Research in Industrial Projects for Students

Sponsor

Lawrence Livermore National Laboratory

Midterm Report

Non-Intrusive Parallel-in-Time Solvers for Challenging Problems

Student Members

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

Bryan Li, *University of California, Berkeley*

Daniel Agraz Vallejo, *CETYS University*

Tran Duy Anh Le, *University of Rochester*

Academic Mentor

Dr. Jean-Michel Maldague

Sponsoring Mentors

Dr. Daniel Osei-Kuffuor

Dr. Rob Falgout

Dr. Rui Peng Li

Dr. Wayne Mitchell

Date: July 25, 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 modification of 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 its efficiency by approximating the coarse-grid operator through machine learning approaches.

Acknowledgments

As a research team, we would like to express our gratitude to all those 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 guidance respect 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 for sharing their knowledge and supporting the completion of this project.

Contents

List of Figures

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, using cutting-edge technology to achieve breakthroughs in various fields such as enterprise resilience and counter terrorism, defense and intelligence, and energy security and climate resilience. 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 computer capable of performing 2 exaFLOPS (10^{18} floating point operations per second), making it the fastest computer 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.

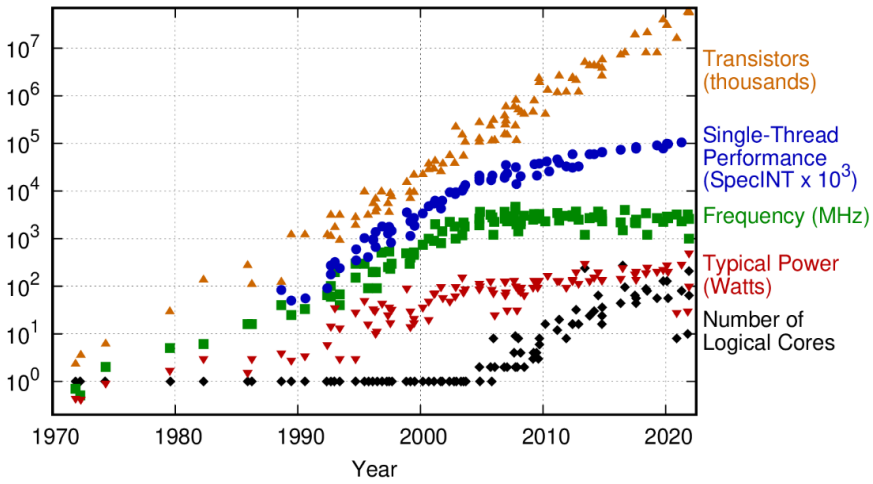

Figure 1.1: Stagnation of Processor Speed. Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K.

This limitation arises the need for algorithmic research aimed on enhancing concurrency to improve the convergence of iterative solutions. In particular, the development of non-intrusive algorithms is a priority, which can generalize to a wide array of problems and various problem sizes without requiring additional information from or modification of the original problems are highly desirable.

1.3 Report Overview

The midterm report starts with Chapter 2, which provides a comprehensive background on the mathematical basis related to the problem, then Chapter 3, covers the current approach to approximate a solution through neural networks, next, we will explain our loss functions ideas and finally we will present our next steps and goals for the project.

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

Let $x_i = hi, \quad h = \frac{1}{N}$ where N is the number of partitions in space and $u_i \approx u(x_i)$. Note that for $x \in [0, 1]$ we have

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

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

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

Applying the approximation for u' we get

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

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

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

Next we can consider the application of this method for the *advection-diffusion equation*, a linear PDE that is used to describe the behaviour of a scalar field such as the concentration of a pollutant or a 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 $\varepsilon \in [0,\infty)$, $a = a(x)$, $f = f(x,t)$. x and t to represent our space and time dimensions, respectively. We divide the interval into evenly-spaced grid nodes. Let $M+1$ be the number of spatial divisions and $N+1$ is the total time divisions. A uniform discretization sets

$$x_i = hi, \quad h = \frac{1}{M}$$

and

$$t_j = kj, \quad k = \frac{1}{N}.$$

Additionally, We label u_i^j as our approximation to $u(x_i, t_j)$.

We can employ a finite difference scheme to obtain the following discretizations:

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

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 $b \in \mathbb{R}^n$. The general form for relaxation using splitting is to let

$$A = M + N$$

Then the 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 relevant relaxation methods to this project are *Jacobi* and *Gauss-Seidel*. For these methods, lets consider $A = L + D + U$ where L is lower triangular, D is diagonal, and U is upper triangular.

In *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 *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 does 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 $\mathbf{e}_{k+1}$ as

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

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. $I - M^{-1}A = PDP^{-1}$, $(I - M^{-1}A)^k = PD^kP^{-1}$ only converge to 0 if all eigenvalues of $I - M^{-1}A$ are less than 1. If this is true, the error converges to 0 and thus for any initial guess x_0, we have that $x_k \to A^{-1}b$ as $k \to \infty$.

However one drawback for relaxation methods is that they are better at targeting oscillatory errors compared to smooth errors, which means they take more iterations to correct smooth errors. This ultimately leads to the employment of Multigrid methods which attacks this problem in relaxation methods.

2.3 Multigrid Methods

The 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, smooth error is characterized to have small eigenmodes. Assuming A 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.

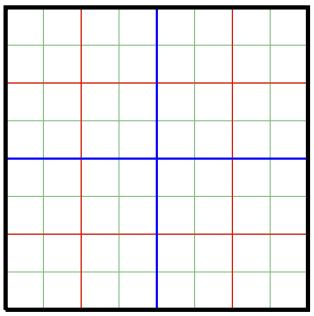

Figure 2.1: Multigrid Hierarchy

For some linear system $Au = f$ with and $u^* = A^{-1}f$ 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 $u_1 = u_0 + x$.

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.

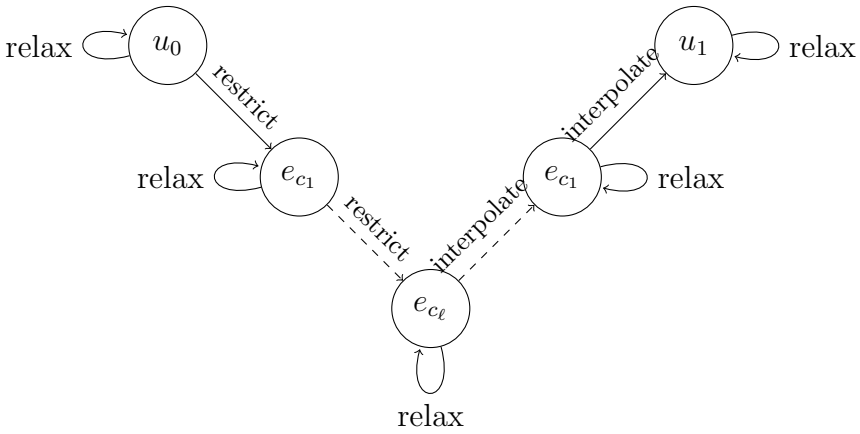

Figure 2.2: Multigrid V-Cycle

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 [6].

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

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

where $u_j^i \approx u(x_j, t_i)$.

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

$$\mathbf{u}_i = \Phi \mathbf{u}_{i-1} + \delta t \mathbf{f}_{i-1}$$

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 has the stencil

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

Using the partition in Figure 2.3, we let $t_i = i\delta t$ 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

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

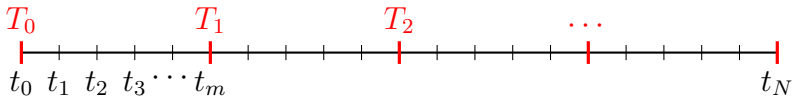

Figure 2.3: Coarse and fine time grid

Φ^m is known as the ideal *coarse grid operator*. However, as matrices get denser being raise to a power, Φ^m is 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

The Model

3.1 Neural Network Architecture

We are using a (feedforward) neural network. The input is the stencil $\phi \in \mathbb{R}^3$ along with the exponent $m \in \mathbb{N}$ for the matrix Φ and the output is a stencil $\psi \in \mathbb{R}^3$ for the matrix Ψ, which approximates Φ^m.

3.2 The Loss Function

Our goal is to find a "spectrally equivalent" but sparser matrix to approximate the ideal coarse-grid timestepping operator

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

with a matrix of the form

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

where Ψ is a sparse (e.g. tridiagonal) generated by a stencil v.

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. 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. Initially all the vectors have the same weight when accounting the loss function. However, we found that directly using this as our loss function led to poor convergence as it placed a disproportional emphasis on vectors where $\|Ty_k\| \ll \|y_k\|$.

Our second idea for a loss function comes from Ru Huang et. al. [7], where we first compute the largest K (normalized) eigenvectors v_k of the ideal operator Φ^m, and then construct the loss function

$$\mathcal{L}_2 = \sum_{k=1}^{K} \|\Phi^m v_k - \Psi v_k\|^2. \tag{3.2}$$

3.2.1 Analyses of loss functions

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 M rows of $[-\Phi^m, I]$ and $[-\Psi^m, I]$ respectively, then $T, A(v) \in \mathbb{R}^{Mn\times Mn}$. We let $y_1, y_2, \ldots, y_K \in \mathbb{R}^{Mn}$ such that for each $1 \le k \le K$,

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

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

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

and we have

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

We see that by the triangle inequality,

$$\|Ty_k - A(v)y_k\|_{\ell_2} \le \sum_{i=1}^{M-1} \|(\Psi - \Phi^m)U_{k,i}\|_{\ell_2}$$

and by using Cauchy-Schwarz inequality and triangle inequality,

$$\begin{aligned} \|Ty_k\|_{\ell_2} &\ge \frac{1}{\sqrt{M}} \left(\|U_{k,1}\|_{\ell_2} + \sum_{i=1}^{M-1} \|U_{k,i+1} - \Phi^m U_{k,i}\|_{\ell_2} \right) \\ &\ge \frac{1}{\sqrt{M}} \left(\|U_{k,1}\|_{\ell_2} + \sum_{i=1}^{M-1} |(\|U_{k,i+1}\|_{\ell_2} - \|\Phi^m U_{k,i}\|_{\ell_2})| \right). \end{aligned}$$

Hence

$$\mathcal{L}_1 \leq \sqrt{M}\sum_k w_k \frac{\sum_{i=1}^{M-1}\|(\Psi-\Phi^m)U_{k,i}\|_{\ell_2}}{\|U_{k,1}\|_{\ell_2}+\sum_{i=1}^{M-1}|(\|U_{k,i+1}\|_{\ell_2}-\|\Phi^m U_{k,i}\|_{\ell_2})|}.$$

If we take each of the components $U_{k,i}$ to be a normalized eigenvector of $\lambda_{k,i}$ with respect to matrix Φ, then we can simplify

$$\mathcal{L}_1 \leq \sqrt{M}\sum_k w_k \frac{\sum_{i=1}^{M-1}\|(\Psi-\Phi^m)U_{k,i}\|_{\ell_2}}{1+\sum_{i=1}^{M-1}||\lambda_{k,i}^m|-1|}.$$

Then we get a loss function similar to the second loss function.

One other idea is to look at a singular row of the matrix T and $A(v)$ as the row of Ψ is generated by a simple stencil. Note that

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

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

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

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

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

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 $\{y_1', y_2', \dots, y_k'\}$ and some "weights" $\{w_1, w_2, \dots, w_k\}$.

Let $\{\lambda_1, \lambda_2, \dots, \lambda_n\}$ be the eigenvalues of Φ. Then for any $m \in \mathbb{N}$, $\{\lambda_1^m, \lambda_2^m, \dots, \lambda_n^m\}$ 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 converge 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.

Hence, we want to develop a loss function that works well with the large eigenvalues. However, 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.

3.3 Results

3.3.1 Testing Results with PyMGRIT

We have created a class, `Heat1DNN` that uses the output from the our trained neural network to generate the coarse-grid operator for the problem.

Testing on a neural network with 5 hidden layers of 96 nodes each, using the second loss function, trained on the stencils resulting from the discretizations of the heat equation with $\delta x = 1.2$, $\delta t = 1/2^i$ for $i = 3, \ldots, 11$, $m = 1, \ldots, 8$, our results are currently worse than rediscretization. However, this is because rediscretization is already known to be very effective for diffusion-dominated problems.

Testing on the stencil resulting from the discretization choosing $\delta x = 1/16$, and $\delta t = 1/4096$, with coarsening factor 4, we see that using the rediscretization gives us the stencil $\begin{bmatrix}0.25 & 0.5 & 0.25\end{bmatrix}$ and our neural network gives us the stencil $\begin{bmatrix}0.0561 & 0.8222 & 0.0618\end{bmatrix}$.

Evaluating the matrix norm of a matrix of dimension 500 generated by these two stencils, we get an error of 3.75 for the rediscretization and 5.77 for the trained Ψ operator. However, in empirical tests with PyMGRIT using a 2-level multigrid, using this Ψ does in fact result in (slow) convergence. It took around 22 V-cycles to reach a 0.02 residual error using the trained operator, whereas it only took around 5 V-cycles to reach the same error using the rediscretized operator.

We hope to fine-tune the neural network to have more promising results in the future.

3.4 Next Steps

3.4.1 Generalization

The application of this method to the 1D diffusion equation is only a proof-of-concept. We hope to generalize this method to other types of equations, with 1D advection being our next target.

However, a tridiagonal stencil for Ψ may not work well for all problems. 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 [7], although this would require modifying the structure of our output vector.

3.4.2 Improving the neural network

Something we can experiment with further is modifying different parameters in the neural network. The number of hidden layers, number of nodes, training rate, changing training rate as the model learns, number of epochs, etc.

3.4.3 Using bootstrapping techniques

Currently we are improving the loss function by using 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.

3.4.4 Applying Offline and Online learning techniques

At the moment, the model's training dataset consists of a set of five stencils, where each one has a coarsening factor m appended as last element, after forward propagation is executed, a group of test vectors y_k are generated to compute the loss function after a number of epochs. Overall, this training process follows an Offline approach because we are training the neural network with

stencils that have fixed β values at once, therefore we aim to expand our training approach to apply online learning techniques that are able of updating the weights of the neural network based on new stencils, thus, we can increment the learning capability of generalizing to available stencils.

Chapter 4

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

References

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

[6] 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.

[7] R. Huang, K. Chang, H. Huan, R. Li, and Y. Xi, *Reducing operator complexity in algebraic multigrid with machine learning approaches*, 2023.

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