\lambda_1, \lambda_2, \ldots, \lambda_n$ are the eigenvalues of Φ, then $1 - \lambda_1^m, 1 - \lambda_2^m, \ldots, 1 - \lambda_n^m$ are the eigenvalues of $I - \Phi^m$, with the same set of eigenvectors. Then if all of the components of y_k are the normalized eigenvector with respect to some eigenvalue λ_k, then

$$\frac{\|Ty_k - A(v)y_k\|_{\ell_2}^2}{\|Ty_k\|_{\ell_2}^2} = \frac{(r-1)\|(\Psi - \Phi^m)U_{k,i}\|_{\ell_2}^2.}{1 + (r-1)|1 - \lambda_k^m|^2}$$

Hence, we derive the third loss function

$$\mathcal{L}(\Phi, \Psi, m, (w_1, \ldots, w_k)) = \sum_{k=1}^{K} \frac{(r-1)\|(\Psi - \Phi^m)w_k\|_{\ell_2}^2}{1 + (r-1)|1 - \lambda_k^m|^2}$$

and if we use a constant vector of all 1 (let it be $\mathbf{1}$), then

$$\mathcal{L}(\Phi, \Psi, m, (w_1, \ldots, w_k)) = \sum_{k=1}^{K} \frac{(r-1)\|(\Psi - \Phi^m)w_k\|_{\ell_2}^2}{1 + (r-1)|1 - \lambda_k^m|^2} + \frac{(r-1)\|(\Psi - \Phi^m)\mathbf{1}\|_{\ell_2}^2}{n + (r-1)\|(I - \Phi^m)\mathbf{1}\|_{\ell_2}^2}.$$

From Theorem 3.3 in [5], we want the eigenvalues of Φ^m and Ψ to be close to each other with a similar set of eigenvectors. Let $\{\lambda_1, \lambda_2, \ldots, \lambda_n\}$ be the eigenvalues of Φ. Then for any $m \in \mathbb{N}$, $\{\lambda_1^m, \lambda_2^m, \ldots, \lambda_n^m\}$ are the eigenvalues of Φ^m. We want to study the convergence rate of Φ^m for some large value m. We expect that the closer the eigenvalue λ is to 1, the longer it takes to approximate λ^m. However, the distributions of $|\lambda_i|$ vary between different types of partial differential equations.

For example, consider the diffusion equation $u_t = u_{xx}$, where $u : [0,1] \times [0,1] \to \mathbb{R}$ is some function. After using discretization and use the formula for Toeplitz matrix, one can find explicitly find the eigenvalues are

$$\lambda_i = 1 - 2\beta\left(1 - \cos\left(\frac{i\pi}{n+1}\right)\right), 1 \leq i \leq n.$$

This can be generalized to diffusion-dominated equations, where we expect to have many of the eigenvalues λ_k such that $|\lambda_k| \ll 1$, while there are a few eigenvalues λ_k that $\|\lambda_k\| \approx 1$. Hence, as $\{\lambda_1^m, \lambda_2^m, \ldots, \lambda_n^m\}$ are the eigenvalues of Φ^m, the eigenvalues that are much smaller than 1 will decay with a high speed so we only need to work on those with higher magnitude.

On the other hand, consider the advection equation $u_t + cu_x = 0$. By using Forward-In-Time Forward-In-Space, the eigenvalues are given by

$$\lambda_i = \beta' + 1, \beta' = \frac{c\delta t}{\delta x}, 1 \leq i \leq n.$$

We see that the eigenvalues are closer to 1 than to 0 so approximation for Φ^m will be harder as we deal with eigenvalues that converge slowly. For that reason, the problem of finding the exact eigenvalues and eigenvectors can be computationally expensive so we want to approximate them within some error bound. We can also use the vectors "near" the normalized eigenvectors for training since the matrix Ψ is a linear operator. Therefore, the way Ψ behaves near the eigenvectors must also lie in some small error within the correct answer.

Appendix B

Abbreviations

IPAM. Institute for Pure and Applied Mathematics. An institute of the National Science Foundation, located at UCLA.

RIPS. Research in Industrial Projects for Students. A regular summer program at IPAM, in which teams of undergraduate (or fresh graduate) students participate in sponsored team research projects.

UCLA. The University of California at Los Angeles.

MGRIT. Multigrid Reduction-in-Time.

LLNL. Lawrence Livermore National Laboratory.

CASC. Center of Applied Scientific Computing

HPC. High Performance Computing

NN. Neural Network

FNN. Feedforward Neural Network

ReLU. Rectified Linear Unit

Selected Bibliography Including Cited Works

[1] H. De Sterck, R. D. Falgout, S. Friedhoff, O. A. Krzysik, and S. P. MacLachlan, *scalar non multigrid reduction-in-time and parareal coarse-grid operators for linear advection*, Numerical Linear Algebra with Applications, 28 (2021), p. e2367.

[2] H. De Sterck, R. D. Falgout, and O. A. Krzysik, *Fast multigrid reduction-in-time for advection via modified semi-lagrangian coarse-grid operators*, SIAM Journal on Scientific Computing, 45 (2023), pp. A1890–A1916.

[3] H. De Sterck, R. D. Falgout, O. A. Krzysik, and J. B. Schroder, *Efficient multigrid reduction-in-time for method-of-lines discretizations of linear advection*, Springer Journal of Scientific Computing, 96 (2023).

[4] ——, *Parallel-in-time solution of scalar nonlinear conservation laws*, arXiv preprint arXiv:2401.04936, (2024).

[5] V. A. Dobrev, T. Kolev, N. A. Petersson, and J. B. Schroder, *Two-level convergence theory for multigrid reduction in time (mgrit)*, SIAM Journal on Scientific Computing, 39 (2017), pp. S501–S527.

[6] R. D. Falgout, *An introduction to algebraic multigrid*, Computing in Science & Engineering 8.6, (2006), pp. 24–33.

[7] R. D. Falgout, S. Friedhoff, T. Kolev, S. P. MacLachlan, and J. B. Schroder, *Parallel time integration with multigrid*, SIAM Journal on Scientific Computing, 36 (2014), pp. C635–C661.

[8] R. Huang, K. Chang, H. Huan, R. Li, and Y. Xi, *Reducing operator complexity in algebraic multigrid with machine learning approaches*, arXiv preprint arXiv:2307.07695, (2023).

[9] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning with Applications in Python*, Springer Texts in Statistics, Springer, 2023.

[10] R. J. LeVeque, *Finite difference methods for ordinary and partial differential equations*, Society for Industrial and Applied Mathematics, Jan. 2007.