Distribution Agreement

In presenting this thesis as a partial fulfillment of the requirements for a degree from Emory University, I hereby grant to Emory University and its agents the non-exclusive license to archive, make accessible, and display my thesis in whole or in part in all forms of media, now or hereafter now, including display on the World Wide Web. I understand that I may select some access restrictions as part of the online submission of this thesis. I retain all ownership rights to the copyright of the thesis. I also retain the right to use in future works (such as articles or books) all or part of this thesis.

Signature: Kai Chang

Kai Chang March 23, 2023

Reducing Operator Complexity in Algebraic Multigrid with Machine Learning Approaches

By

Kai Chang

Yuanzhe Xi, Ph.D.
Advisor

Mathematics

Yuanzhe Xi, Ph.D.
Advisor

James Nagy, Ph.D.
Committee Member

Lars Ruthotto, Ph.D.
Committee Member

Daniel Weissman, Ph.D.
Committee Member

2023

Reducing Operator Complexity in Algebraic Multigrid with Machine Learning Approaches

By

Kai Chang

Yuanzhe Xi, Ph.D.
Advisor

An abstract of
a thesis submitted to the Faculty of Emory College of Arts and Sciences of
Emory University in partial fulfillment
of the requirements of the degree of
Bachelor of Science with Honors

Mathematics

2023

Abstract

Reducing Operator Complexity in Algebraic Multigrid
with Machine Learning Approaches
By Kai Chang

Large-scale sparse linear systems arisen from discretizing partial differential equations (PDEs) appear ubiquitously in science and engineering. Algebraic multigrid methods (AMG) are among the most efficient algorithms for numerically solving such systems. However, AMG often suffers from the problem of increasing density (i.e., increased number of non-zero entries in the linear operator) as a parallel solver, which induces an excessively high computational cost for tasks, such as data communications, other than the actual computations. How to alleviate this issue to improve the efficiency and scalability of AMG remains challenging and crucial.

In this thesis, we try to tackle this problem by leveraging modern data-driven methods. We propose a multilevel deep learning method to sparsify all the discretized coarse-grid operators in the hierarchy of multigrid methods when solving parametric PDEs, i.e. PDEs dependent on some parameters. The method succeeds in reducing the number of non-zero entries in all the discretized operators by at least 44%. Strictly following the multigrid convergence theory, the method does not trade sparsity with the convergence behaviour of multigrid methods. Another key feature of the method is its capability of generalizing to not only problems of larger sizes, but also PDEs with different parameters from those in the training set. This allows the trained sparsifiers to be used in various settings aside from the training instances, thereby enhancing the practical utility of the algorithm. We provide extensive numerical experiments on challenging anisotropic rotated Laplacian problems and linear elasticity problems to illustrate its superior performance.

Reducing Operator Complexity in Algebraic Multigrid with Machine Learning Approaches

By

Kai Chang

Yuanzhe Xi, Ph.D.
Advisor

A thesis submitted to the Faculty of Emory College of Arts and Sciences
of Emory University in partial fulfillment
of the requirements of the degree of
Bachelor of Science with Honors

Mathematics

2023

Acknowledgments

I hope to express my sincere gratitude to Prof. Yuanzhe Xi, my research advisor for the past two years—from him, I have learned much more than I originally expected when I started this research journey. I thank him for always being responsive and enthusiastic whenever I had questions (and I truly had a lot of questions) and for teaching me so many things that I could otherwise unlikely learn in any class.

I am grateful to Prof. Lars Ruthotto, whom I am lucky to have known since my sophomore year. Lars has always been so supportive whenever I am in need. He generously involved me in his research training seminar in my junior year, from which I learned plenty of research skills that are consistently helpful to my academic life. I thank him for his consistent encouragements, for his timely responses and many helpful discussions, and for his jokes that have made my day for plenty of times.

My equal gratitude extends to Prof. James Nagy. Jim has been my go-to person whenever I have important decisions to make for my academic career. I thank him for always being enthusiastic about talking to me whenever I step into his office and for his support and consistent willing to help me.

I am also thankful to Prof. Daniel Weissman for being on my committee and allowing me to enroll in his *Mathematical Physics* class. I am grateful to him for answering all of my physics and math questions without weariness.

Last but not least, a huge thank you goes to my parents, my girlfriend Jennifer, my best friend Zephyr, and all the other friends who have supported and encouraged me. You guys are really what carried me through the past four years, emotionally and financially. I am forever indebted to your support, love, and kindness.

To all those who have been mentioned, I cannot possibly be where I am without any one of you. Thank you.

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

We consider the problem of solving a linear system of the form

$$\boldsymbol{A}_\beta \boldsymbol{u} = \boldsymbol{f}, \tag{1.1}$$

where $\boldsymbol{A}_\beta \in \mathbb{R}^{N\times N}$ is a symmetric positive definite (SPD) linear operator obtained by discretizing certain parameteric partial differential equations (PDEs) dependent on some parameters $\beta \in \mathbb{R}^p$, $\boldsymbol{u} \in \mathbb{R}^N$ is the solution, and $\boldsymbol{f} \in \mathbb{R}^N$ is some forcing term.

Over the past several decades, a tremendous amount of computational mathematical research has been devoted to the development of fast algorithms for solving (SPD) linear systems due to their ubiquitous appearances in various areas of science and engineering. There are two major types of linear solvers: direct solvers and iterative solvers; and they are intrinsically different from each other.

Direct solvers are based on Gaussian eliminations and matrix LU decompositions. They typically take advantage of the structure of a matrix, such as its sparsity or bandedness, to identify an efficient way of doing the LU decomposition. After the LU factors are obtained, solving the system will be rather easy as the complexity is only $O(N^2)$. Direct methods are accurate when they are able to finish within a reasonable

amount of time, since they attempt to find the true solution. In contrast, iterative methods are based on the idea of finding some reasonably good approximations to the true solution in some vector space. They thus are often not as accurate as direct solvers, but are "accurate enough" for practical usage.

Nowadays, although state-of-the-art (SOTA) direct solvers can achieve quite remarkable performance for certain problems, iterative methods have become more and more favorable over direct methods, and there are many good reasons behind this. We list two major ones here. First, direct solvers can be extremely slow when they are applied to hard problems such as 3-dimensional problems and even 2-dimensional problems with many degrees of freedom per point. The memory and computational requirements of those problems make them unlikely to be handled by direct solvers. Second, many of the most efficient iterative methods depend only on matrix-vector products and are thus easier to implement efficiently. Such a feature makes iterative solvers more user-friendly in practice than direct solvers.

Among all those iterative methods, Algebraic Multigrid (AMG) is one of the most efficient and scalable algorithms for solving (1.1). For systems that arise from discretizations of elliptic-type PDEs, classical AMG methods can often show the optimal linear computational complexities. Nevertheless, improving the efficiency of AMG is still actively being researched and remains challenging by and large. The overall efficiency of AMG relies on the several key operators of AMG and the interplay between them. There has been a line of works to design better AMG elements by leveraging data-driven methods. [20, 16, 13] deal with learning better prolongation (a.k.a. interpolation) operators. [30] takes advantage of techniques in deep reinforcement learning to better tackle the problem of optimally partitioning the fine level nodes during coarsening. Both [14] and [18] focus on the problem of learning better smoothers (relaxation matrices). In [14], smoothers are directly parameterized by multi-layer CNNs while [18] focuses on learning to assign better weights to the

weighted-Jacobi-type smoothers.

In this work, we tackle the problem of increasing complexities of coarse-grid operators, computed as Galerkin products, along the levels of AMG hierarchies. This issue has adverse effects of the overall performance of AMG as it can increase the computational cost of applying the operators in deeper levels, can impair the effectiveness and robustness of other AMG components such as coarsening algorithms and interpolating algorithms, and can make the communications more expensive in distributed computing environments, to name a few. Traditional "Non-Galerkin" approaches to reduce the operator complexity employs sparsification of the Galerkin product by, for instance, removing weakly connected nodes, minimizing spectral in-equivalence of stencils, and sparsifying smooth aggregations, [10, 32, 3].

We take on a different perspective by using deep learning methods to find sparsified operators on each level. Our contributions are summarized as follows:

- we developed a multi-level algorithm based on the deep learning methods to sparsify all the coarse-grid operators in the multigrid hierarchy;
- our method succeeds in reducing the operator density while maintaining the convergence behavior of the employed multigrid method ;
- our models, once trained, work for a class of parametric PDEs with the parameters following certain probability distributions;
- the sparsifier on each level can be trained in parallel once the training data is prepared;
- and the averaged number of non-zero entries per row in the coarse-grid operators can be chosen by the user, despite having to be greater than a certain amount.

The rest of the thesis is organized as follows. In Chapter 2, we provide necessary backgrounds on discretizations of PDEs and their stencil representations. In Chapter

3, some relevant iterative methods for solving linear systems are reviewed. Followed by introducing the problem of increasing operator density (and therefore complexity) in Chapter 4, which is the primary motivation for our work, we elaborate on our proposed machine-learning-based coarse-grid-operator sparsification method in Chapter 5. We present the numerical experiments and results in Chapter 6, discuss the limitation of our method and possible future works in Chapter 7, and conclude the work in Chapter 8.

Chapter 2

Preliminaries

In this chapter, we provide some preliminary backgrounds on numerical PDEs and deep neural networks. In particular, we discuss a multi-headed neural network inspired by the attention mechanism in Section 2.3, which will be used later in our algorithm. For a more comprehensive introduction to numerical PDEs and deep learning, see [24, 25, 31] and [11, 23] respectively.

2.1 Discretizations of PDEs and Stencil Representations

The solution of a PDE, more often than not, cannot be expressed with an analytical formula. Therefore, it is necessary to approximate the solutions numerically on a computing device. In order to do so, the first step is to transform the equation into something that a computer can understand. Mathematically, this process is known as *discretization*, which is to artificially place some points in the domain on which the equation of interest is defined. After discretization, the equation is transformed into a linear system, which is understandable by a computer. The approximate solution can then be obtained from there. In this chapter, we discuss briefly the process of

discretization.

We use a classical example to introduce some necessary background on discretizations of PDEs. Consider a 2-dimensional Poisson's Equation

$$-\Delta u = f$$

defined on the unit box $\mathcal{D} := [0,1] \times [0,1] \in \mathbb{R}^2$ and subject to certain boundary conditions. A classical approach to discretize the equation is through the second-order finite difference scheme with equal spatial step on the $x-$ and $y-$axis, which reads

$$-\Delta u(x_i, y_j) \approx \frac{-u_{i+1,j} - u_{i,j+1} + 4u_{i,j} - u_{i-1,j} - u_{i,j-1}}{h^2} \tag{2.1}$$

where

$$\begin{cases} h = \frac{1}{N} & \\ x_i = ih, & 0 \le i \le N \\ y_j = jh, & 0 \le j \le N \\ u_{i,j} = u(x_i, y_j), & \forall i, j \end{cases}.$$

The points $\{(x_i, y_j)\}_{i,j=0}^{N}$ are known as the discretization points. Note that it is not hard to verify through Taylor's expansion that

$$u_{i,j} = \frac{-u_{i+1,j} - u_{i,j+1} + 4u_{i,j} - u_{i-1,j} - u_{i,j-1}}{h^2} + O(h^2),$$

and that is why the method is second-order — the truncation error decreases quadratically with the fineness of the discretization points.

We plug (2.1) back into the original equation and can then obtain

$$\frac{-u_{i+1,j} - u_{i,j+1} + 4u_{i,j} - u_{i-1,j} - u_{i,j-1}}{h^2} = f_{i,j}$$

where $f_{i,j} = f(x_i, y_j)$. If we list up the equations for all i and j, we may write it out neatly as a linear system

$$\boldsymbol{A}\boldsymbol{u} = \boldsymbol{f} \tag{2.2}$$

where

$$\boldsymbol{A} = \frac{1}{h^2} \cdot \begin{bmatrix} 4 & -1 & 0 & -1 & 0 & \dots & 0 \\ -1 & 4 & -1 & 0 & -1 & \ddots & \vdots \\ 0 & -1 & 4 & 0 & 0 & \ddots & 0 \\ -1 & 0 & 0 & 4 & -1 & \ddots & -1 \\ 0 & -1 & 0 & -1 & 4 & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & -1 & 0 & 0 & 4 \end{bmatrix} \in \mathbb{R}^{(N+1)^2 \times (N+1)^2},$$

$$\boldsymbol{u} = [\boldsymbol{u}_0^T, \boldsymbol{u}_1^T, ..., \boldsymbol{u}_{N-1}^T, \boldsymbol{u}_N^T]^T \in \mathbb{R}^{(N+1)^2},$$

$$\boldsymbol{u}_i = [u_{i,0}, u_{i,1}, ..., u_{i,N-1}, u_{i,N}]^T \in \mathbb{R}^{(N+1)}, \quad 0 \le i \le n,$$

$$\boldsymbol{f} = [\boldsymbol{f}_0^T, \boldsymbol{f}_1^T, ..., \boldsymbol{f}_{N-1}^T, \boldsymbol{f}_N^T]^T \in \mathbb{R}^{(N+1)^2},$$

and

$$\boldsymbol{f}_i = [f_{i,0}, f_{i,1}, ..., f_{i,N-1}, f_{i,N}]^T \in \mathbb{R}^{(N+1)}, \quad 0 \le i \le n.$$

There is a neater representation for the matrix $\boldsymbol{A}$, known as the stencil representation. It is typically used to represent a family of matrices out of a specific discretization scheme regardless of the number of discretization points, or the mesh size. The idea of using such a representation is as follows. In the discretization scheme (2.1), no matter how i or j changes, the coefficients in front of the solution at the discretization points, i.e. $u_{i+1,j}, u_{i,j+1}, u_{i,j}, u_{i-1,j}$, and, $u_{i,j-1}$ remain the same. Therefore, what really matters here are the coefficients instead of the number of

discretization points. We thus use the matrix

$$\mathcal{A} = \begin{bmatrix} & -1 & \\ -1 & 4 & -1 \\ & -1 & \end{bmatrix}$$

to represent the family of matrices defined by the same discretization scheme (2.1) regardless of the mesh size. This notation allows us to represent a family of matrices which will come in handy later in the thesis.

2.2 Deep Neural Networks

From a mathematical point of view, a deep neural network is simply a composition of some non-linear and affine functions. Consider a data matrix $\boldsymbol{X} \in \mathbb{R}^{d_0 \times n}$ where each column of $\boldsymbol{X}$ represents a data point. $\boldsymbol{X}$ is what will be inputted into a neural network. To demonstrate the whole architecture, we start with the first layer, which consists of an affine transformation

$$\boldsymbol{H}^{[1]} \leftarrow \boldsymbol{W}^{[1]}\boldsymbol{X} + \boldsymbol{b}^{[1]}\mathbf{1}^T$$

followed by a non-linear activation function $\sigma : \mathbb{R} \to \mathbb{R}$ applied point-wisely on $\boldsymbol{H}^{[1]}$,

$$\boldsymbol{Z}^{[1]} \leftarrow \sigma\left(\boldsymbol{H}^{[1]}\right),$$

where $\boldsymbol{W}^{[1]} \in \mathbb{R}^{d_1 \times d_0}$ is the weight matrix in layer 1, $\boldsymbol{b}^{[l]} \in \mathbb{R}^{d_1 \times 1}$ is the bias vector in layer 1, $\mathbf{1} \in \mathbb{R}^{n \times 1}$ is an all-one vector, and $\boldsymbol{Z}^{[1]} \in \mathbb{R}^{d_1 \times n}$ denotes the output of the first

layer. The rest is essentially a recursion of the first layer: in the l^{th} layer, we have

$$\boldsymbol{H}^{[l]} \leftarrow \boldsymbol{W}^{[l]}\boldsymbol{Z}^{[l-1]} + \boldsymbol{b}^{[l]}\mathbf{1}^T,$$
$$\boldsymbol{Z}^{[l]} \leftarrow \sigma\left(\boldsymbol{H}^{[l]}\right)$$

where $\boldsymbol{W}^{[l]} \in \mathbb{R}^{d_l \times d_{l-1}}$ is the weight matrix in the l^{th} layer, $\boldsymbol{b} \in \mathbb{R}^{d_l \times 1}$ is the bias vector in the l^{th} layer, $\boldsymbol{Z}^{[l]} \in \mathbb{R}^{d_l \times n}$ denotes the output of the l^{th} layer, $1 \leq l \leq L$, and $\boldsymbol{Z}^{[0]} := \boldsymbol{X}$.

Some typical choices of the activation function σ are

- the sigmoid function $\sigma(x) = \dfrac{1}{1+e^{-x}}$,
- the rectified linear unit function $ReLU(x) = max\{0, x\}$,
- and the hyperbolic tangent function $tanh(x) = \dfrac{2}{1+e^{-2x}} - 1$.

Another function that is often used in classification tasks is the $softmax$ function. Consider an input vector

$$\boldsymbol{x} := [x_1, x_2, \ldots, x_d]^T \in \mathbb{R}^d.$$

The $softmax$ function applied on the vector is defined as

$$softmax(\boldsymbol{x}) := [y_1, y_2, \ldots, y_d]^T$$

where

$$y_j := \frac{e^{y_j}}{\sum_{i=1}^{d} e^{y_i}}.$$

2.3 A Multi-Headed Neural Network

Later in our algorithmic development, we will use a so-called multi-headed neural network as our network architecture. We name it as such because it is inspired by the multi-headed attention mechanism proposed in [34]. It is an empirical observation that such a network works much better than the vanilla neural networks for our task. The network is described as follows.

In the l^{th} layer, we first copy the input $\boldsymbol{Z}^{[l-1]}$ for r times, and stack them up as a new matrix:

$$\boldsymbol{Z}^{[l-1]} \leftarrow \begin{bmatrix} \boldsymbol{Z}^{[l-1]} \\ \boldsymbol{Z}^{[l-1]} \\ \vdots \\ \boldsymbol{Z}^{[l-1]} \end{bmatrix} \in \mathbb{R}^{r \cdot d_{l-1} \times n}.$$

We then apply the affine transformation

$$\boldsymbol{H}^{[l]} \leftarrow \boldsymbol{W}^{[l]} \boldsymbol{Z}^{[l-1]} + \boldsymbol{B}^{[l]},$$

while this time the weight matrix $\boldsymbol{W}^{[l]}$ is defined as

$$\boldsymbol{W}^{[l]} := \begin{bmatrix} \boldsymbol{W}^{[l,1]} & 0 & 0 & 0 \\ 0 & \boldsymbol{W}^{[l,2]} & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \boldsymbol{W}^{[l,r]} \end{bmatrix}$$

where $\boldsymbol{W}^{[l,j]} \in \mathbb{R}^{d_l \times d_{l-1}}$ for $1 \leq j \leq r$, and

$$\boldsymbol{B}^{[l]} := \begin{bmatrix} \boldsymbol{b}^{[l,1]}\mathbf{1}^T \\ \boldsymbol{b}^{[l,2]}\mathbf{1}^T \\ \vdots \\ \boldsymbol{b}^{[l,r]}\mathbf{1}^T \end{bmatrix}$$

where $\boldsymbol{b}^{[l,j]} \in \mathbb{R}^{d_l \times 1}$ for $1 \leq j \leq r$ and $\mathbf{1} \in \mathbb{R}^{n \times 1}$ is an all-one vector. Followed by this, we do

$$\boldsymbol{Z}^{[l]} \leftarrow \sigma\left(\boldsymbol{H}^{[l]}\right)$$

and

$$\boldsymbol{Z}^{[l]} \leftarrow \boldsymbol{R}^{[l]}\boldsymbol{Z}^{[l]}$$

where $\boldsymbol{R}^{[l]} \in \mathbb{R}^{d_l \times r \cdot d_l}$ maps everything back to the normal dimension. We will refer to a network with layers like this as a multi-headed neural network.

Chapter 3

Iterative Methods for Solving Linear Systems

In this chapter, we review some of the iterative methods relevant to this work. We first talk about relaxation methods, also known as stationary iterative methods, since they form a crucial component of the multigrid methods. After that, we briefly review the Generalized Minimal Residual (GMRES) method, which is one of the most well-known "general-purpose" iterative methods developed so far. We include GMRES here because it is often used as a top-level accelerator in multigrid methods, and it is used in our numerical experiments later. Last but not least, we give a detailed introduction on multigrid methods, our main focus in this work.

We remark that our discussions of the algorithms are by no means complete. We refer the audience to [27, 33, 12, 19] for a more comprehensive introduction of iterative methods and to [8, 29, 36, 9] for multigrid methods.

3.1 Relaxation Methods

3.1.1 The Basic Form

Recall that our goal is to solve a linear system of the form

$$\boldsymbol{A}_\beta \boldsymbol{u} = \boldsymbol{f} \tag{3.1}$$

where $\boldsymbol{A}_\beta := [a_{ij}] \in \mathbb{R}^{n\times n}$ is an SPD matrix, $\beta \in \mathbb{R}^p$ is some parameter, and $\boldsymbol{u}, \boldsymbol{f} \in \mathbb{R}^n$ are column vectors. Note that we keep the β here entirely for the consistency of notations. Here we think about (3.1) as a linear system, and the parameter does not matter to the algorithms introduced in this whole chapter.

A relaxation method approximates $\boldsymbol{u}$ by taking iterations of the form

$$\boldsymbol{u}_{k+1} = (\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A}_\beta)\boldsymbol{u}_k + \boldsymbol{M}^{-1}\boldsymbol{f} \tag{3.2}$$

where $\boldsymbol{M}$ is a so-called relaxation matrix. If we let $\boldsymbol{G} = \boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A}_\beta$, we may rewrite (3.2) into

$$\boldsymbol{u}_{k+1} = \boldsymbol{G}\boldsymbol{u}_k + \boldsymbol{M}^{-1}\boldsymbol{f} \tag{3.3}$$

where $\boldsymbol{G}$ is the iteration matrix associated with (3.2).

3.1.2 Examples of Relaxation Methods

Suppose $\boldsymbol{A}_\beta$ is splitted into three matrices: $\boldsymbol{A}_\beta = \boldsymbol{D} - \boldsymbol{L} - \boldsymbol{U}$ where

$$\boldsymbol{D} = \begin{bmatrix} a_{11} & 0 & \dots & 0 \\ 0 & a_{22} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & a_{nn} \end{bmatrix}, -\boldsymbol{L} = \begin{bmatrix} 0 & 0 & \dots & 0 \\ a_{21} & 0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ a_{n1} & \dots & a_{n,n-1} & 0 \end{bmatrix}, \text{ and } -\boldsymbol{U} = \begin{bmatrix} 0 & a_{12} & \dots & a_{1n} \\ 0 & 0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & a_{n-1,n} \\ 0 & \dots & 0 & 0 \end{bmatrix}.$$

When $\boldsymbol{M} = \boldsymbol{D}$ in (3.2), the method is called the Jacobi method; when $\boldsymbol{M} = \omega\boldsymbol{D}$ where $0 \leq \omega \leq 1$, it is the weighted Jacobi method; when $\boldsymbol{M} = \boldsymbol{D} - \boldsymbol{L}$, it becomes the forward Gauss-Seidel method; the backward Gauss-Seidel method corresponds to the case where $\boldsymbol{M} = \boldsymbol{D} - \boldsymbol{U}$; and when $\boldsymbol{M} = \frac{1}{\omega}\boldsymbol{D} - \boldsymbol{L}$ where $0 \leq \omega \leq 1$, we are looking at the Successive Over Relaxation method.

Let us denote the residual of the k^{th} iterate as $\boldsymbol{r}_k$; that is,

$$\boldsymbol{r}_k := \boldsymbol{f} - \boldsymbol{A}_\beta \boldsymbol{u}_k. \tag{3.4}$$

We denote the true solution to (3.1) as $\boldsymbol{u}_*$; i.e. $\boldsymbol{A}_\beta \boldsymbol{u}_* = \boldsymbol{f}$. When applying a relaxation method, the approximation error in the $k + 1^{th}$ iterate is defined as

$$\begin{aligned} \boldsymbol{e}_{k+1} &:= \boldsymbol{u}_* - \boldsymbol{u}_{k+1} \\ &= \boldsymbol{u}_* - \boldsymbol{G}\boldsymbol{u}_k - \boldsymbol{M}^{-1}\boldsymbol{f} \\ &= \boldsymbol{u}_* - \left(\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A}_\beta\right)\boldsymbol{u}_k - \boldsymbol{M}^{-1}\boldsymbol{f} \\ &= \boldsymbol{u}_* - \boldsymbol{u}_k + \boldsymbol{M}^{-1}\boldsymbol{A}_\beta\boldsymbol{u}_k - \boldsymbol{M}^{-1}\boldsymbol{f} \\ &= \boldsymbol{e}_k - \boldsymbol{M}^{-1}\left(\boldsymbol{f} - \boldsymbol{A}_\beta\boldsymbol{u}_k\right) \\ &= \boldsymbol{e}_k - \boldsymbol{M}^{-1}\boldsymbol{r}_k \\ &= (\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A}_\beta)\boldsymbol{e}_k = \boldsymbol{G}\boldsymbol{e}_k. \end{aligned} \tag{3.5}$$

Equation (3.5) provides a simple convergence criteria for relaxation methods: a re-

laxation method is guaranteed to converge when the spectral radius of the iteration matrix $\boldsymbol{G}$ is less than 1. That is, when

$$\rho(\boldsymbol{G}) < 1.$$

Also note that we employed the fact that

$$\boldsymbol{r}_k = \boldsymbol{A}_\beta \boldsymbol{e}_k \tag{3.6}$$

for all k when deriving (3.5).

3.2 Krylov Subspace Methods and the Generalized Minimal Residual (GMRES) Method

The Generalized Minimal Residual method (GMRES) [26] belongs to a larger class of algorithms called the Krylov subspace algorithms. In this section, we give a high-level introduction to why Krylov subspace methods work and the idea behind GMRES.

Let us first define what a Krylov subspace is.

Definition. A Krylov subspace, denoted by $\mathcal{K}_m(\boldsymbol{A}, \boldsymbol{f})$ where m is an integer, $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ is a square matrix, and $\boldsymbol{f}$ is a starting vector, has the form

$$\mathcal{K}_m(\boldsymbol{A}, \boldsymbol{f}) = span\{\boldsymbol{f}, \boldsymbol{A}\boldsymbol{f}, \ldots, \boldsymbol{A}^{m-1}\boldsymbol{f}\}.$$

In the m^{th} iteration of a Krylov subspace algorithm, an approximate solution $\boldsymbol{u}_m$ to the linear system

$$\boldsymbol{A}\boldsymbol{u} = \boldsymbol{f}$$

is searched in the space $\mathcal{K}_m(\boldsymbol{A}, \boldsymbol{f})$, given that the initial guess $\boldsymbol{u}_0 = 0$. To demonstrate

why this could be (and actually is) a good idea, we consider the case when $\boldsymbol{A}$ is non-singular. Then the obvious closed form solution to the system is

$$\boldsymbol{u}_* = \boldsymbol{A}^{-1}\boldsymbol{f}.$$

Now, by the Cayley-Hamilton Theorem [28], the charateristic polynomial of $\boldsymbol{A}$,

$$c_{\boldsymbol{A}}(\lambda) = det(\boldsymbol{A} - \lambda\boldsymbol{I}) := \sum_{j=0}^{n} a_j \lambda^j$$

has the property of

$$c_{\boldsymbol{A}}(\boldsymbol{A}) = 0.$$

Note that since $\boldsymbol{A}$ is non-singular,

$$c_{\boldsymbol{A}}(0) = det(\boldsymbol{A}) = a_0 \neq 0.$$

We thus have

$$\sum_{j=1}^{n} a_j \boldsymbol{A}^j = -a_0 \boldsymbol{I}$$

where $a_0 \neq 0$, which implies that

$$\boldsymbol{A}\left(\sum_{j=0}^{n-1} -\frac{a_j}{a_0}\boldsymbol{A}^j\right) = \boldsymbol{I}.$$

Therefor, $\boldsymbol{A}^{-1}$ can be expressed as $p(\boldsymbol{A})$ where $p(x)$ is a polynomial with a degree of at most $n-1$. This justifies that searching a solution in the Krylov subspace is reasonable.

GMRES is one of the most robust Krylov subspace method for solving a general linear system [33]. It is optimal in the sense that in the m^{th} iteration, it searches for

a vector $\boldsymbol{y}$ that minimizes the 2-norm of the residual

$$\|\boldsymbol{f} - \boldsymbol{A}\boldsymbol{V}_m y\|_2$$

where the columns of $\boldsymbol{V}_m$ form an orthonormal basis for $\mathcal{K}_m(\boldsymbol{A}, \boldsymbol{f})$.

3.3 The Multigrid V-Cycle

Multigrid methods rely on solving a series of problems to approximate the solution of (3.1). There are different ways to formulate multigrid methods, and each of them could be useful in various scenarios. Here, we limit our discussion to the multigrid V-cycle, which is arguably the most natural and popular kind of multigrid methods in the literature. For other kinds, such as the W-cycle and full cycle, we refer the readers to [8]. Note that we will consider Galerkin-product-based multigrid methods only.

There are several concepts that we need to formalize to more rigorously introduce multigrid methods. The first is the concept of "level". We let Ω^l denote the space of vectors of dimension N_l. On the l^{th} level of the V-cycle scheme, we are given a linear system

$$\boldsymbol{A}_{g,\beta}^{(l)} \boldsymbol{e}^{(l)} = \boldsymbol{r}^{(l)} \tag{3.7}$$

where $\boldsymbol{A}_{g,\beta}^{(l)}$ is an operator mapping from Ω^l to Ω^l. In its matrix form, $\boldsymbol{A}_{g,\beta}^{(l)}$ is of size $N_l \times N_l$, and we remark that N_l decreases as l increases. Note that (3.1) is the linear system on the first level and that we assume the L^{th} level to be the last level.

The second is the concept of smoothing and smoothness. Let us first define what a smoothing step means in multigrid methods. Applying a smoothing step to a linear system is to apply a relaxation method (introduced in Section 3.1) for several iterations to approximate the solution of the linear system. The matrix $\boldsymbol{M}$ in (3.9)

is therefore also known as a smoother. Recall that we have derived in (3.5) that applying a relaxation method to a linear system is equivalent to applying a linear evolution on the error component by the formula

$$\boldsymbol{e}^{(k+1)} = \boldsymbol{G}\boldsymbol{e}^{(k)}, \tag{3.8}$$

in which $\boldsymbol{G}$ is defined upon the smoother $\boldsymbol{M}$. As such, in the multigrid context, smooth vectors are defined as those vectors that are not easily damped away through the formula (3.8). For a more detailed account of that, we refer the readers to [8].

With the concepts of smoothing and smoothness defined, let us introduce the algorithm. On each level in the multigrid hierarchy before the L^{th} level is reached, a pre-smoothing step is firstly applied by computing

$$\boldsymbol{e}^{(l)} \leftarrow (\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A}^{(l)}_{g,\beta})\boldsymbol{e}^{(l)} + \boldsymbol{M}^{-1}\boldsymbol{r}^{(l)} \tag{3.9}$$

for ν times to eliminate the error components of high frequencies. $\boldsymbol{M}$ is a pre-determined smoothing operator which we assume to take on the same form consistently throughout the multigrid hierarchy. Let

$$\boldsymbol{P}^{(l)} : \Omega^{l+1} \rightarrow \Omega^{l}$$

denote the prolongation operator on the l^{th} level and

$$\boldsymbol{R}^{(l)} : \Omega^{l} \rightarrow \Omega^{l+1}$$

denote the restriction operator on the l^{th} level. The system is then projected to the

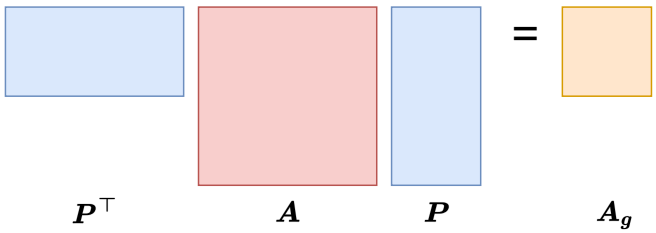

Figure 3.1: Galerkin product illustration.

next level by computing

$$\begin{cases} \boldsymbol{A}_{g,\beta}^{(l+1)} \leftarrow \boldsymbol{R}^{(l)} \boldsymbol{A}_{g,\beta}^{(l)} \boldsymbol{P}^{(l)} \\ \boldsymbol{r}^{(l+1)} \leftarrow \boldsymbol{R}^{(l)} \left(\boldsymbol{r}^{(l)} - \boldsymbol{A}^{(l)} \boldsymbol{e}^{(l)} \right), \quad \forall\, l \in \{1, 2, \ldots, L-1\}. \end{cases}$$

The formula

$$\boldsymbol{A}_{g,\beta}^{(l+1)} \leftarrow \boldsymbol{R}^{(l)} \boldsymbol{A}_{g,\beta}^{(l)} \boldsymbol{P}^{(l)}$$

is what we have been mentioning as the Galerkin product. We remark that $\boldsymbol{R}^{(l)}$ is typically chosen as $\boldsymbol{P}^{(l)}$. An matrix illustration of it is provided in Figure 3.1. From the figure, it is not hard to realize that the size of the linear operator gets smaller after the Galerkin product is applied.

After the projection step, a coarse-grid correction step is then applied by doing

$$\boldsymbol{e}^{(l)} \leftarrow \boldsymbol{P}^{(l)} \boldsymbol{e}^{(l+1)}$$

and a post-smoothing step of the same form as (3.9) is then followed. Lastly, when the algorithm gets to the L^{th} level, we simply solve (3.7) with a direct solver, which will not be too computationally expensive. This is because N_L is already pretty small, and direct solvers are able to solve small systems efficiently and accurately.

The algorithmic details are summarized in Algorithm 1.

Algorithm 1: The Multigrid V-Cycle Scheme [21]

Input: l, L, $\boldsymbol{A}_\beta$, $\boldsymbol{f}$, $\{\boldsymbol{P}^{(k)}\}_{k=1}^{L-1}$, $\{\boldsymbol{R}^{(k)}\}_{k=1}^{L-1}$, $\boldsymbol{M}$, ν

Output: $\boldsymbol{u}^{(l)}$ such that $\boldsymbol{A}_\beta \boldsymbol{u} \approx \boldsymbol{f}$

1: Initialize $\boldsymbol{u}^{(l)} = \boldsymbol{0}$
2: Do $\boldsymbol{u}^{(l)} \leftarrow (\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A}_\beta)\boldsymbol{u}^{(l)} + \boldsymbol{M}^{-1}\boldsymbol{f}$ for ν steps
3: Compute $\boldsymbol{r} = \boldsymbol{f} - \boldsymbol{A}_\beta \boldsymbol{u}^{(l)}$
4: Compute $\boldsymbol{A}_{g,\beta}^{(l+1)} = \boldsymbol{R}^{(l)}\boldsymbol{A}_\beta \boldsymbol{P}^{(l)}$, $\boldsymbol{r}^{(l+1)} = \boldsymbol{R}^{(l)}\boldsymbol{r}$
5: **if** $l+1$ is L **then**
6: Solve for $\boldsymbol{e}^{(l+1)}$ in $\boldsymbol{A}_{g,\beta}^{(l+1)}\boldsymbol{e}^{(l+1)} = \boldsymbol{r}^{(l+1)}$
7: **else**
8: Get $\boldsymbol{e}^{(l+1)}$ by doing V-Cycle with inputs $l+1$, L, $\boldsymbol{A}_{g,\beta}^{(l+1)}, \boldsymbol{r}^{(l+1)}$, $\{\boldsymbol{P}^{(k)}\}$, $\{\boldsymbol{R}^{(k)}\}$, $\boldsymbol{M}$, ν
9: **end if**
10: Prolongate and correct: $\boldsymbol{u}^{(l)} \leftarrow \boldsymbol{u}^{(l)} + \boldsymbol{P}^{(l)}\boldsymbol{e}^{(l+1)}$
11: Do $\boldsymbol{u}^{(l)} \leftarrow (\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A}_\beta)\boldsymbol{u}^{(l)} + \boldsymbol{M}^{-1}\boldsymbol{f}$ for ν steps
12: **return** $\boldsymbol{u}^{(l)}$

Chapter 4

The Issue of Increasing Operator Density in Multigrid

Recall that in Galerkin-product-based AMG, a key step is to project a finer-level linear system onto a coarser level by computing the Galerkin product (3.3). When doing so, the size of the system typically decreases by at least a half. However, it has been observed that although the size gets smaller, the linear operator tends to lose sparsity. This gets more severe as the level goes deeper. To demonstrate this, let us consider the 3-dimensional Poisson's equation

$$-\Delta u = f \tag{4.1}$$

discretized with the seven-point finite difference method on a $100 \times 100 \times 100$ grid [3]. The matrix sparsity patterns in the hierarchy of the V-cycle are presented in Figure 4.1. From the figure, it is clear that the bandwidths of the matrices increase as the level goes deeper. Table 4.1 shows that as the problem size decreases, the average number of non-zeros increases substantially. The diminishing sparsity patterns in coarse grid operators induce a growth in the cost of data communications in parallel solvers. Consequently, a much longer period of time has to be spent on solving the

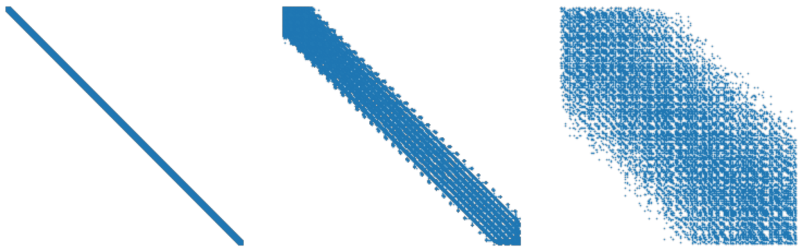

Figure 4.1: The sparsity patterns of $\boldsymbol{A}_g^{(1)}$, $\boldsymbol{A}_g^{(4)}$, and $\boldsymbol{A}_g^{(5)}$ in Algorithm 1 when using it to solve the 3-dimensional Poisson's equation

residual equations on coarser-levels. Figure 4.2 shows the total costs partitioned into the local computation cost and the communication cost in multigrid hierarchies for solving (4.1). We see from the figure that the decreasing problem size causes a reduction in the cost of the actual computing as the level goes deeper. While this is expected, starting from the first coarse level (level 2), the total computational cost is dominated by data communications.

level	matrix size	non-zero entries	non-zero entries per row
1	1,000,000	6,940,000	7
2	500,000	9,320,600	19
3	83,338	2,775,206	34
4	10,401	517,309	50
5	738	30,272	42

Table 4.1: The sparsity data for the coarse-level operators when using classical multigrid V-cycle to solve 3-D Poisson's equation.

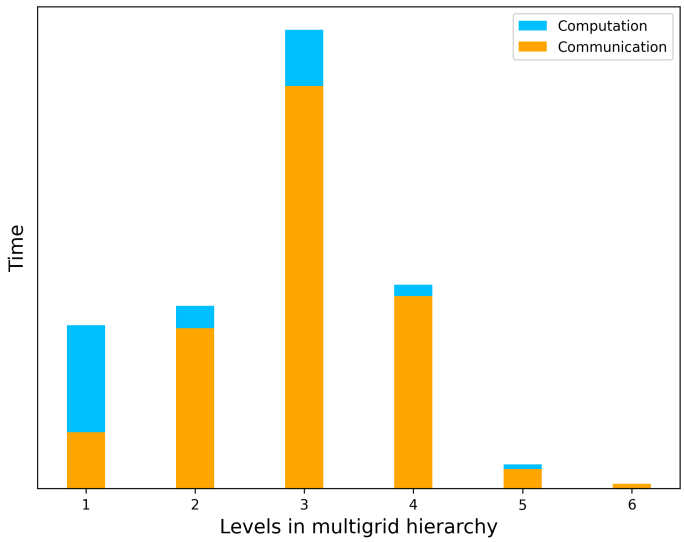

Figure 4.2: Communication cost and computation cost in different levels when using multigrid to solve 3-D Poisson's Equation [4]

Chapter 5

Sparsification of Coarse-Grid Operators via Machine Learning

In this chapter, we detail the design of our algorithm.

5.1 Spectral Equivalence and Multigrid Convergence

Our algorithmic design relies on the notion of spectral equivalence, which is defined upon two sequences of matrices with increasing sizes.

Definition (*Spectrally Equivalent Sequences of Matrices [1, 5]*)**.** *Let* $\{A_j\}$ *and* $\{B_j\}$ *be two sequences of (positive definite) matrices with increasing size* N_j*, where* A_j *and* $B_j \in \mathbb{R}^{N_j \times N_j}$*. If the eigenvalues of* $B_j^{-1}A_j$*,* $\lambda(B_j^{-1}A_j)$*, satisfy*

$$0 < \alpha < \lambda(B_j^{-1}A_j) \leq \beta < \infty$$

for all j *and* α *and* β *are mesh independent (i.e., independent on* N_j*), then the sequences* $\{A_j\}$ *and* $\{B_j\}$ *are said to be spectrally equivalent sequences of matrices.*

Intuitively, this definition formalizes the similarity between the spectrum of two sequences of matrices. This provides a natural way of defining spectrally equivalent stencils. Recall that a stencil can represent a matrix of any size, as long as the pattern remains the same. Therefore, for two stencils to be spectrally equivalent, it is reasonable to require the spectrum of any two matrices generated by the respective stencils to be similar, as long as the sizes of the two matrices are the same. This is summarized in the following definition.

Definition (*Spectrally Equivalent Stencils [5]*)**.** *Suppose the sequences of matrices* $\{\boldsymbol{A}_j\}$ *and* $\{\boldsymbol{B}_j\}$ *are constructed with stencils* $\mathcal{A}$ *and* $\mathcal{B}$ *respectively, and the size of* $\boldsymbol{A}_j$ *is the same as that of* $\boldsymbol{B}_j$ *for all* j*. We say* $\mathcal{A}$ *and* $\mathcal{B}$ *are spectrally equivalent if* $\{A_j\}$ *and* $\{B_j\}$ *are spectrally equivalent in the sense of Definition* 5.1.

With the definition, several questions could be of immediate interest: 1) does there exist spectrally equivalent stencils? 2) for a fixed stencil, does there exist a stencil spectrally equivalent to that particular one? 3) if the answer to the last question is positive, how to find such a stencil computationally? and 4) is it possible for the spectrally equivalent stencil to have fewer non-zero entries so that the multigrid method can be benefited?

We first answer question 1) by considering a special class of stencils, namely the circulant stencils (which will be defined shortly). In fact, it has been shown that there exists a closed-form 5-point stencil that is spectrally equivalent to a 9-point circulant stencil [5]. We summarize this in the next theorem.

Theorem 5.1.1 (*Spectrally Equivalent Circulant Stencils*)**.** *Suppose a 9-point circulant stencil of the form*

$$\begin{bmatrix} c & b & c \\ a & -2(a+b)-4c & a \\ c & b & c \end{bmatrix} \tag{5.1}$$

has generating symbol as a unique single zero at the origin. It is spectrally equivalent to the following 5-point stencil

$$\begin{bmatrix} & b+2c & \\ a+2c & -2(a+b)-8c & a+2c \\ & b+2c & \end{bmatrix}. \tag{5.2}$$

Proof. See [5]. □

Theorem 5.1.1 proves to us that it is indeed possible for spectrally equivalent stencils to exist. Not only that, it is even possible to write out a closed-form stencil that is much sparser than the original one. But is this always the case? To the best of our knowledge, the answer is no. In fact, a theoretically justifiable way of determining whether a spectrally equivalent stencil exists for any target stencil or finding such a stencil if it actually exists has not been developed yet.

However, what can be done is to develop a heuristic so that the multigrid convergence is not affected by a lot. This is sufficient for our goal, because in the end, what we are trying to do is to apply the multigrid method to solve linear systems, and we would like to improve a part of the method, namely to sparsify the coarse-grid operators, without sacrificing other parts, namely the convergence. In particular, the heuristic will be developed for the two-level multigrid V-cycle, or the two-grid scheme (TG). Such an effort suffices because a V-cycle is a recursive version of TG.

Let us introduce some notations first. We use $\boldsymbol{A}$, $\boldsymbol{A}_g$, and $\boldsymbol{A}_c$ to denote the operator in the original problem, the coarse-grid operator defined by the Galerkin product, and the sparsified coarse-grid operator in TG respectively. Following the notations used in [10], we let

$$\boldsymbol{E}_g = (\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A})^\nu (\boldsymbol{I} - \boldsymbol{P}\boldsymbol{A}_g^{-1}\boldsymbol{R}\boldsymbol{A})(\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A})^\nu$$

be the error propagating operator of TG using $\boldsymbol{A}_g$ as the coarse-grid operator. Similarly, let

$$\boldsymbol{E}_c = (\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A})^\nu(\boldsymbol{I} - \boldsymbol{P}\boldsymbol{A}_c^{-1}\boldsymbol{R}\boldsymbol{A})(\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A})^\nu.$$

be the error propagating operator when using $\boldsymbol{A}_c$ as the coarse-grid operator.

The next theorem provides a bound on the spectral radius of $\boldsymbol{E}_c$, which determines the convergence of multigrid methods.

Theorem 5.1.2 (Bounding Condition Numbers and Spectral Radius [10])**.** *Let $\boldsymbol{B}_g$ and $\boldsymbol{B}_c$ be defined by the equations $\boldsymbol{E}_g = \boldsymbol{I} - \boldsymbol{B}_g^{-1}\boldsymbol{A}$ and $\boldsymbol{E}_c = \boldsymbol{I} - \boldsymbol{B}_c^{-1}\boldsymbol{A}$. Define*

$$\phi := \|\boldsymbol{I} - \boldsymbol{A}_c\boldsymbol{A}_g^{-1}\|_2 = \|\boldsymbol{I} - \boldsymbol{A}_g^{-1}\boldsymbol{A}_c\|_2. \tag{5.3}$$

Assume $\boldsymbol{A}_c$ and $\boldsymbol{A}_g$ are both SPD, if $\phi < 1$, then:

$$\kappa(\boldsymbol{B}_c^{-1}\boldsymbol{A}) \leq (\frac{1+\phi}{1-\phi})\kappa(\boldsymbol{B}_g^{-1}\boldsymbol{A}),$$

and

$$\rho(\boldsymbol{E}_c) \leq \max(\lambda_{\max}(\boldsymbol{B}_g^{-1}\boldsymbol{A}) \cdot \frac{1}{1-\phi} - 1, 1 - \lambda_{\min}(\boldsymbol{B}_g^{-1}\boldsymbol{A}) \cdot \frac{1}{1+\phi}), \tag{5.4}$$

where $\kappa(\cdot)$ denotes the condition number, $\lambda_{\max}(\cdot)$ denotes the largest eigenvalue, and $\lambda_{\min}(\cdot)$ denotes the smallest eigenvalue.

By bounding the spectral radius of $\boldsymbol{E}_c$ with ϕ, we can establish a criterion for the convergence of TG with $\boldsymbol{E}_c$. This is demonstrated in the next corollary.

Corollary 5.1.3. *Following **Theorem** 5.1.2, if $\phi < 1 - \lambda_{\max}(\boldsymbol{B}_g^{-1}\boldsymbol{A})/2$, TG with $\boldsymbol{A}_c$ converges.*

Proof. Note that

$$\phi < 1 - \lambda_{\max}(\boldsymbol{B}_g^{-1}\boldsymbol{A})/2$$

implies that

$$\lambda_{\max}(\boldsymbol{B}_g^{-1}\boldsymbol{A}) \cdot \frac{1}{1-\phi} - 1 < 1.$$

Since $\boldsymbol{B}_g$ and $\boldsymbol{A}$ are SPD [10], it follows that

$$1 - \lambda_{\min}(\boldsymbol{B}_g^{-1}\boldsymbol{A}) \cdot \frac{1}{1+\phi} < 1.$$

Further, the convergence of TG with $\boldsymbol{A}_g$ ensures

$$\rho(\boldsymbol{E}_g) = \max\left(\lambda_{\max}(\boldsymbol{B}_g^{-1}\boldsymbol{A}) - 1, 1 - \lambda_{\min}(\boldsymbol{B}_g^{-1}\boldsymbol{A})\right) < 1$$

which gives $\lambda_{\min}(\boldsymbol{B}_g^{-1}\boldsymbol{A}) > 0$ and $\lambda_{\max}(\boldsymbol{B}_g^{-1}\boldsymbol{A}) < 2$. This combined with (5.4) completes the proof. □

5.2 The Algorithmic Pipeline

Recall that we consider a class of problems of the form

$$\boldsymbol{A}_\beta \boldsymbol{u} = \boldsymbol{f}$$

where $\beta \in \mathscr{B} \subset \mathbb{R}^p$, and $\mathscr{B}$ is associated with some probability distribution p_β. For some fixed $\beta \in \mathscr{B}$, the procedure of our algorithm reads as follows. On the l^{th} level, given the coarse grid stencil $\mathcal{A}_{g,\beta}^{(l)}$, the construction of the sparse stencil $\mathcal{A}_{c,\beta}^{(l)}$ involves two steps:

1. select the locations of non-zero entries in $\mathcal{A}_{c,\beta}^{(l)}$, the full procedure of which will be discussed shortly;

2. compute the values of the non-zero entries with another neural network $g_{\psi_l}^{(l)}$ where ψ_l denotes the parameters for the network.

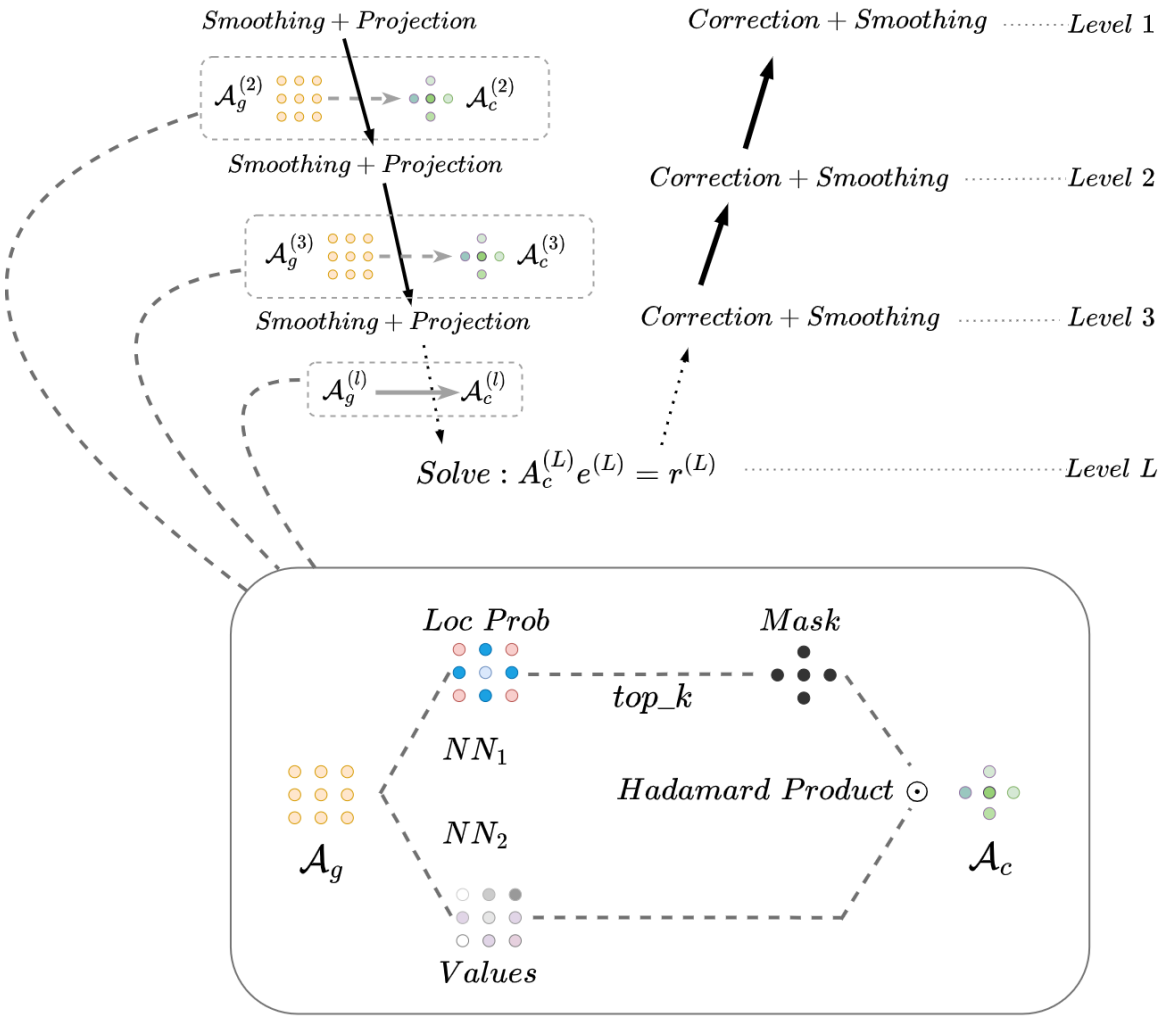

Figure 5.1: Illustration of learning to sparsify coarse-grid operators.

We now discuss the procedure of step 1. We let $f_{\theta_l}^{(l)}$ be a neural network with parameters θ_l. $f_{\theta_l}^{(l)}$ is directly applied to the vectorized input stencil, $vec(\mathcal{A}_{g,\beta}^{(l)})$, meaning that the input stencil is regrouped into a single vector and the neural network operations are applied on that vector. Followed by this, the softmax function is applied to the output of $f_{\theta_l}^{(l)}$. We thus define

$$\mathcal{P}_{\theta_l}^{(l)} := softmax\left(f_{\theta_l}^{(l)}\left(vec\left(\mathcal{A}_{g,\beta}^{(l)}\right)\right)\right).$$

Here, due to the $softmax$ function, each entry of $\mathcal{P}_{\theta_l}^{(l)}$ can be interpreted as the probability of that entry being non-zero in the final sparsified stencil $\mathcal{A}_{c,\beta}^{(l)}$. We select the largest k entries in $\mathcal{P}_{\theta_l}^{(l)}$ and let $\mathcal{I}_{\theta_l}^{(l)}$ be the set of the indices of those entries. That is,

$$\mathcal{I}_{\theta_l}^{(l)} := \left\{ index \;\middle|\; \mathcal{P}_{\theta_l}^{(l)}(index) \text{ is among the largest } k \text{ entries in } \mathcal{P}_{\theta_l}^{(l)} \right\}$$

We then build a mask matrix $\mathcal{M}_{\theta_l}^{(l)}$ with the same shape as that of $\mathcal{P}_{\theta_l}^{(l)}$, and $\mathcal{M}_{\theta_l}^{(l)}$ is defined as

$$\mathcal{M}_{\theta_l}^{(l)}\left(index\right) = \begin{cases} 1, & \text{if } index \in \mathcal{I}_{\theta_l}^{(l)} \\ 0, & \text{otherwise} \end{cases},$$

which determines the position of the non-zero entries in the final sparsified stencil.

Now, the neural network $g_{\psi_l}^{(l)}$ is also applied directly to the input dense stencil. We use

$$\mathcal{V}_{\psi_l}^{(l)} := g_{\psi_l}^{(l)}\left(vec\left(\mathcal{A}_{g,\beta}^{(l)}\right)\right)$$

to denote the output of $g_{\psi_l}^{(l)}$, which determines the values of the non-zero entries in the final sparsified stencil. Finally, we obtain the sparsified stencil by computing the Hadamard (or element-wise) product of $\mathcal{M}_{\theta_l}^{(l)}$ and $\mathcal{V}_{\psi_l}^{(l)}$:

$$\mathcal{A}_{c,\beta}^{(l)} = \mathcal{M}_{\theta_l}^{(l)} \odot \mathcal{V}_{\psi_l}^{(l)}$$

where $\odot$ denotes the Hadamard product. This procedure is summarized in Algorithm 2 and the complete sparsified V-cycle is summarized in Algorithm 3.

We add several remarks. First, note that the integer k which determines the number of non-zero entries in the final sparsified stencil is a customizable integer. Therefore, the sparsification rate is customizable. Second, the shape of the mask matrix $\mathcal{M}_{\theta_l}^{(l)}$ and the shape of the value matrix $\mathcal{V}_{\psi_l}^{(l)}$ are set to be the same through the implementations of neural networks. Therefore, the Hadamard product is applied properly. Third, here we assume the neural networks $f_{\theta_l}^{(l)}$ and $g_{\psi_l}^{(l)}$ have already been properly trained. How they are trained and why the neural networks shall output a stencil that does not affect the convergence of the multigrid methods will be discussed in the next section.

Algorithm 2: $Sparsify(\mathcal{A}_g, f_\theta, g_\psi, k)$

1: Compute $\mathcal{P} = f_\theta(\mathcal{A}_g)$ and $\mathcal{V} = g_\psi(\mathcal{A}_g)$
2: Set the k largest values in $\mathcal{P}$ to 1 and the rest to 0
3: Compute $\mathcal{A}_c = \mathcal{P} \odot \mathcal{V}$
4: **return** $\mathcal{A}_c$

5.3 Training, Testing, and Generalization

The most important components in Algorithm 3 are the neural networks $f_{\theta_l}^{(l)}$ and $g_{\psi_l}^{(l)}$. In the algorithm, it is assumed that they have been trained well. In this section, we discuss how they are trained to output a stencil that is sparser than the input stencil but still maintains the convergence behavior of the multigrid method.

The architecture of the neural networks is selected to be the multi-headed neural network discussed in Section 2.3, which owes to an empirical observation that such an architecture works better than the vanilla deep neural networks on our task.

How to choose the loss function is the most crucial and yet trickiest part. If we

Algorithm 3: Sparsified V-Cycle

Input: l, L, $\boldsymbol{A}$, $\boldsymbol{f}$, $\boldsymbol{M}$, ν, k, $\{\boldsymbol{P}^{(l)}\}_{l=1}^{L-1}$, $\{\boldsymbol{R}^{(l)}\}_{l=1}^{L-1}$, $\{f_{\theta_l}^{(l)}\}_{l=2}^{L}$, $\{g_{\psi_l}^{(l)}\}_{l=2}^{L}$

Output: $\boldsymbol{u}^{(l)}$ such that $\boldsymbol{A}\boldsymbol{u}^{(l)} \approx \boldsymbol{f}$

Initialize $\boldsymbol{u}^{(l)} = \boldsymbol{0}$
if $l > 1$ **then**
    Get $\mathcal{A}$ from $\boldsymbol{A}$
    $\mathcal{A}_c \leftarrow Sparsify\left(\mathcal{A}, f_{\theta_l}^{(l)}, g_{\psi_l}^{(l)}, k\right)$ and generate $\boldsymbol{A}_c$ accordingly
    $\boldsymbol{A} \leftarrow \boldsymbol{A}_c$
end if
Do $\boldsymbol{u}^{(l)} \leftarrow (\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A})\boldsymbol{u}^{(l)} + \boldsymbol{M}^{-1}\boldsymbol{f}$ for ν steps
Compute $\boldsymbol{r} = \boldsymbol{f} - \boldsymbol{A}\boldsymbol{u}^{(l)}$
Compute $\boldsymbol{A}_g^{(l+1)} = \boldsymbol{R}^{(l)}\boldsymbol{A}\boldsymbol{P}^{(l)}$, $\boldsymbol{r}_g^{(l+1)} = \boldsymbol{R}^{(l)}\boldsymbol{r}$
Get $\mathcal{A}_g^{(l+1)}$ from $\boldsymbol{A}_g^{(l+1)}$
$\mathcal{A}_c^{(l+1)} \leftarrow Sparsify\left(\mathcal{A}_g^{(l+1)}, f_{\theta_{l+1}}^{(l+1)}, g_{\psi_{l+1}}^{(l+1)}\right)$
Generate $\boldsymbol{A}_c^{(l+1)}$ accordingly
if $l+1$ is L **then**
    Solve for $\boldsymbol{e}_g^{(l+1)}$ in $\boldsymbol{A}_c^{(l+1)}\boldsymbol{e}_g^{(l+1)} = \boldsymbol{r}_g^{(l+1)}$
else
    Get $\boldsymbol{e}_g^{(l+1)}$ by doing V-Cycle with inputs $l+1$, L, $\boldsymbol{A}_g^{(l+1)}, \boldsymbol{r}_g^{(l+1)}$, $\boldsymbol{M}$, ν, k, $\{\boldsymbol{P}^{(l)}\}_{l=1}^{L-1}$, $\{\boldsymbol{R}^{(l)}\}_{l=1}^{L-1}$, $\{f_{\theta_l}^{(l)}\}_{l=2}^{L}$, $\{g_{\psi_l}^{(l)}\}_{l=2}^{L}$
end if
Prolongate and correct: $\boldsymbol{u}^{(l)} \leftarrow \boldsymbol{u}^{(l)} + \boldsymbol{P}^{(l)}\boldsymbol{e}_g^{(l+1)}$
Do $\boldsymbol{u}^{(l)} \leftarrow (\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A})\boldsymbol{u}^{(l)} + \boldsymbol{M}^{-1}\boldsymbol{f}$ for ν steps
return $\boldsymbol{u}^{(l)}$

follow what has been discussed in Section 5.1, it seems that Corollary 5.1.3 would imply that the quantity ϕ might be a good choice of loss function for the networks to give a sparser $\boldsymbol{A}_c$. However, there is a caveat:

$$\min \phi = \|\boldsymbol{I} - \boldsymbol{A}_g^{-1}\boldsymbol{A}_c\|_2$$

is equivalent to

$$\min \min_{\boldsymbol{v},\, \|\boldsymbol{v}\|_2=1} \left\| \left(\boldsymbol{I} - \boldsymbol{A}_g^{-1}\boldsymbol{A}_c\right)\boldsymbol{v} \right\|_2$$

which simply reads

$$\min_{\boldsymbol{v},\, \|\boldsymbol{v}\|_2=1} \left\| \left(\boldsymbol{I} - \boldsymbol{A}_g^{-1}\boldsymbol{A}_c\right)\boldsymbol{v} \right\|_2$$

by the definition of the spectral norm. This is to say that to minimize ϕ, we have to minimize the norm of $\left(\boldsymbol{I} - \boldsymbol{A}_g^{-1}\boldsymbol{A}_c\right)\boldsymbol{v}$ for all $\boldsymbol{v}$ with a unit 2-norm. In practice, such a requirement is much demanding because a deep learning method is not likely to obtain a satisfying loss value when the amount of data is huge, and in this case, the data is all the vectors of with a unit 2-norm. Therefore, it is necessary to relax the loss function to make it easier to optimize.

We first note that by the sub-multiplicativity of the spectral norm, we have

$$\phi = \|\boldsymbol{I} - \boldsymbol{A}_g^{-1}\boldsymbol{A}_c\|_2 \leq \|\boldsymbol{A}_g^{-1}\|_2 \cdot \|\boldsymbol{A}_g - \boldsymbol{A}_c\|_2,$$

which despite not resolving our concern, suggests that as long as the difference between the actions of $\boldsymbol{A}_g$ and $\boldsymbol{A}_c$ on vectors is small enough, it is still possible to minimize ϕ. We then realize that if we assume the residual $\boldsymbol{r}$ at the current multigrid step is in the range of the prolongation operator $\boldsymbol{P}$—that is,

$$\boldsymbol{r} = \boldsymbol{P}\boldsymbol{r}_c$$

for some coarse-grid vector $\boldsymbol{r}_c$—then with the two-grid scheme, replacing $\boldsymbol{A}_g$ with $\boldsymbol{A}_c$ in the coarse-level linear system to elimiate $\boldsymbol{r}$ gives

$$\begin{aligned}\boldsymbol{r}_{\text{new}} &= (\boldsymbol{I} - \boldsymbol{P}(\boldsymbol{A}_c)^{-1}\boldsymbol{R}\boldsymbol{A})\boldsymbol{r} \\ &= (\boldsymbol{I} - \boldsymbol{P}(\boldsymbol{A}_c)^{-1}\boldsymbol{R}\boldsymbol{A})\boldsymbol{P}\boldsymbol{r}_c \\ &= \boldsymbol{P}(\boldsymbol{I} - (\boldsymbol{A}_c)^{-1}\boldsymbol{A}_g)\boldsymbol{r}_c \\ &= \boldsymbol{P}(\boldsymbol{A}_c)^{-1}(\boldsymbol{A}_c - \boldsymbol{A}_g)\boldsymbol{r}_c.\end{aligned}$$

This tells us that ideally, if $\boldsymbol{A}_c\boldsymbol{r}_c = \boldsymbol{A}_g\boldsymbol{r}_c$, then the residual after the two-grid method is 0. Since $\boldsymbol{A}_c$ is sparser than $\boldsymbol{A}_g$, it is not possible to require $\boldsymbol{A}_c\boldsymbol{r}_c = \boldsymbol{A}_g\boldsymbol{r}_c$ to hold for any vector $\boldsymbol{r}_c$. Nevertheless, recall that in multigrid methods, after applying relaxation methods on the fine level, the remaining errors are mostly comprised of algebraically smooth vectors. Hence, researchers have suggested to define $\boldsymbol{A}_c$ so that it has a similar behaviour to $\boldsymbol{A}_g$ when applied on algebraically smooth vectors [35, 7]. Such construction of $\boldsymbol{A}_c$ can then yield similar coarse-level corrections as $\boldsymbol{A}_g$. Therefore, the entire multigrid process should not be influenced much by replacing $\boldsymbol{A}_g$ with $\boldsymbol{A}_c$. We thus should expect similar convergence behavior for the method.

As such, we choose to minimize the difference between the behavior of $\boldsymbol{A}_{g,\beta}^{(l)}$ (the dense coarse-grid operator on the l^{th} level when using multigrid to solve for (**??**)) and $\boldsymbol{A}_{c,\beta}^{(l)}$ (the sparser operator parameterized with neural networks) on algebraically smooth vectors. Define the loss function associated with $f_{\theta_l}^{(l)}$ and $g_{\psi_l}^{(l)}$ for a single β as

$$\mathcal{L}_\beta\left(f_{\theta_l}^{(l)}, g_{\psi_l}^{(l)}, \mathcal{A}_{g,\beta}^{(l)}, \{\boldsymbol{v}_\beta^{(j)}\}_{j=1}^{s_l}\right) = \sum_{j=1}^{s_l} ||\boldsymbol{A}_{g,\beta}^{(l)}\boldsymbol{v}_\beta^{(j)} - \boldsymbol{A}_{c,\beta}^{(l)}\boldsymbol{v}_\beta^{(j)}||_2^2 \tag{5.5}$$

where $\{\boldsymbol{v}_\beta^{(j)}\}_{j=1}^{s_l}$ are algebraically smooth vectors generated using Algorithm 4, s_l is the number of vectors generated, and $\mathcal{A}_{c,\beta}^{(l)}$ is determined through Algorithm 2. Since we want the neural networks to work for any $\beta \sim p_\beta$, the quantity we ideally hope to

minimize during the training of $h_{\theta_l}^{(l)}$ and $g_{\psi_l}^{(l)}$ is therefore

$$\mathbb{E}_{\beta\sim p_\beta}\left[\mathcal{L}_\beta\left(f_{\theta_l}^{(l)}, g_{\psi_l}^{(l)}, \mathcal{A}_{g,\beta}^{(l)}, \{\boldsymbol{v}_\beta^{(j)}\}_{j=1}^{s_l}\right)\right]. \tag{5.6}$$

Note that in practice, we do not form $\boldsymbol{A}_{g,\beta}^{(l)}$ and $\boldsymbol{A}_{c,\beta}^{(l)}$ explicitly. Instead, we replace $\boldsymbol{A}_{g,\beta}^{(l)}\boldsymbol{v}_\beta^{(j)}$ by

$$\mathcal{A}_{g,\beta}^{(l)} * \widetilde{\boldsymbol{v}}_\beta^{(j)}$$

and $\boldsymbol{A}_{c,\beta}^{(l)}\boldsymbol{v}_\beta^{(j)}$ by

$$\mathcal{A}_{c,\beta}^{(l)} * \widetilde{\boldsymbol{v}}_\beta^{(j)}$$

where $*$ denotes the convolution operation and $\widetilde{\boldsymbol{v}}_\beta^{(j)}$ is $\boldsymbol{v}_\beta^{(i)}$ padded with layers of zeros around it. The number of padding layers depends on the size of the stencil. Such a reformulation greatly reduces the memory requirement during training.

The training procedure reads as follows. We first sample β from $\mathscr{B}$ according to the probability distribution p_β to get our parameter set $\{\beta_i\}_{i=1}^{N_t}$ where N_t is the number of training samples. A set of stencils $\{\mathcal{A}_{\beta_i}\}_{i=1}^{N_t}$ for the level-1 system is then constructed. To get a consistent stencil representation for the coarse-grid operators, we need to make some assumptions on the prolongation operators $\{\boldsymbol{P}^{(k)}\}_{k=1}^{L-1}$ and the restriction operators $\{\boldsymbol{R}^{(k)}\}_{k=1}^{L-1}$. From our experience, choosing them as the full-coarsening operators allows stencil representations for all the coarse-level systems. The stencils on the coarse levels $\{\mathcal{A}_{c,\beta_i}^{(l)}\}_{i=1}^{N_t}$ for $2 \le l \le L$ are then constructed after projection accordingly. The practical loss function thus reads

$$\mathcal{L}\left(h_{\theta_l}^{(l)}, g_{\psi_l}^{(l)}, \{\mathcal{A}_{g,\beta}^{(l)}\}_{i=1}^{N_t}\right) := \frac{1}{N_t}\sum_{i=1}^{N_t}\mathcal{L}_{\beta_i}\left(h_{\theta_l}^{(l)}, g_{\psi_l}^{(l)}, \mathcal{A}_{g,\beta_i}^{(l)}, \{\boldsymbol{v}_{\beta_i}^{(j)}\}_{j=1}^{s_l}\right)$$

where $\mathcal{L}_{\beta_i}$ is defined in (5.5). The complete training procedure is summarized in Algorithm 5. We remark that the training of the neural networks is entirely independent of each other on each level. Therefore, they can be trained in parallel once the dataset

is prepared.

The testing set is constructed in a similar way except that we enforce the parameters to be different from those for the training set. That is, we construct $\{\beta_j\}_{j=1}^{N_v}$ in such a way that each β_j has not been seen by the models during training. Here N_v denotes the size of our testing set. This tests the capability of our models of generalizing to unseen parameters.

Other than the generalization to unseen parameters, we would also like our models to generalize to problems of larger sizes. The reason of that lies in the construction of the algebraically smooth vectors during training. Recall that such a construction involves solving a generalized eigenvalue problem, as shown in Algorithm 4. Solving such a problem is highly computationally expensive, especially when the size gets large. Therefore, we do not want to solve a generalized eigenvalue problem with a size one million by one million even before we start solving a linear system of the same size. We thus hope to construct our training data by solving a small-size problem and use the trained model to sparsify coarse-level operators of much larger sizes.

Algorithm 4: Generating Algebraically Smooth Basis

Input: Interpolation operator $\boldsymbol{P}$, coarse-level stencil $\mathcal{A}_g$, number of algebraically smooth vectors: s, size of the eigenvalue problem: n

Output: Algebraically smooth basis $\boldsymbol{v}_i, i = 1, \ldots, k$

1: Compute $\boldsymbol{T} = \boldsymbol{P}^\top \boldsymbol{P}$

2: Compute generalized eigen pairs $\boldsymbol{A}_g \boldsymbol{v}_i = \lambda_i \boldsymbol{T} \boldsymbol{v}_i$.

3: **return** $\boldsymbol{v}_i, i = 1, \ldots, k$ with k smallest $\lambda_i, i = 1, \ldots, k$

Algorithm 5: Learning to Sparsify

Input: Interpolation operator $\boldsymbol{P}$, restriction operator $\boldsymbol{R}$, dense coarse-grid stencils $\mathcal{A}_g^j$ $(j = 1, \ldots, m)$, number of basis vectors: k, sparsification ratio ρ

Output: Two neural networks (NNs) f_θ and g_ψ

1: Generate algebraically smooth $\boldsymbol{v}_i^j, i = 1, \ldots, k$ for each $\mathcal{A}_g^j$

2: Initialize NNs f_θ and g_ψ

3: $\{\mathcal{A}_c^j\}_{j=1,\ldots,m} = \text{Sparsify}(\{\mathcal{A}_g^j\}_{j=1,\ldots,m}, f_\theta, g_\psi, \rho)$

4: Minimize the loss function:

$$\mathcal{L}\left(\theta, \psi, \mathcal{A}_g^j, \mathcal{A}_c^j, \boldsymbol{v}_i^j\right) = \sum_{j=1}^{m} \sum_{i=1}^{k} ||\boldsymbol{A}_g^j \boldsymbol{v}_i^j - \boldsymbol{A}_c^j \boldsymbol{v}_i^j||_2^2$$

5: Update the weights of f_θ and g_ψ

Chapter 6

Numerical Experiments

We report numerical experiments with our proposed sparsification method in this chapter. We use PyTorch 1.9.0 [22] for all the deep learning models appeared in the code. We use PyAMG 4.2.3 [2] as our framework. All programs were executed on an Intel Core i7-6700 CPU.

6.1 The Evaluation Metrics

Before going into the actual results, we explain the metrics we use for evaluating the performance of our algorithm.

Recall that **Theorem** 5.1.2 provides us with a quantity ϕ defined in (5.3) to evaluate the convergence behaviour of multigrid methods associated with different stencils. However, ϕ can only depend on the stencil itself. In other words, it cannot be affected drastically if the sizes of $\boldsymbol{A}_g$ and $\boldsymbol{A}_c$ were changed. This subtlety introduces a big practical issue: we simply cannot test for all different matrix sizes. We hence need new metrics. Instead of computing ϕ, we record the number of iterations for the multigrid method to converge to a predetermined tolerance with each stencil. We generate sufficient amount of random test cases and take down the average iterations required for them to converge. This is better than computing ϕ for different meshes

because even if we compute a very large amount of ϕ and they are all consistent with Theorem 5.1.2, we still cannot draw the conclusion that ϕ is mesh independent. But for the metric presented here, if it does not vary much between different stencils, it is reasonable to claim that our method works for the class of testing problems being considered with very high probability by the Law of Large Numbers. To examine whether or not the sparsified stencil is spectrally equivalent to the original stencil, we provide the spectrum plot of the associated matrices of several matrix sizes.

6.2 Circulant Stencils

We first test our method on problems defined by circulant stencils as Theorem 5.1.1 provides theoretical guidance for verifying our result. This allows for direct evaluation of our learned sparser stencil by comparing it with the sparser stencil provided by the theorem. The experiment is set up as follows.

We first randomly generate three numbers

$$a = 2.720,\ b = 1.417,\ c = 0.000114$$

to construct a stencil

$$\begin{aligned} \mathcal{A} &= \begin{bmatrix} c & a & c \\ b & -2\cdot(a+b+2c) & b \\ c & a & c \end{bmatrix} \\ &= \begin{bmatrix} 0.000114 & 2.720 & 0.000114 \\ 1.417 & -8.275 & 1.417 \\ 0.000114 & 2.720 & 0.000114 \end{bmatrix} \end{aligned} \tag{6.1}$$

for the linear system on the first level. We select the prolongation operator so that it

has a stencil presentation of the form

$$\boldsymbol{P} = \frac{1}{4}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}.$$

Such a circulant stencil will ensure the stencil of the coarse-level operator is also circulant. In fact, after projection, the coarse-level stencil is

$$\mathcal{A}_g^{(2)} = \begin{bmatrix} 0.129 & 0.096 & 0.129 \\ 0.422 & -1.551 & 0.422 \\ 0.129 & 0.096 & 0.129 \end{bmatrix}.$$

According to **Theorem** 5.1.1, the optimal 5-point stencil spectrally equivalent to $\mathcal{A}_g^{(2)}$ here is

$$\mathcal{A}_{optimal}^{(2)} = \begin{bmatrix} 0 & 0.354 & 0 \\ 0.680 & -2.069 & 0.680 \\ 0 & 0.354 & 0 \end{bmatrix}.$$

We train the model on the instance defined above and set the size of the linear system as 31×31. The stencil learned by our algorithm reads

$$\mathcal{A}_c^{(2)} = \begin{bmatrix} 0 & 0.348 & 0 \\ 0.666 & -2.024 & 0.663 \\ 0 & 0.347 & 0 \end{bmatrix}$$

which looks pretty similar to $\mathcal{A}_{optimal}^{(2)}$. To check the convergence behavior of multigrid on problems with larger mesh sizes (than 31×31) with different coarse-level operators, we run the two-level multigrid V-cycle on the linear system defined by (6.1) with the coarse-level operator being $\mathcal{A}_g^{(2)}$, $\mathcal{A}_{optimal}^{(2)}$, and $\mathcal{A}_c^{(2)}$ respectively. The relative-error

tolerance is set to be 10^{-6} and the right-hand-side vector is generated randomly. We record the result in Table 6.1.

mesh-size	63	95	127	191	255	383	511
$\mathcal{A}_g^{(2)}$	11	10	10	10	10	10	10
$\mathcal{A}_{optimal}^{(2)}$	11	10	10	10	10	10	10
$\mathcal{A}_c^{(2)}$	11	10	10	10	10	10	10

Table 6.1: The number of iterations for the two-level multigrid to converge to a relative-error tolerance of 10^{-6} using $\mathcal{A}_g^{(2)}$, $\mathcal{A}_{optimal}^{(2)}$, and $\mathcal{A}_c^{(2)}$ as the coarse-level operator respectively. Each column corresponds to a different mesh size. For instance, 63 means the size of the linear system being tested for is $63^2 \times 63^2$.

We see from the table that it takes exactly the same amount of work for the 2-level V-cycle to converge for different operators.

6.3 The 2-D Rotated Laplacian Problem

Consider the 2D anisotropic rotated Laplacian problem

$$-\nabla \cdot (\boldsymbol{T}_{\theta,\xi} \nabla u(x,y)) = f(x,y), \tag{6.2}$$

where the 2×2 vector field $\boldsymbol{T}_{\theta,\xi}$ parameterized by θ and ξ is defined as

$$\boldsymbol{T}_{\theta,\xi} = \begin{bmatrix} \cos^2\theta + \xi \sin^2\theta & \cos\theta \sin\theta(1-\xi) \\ \cos\theta \sin\theta(1-\xi) & \sin^2\theta + \xi \cos^2\theta \end{bmatrix}$$

Here, θ is the angle of the anisotropy and ξ is the conductivity.

In the first set of experiments, we fix ξ and let θ follow a uniform distribution on a certain interval. We conduct 12 experiments where each $\xi \in \{100, 200, 300, 400\}$ is paired up with a $\theta-$interval from $\{(\{\frac{3\pi}{12}, \frac{4\pi}{12}), (\{\frac{4\pi}{12}, \frac{5\pi}{12}), (\{\frac{6\pi}{12}, \frac{7\pi}{12})\}$ and use a 3-level V-cycle for each of them. The dense coarse-level operators $\boldsymbol{A}_g^{(l)}$ is obtained by using

the full-coarsening scheme for 2-dimensional problems: the interpolation operator is selected so that it has a stencil representation of the form

$$\boldsymbol{P} = \frac{1}{4}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

and the restriction operator is set to be $\boldsymbol{R} = \boldsymbol{P}^\top$. We choose to use the Gauss-Seidel method discussed in Section 3.1 for both pre-smoothing and post-smoothing. The relative-error tolerance is set as 10^{-6}: the method stops when $\frac{\|\boldsymbol{f}-\boldsymbol{A}\boldsymbol{u}\|_2}{\|\boldsymbol{f}\|_2} < 10^{-6}$. We set the non-zero elements in the sparsified stencil to be 5 which is a 44% decrease compared to the dense stencil with 9 non-zero elements.

For each experiment, during the training phase, the model sees 5 different instances with the same selected ξ and different θ's that are evenly distributed on the seleted interval. For instance, if $\xi = 100$ and the $\theta-$interval is chosen as $(\frac{3\pi}{12}, \frac{4\pi}{12})$, the parameters for the five instances are $\big\{(100, \frac{3\pi}{12}), (100, \frac{3.25\pi}{12}), (100, \frac{3.50\pi}{12}), (100, \frac{3.75\pi}{12}), (100, \frac{4\pi}{12})\big\}$. The size of the fine-level matrix associated with the training instances is set to be 31×31.

During the testing phase, 10 different θs are drawn randomly from the selected interval. The configurations for the V-cycle are kept the same as those in the training phase. However, the size of the fine-level matrix associate with each testing instance is set to be 511×511 which is about 16 times the size for the training instances. We record the number of iterations required for the 3-level V-cycle to converge and compare the average performance. The result is reported in Table 6.2. It is not hard to see from the table that the convergence behaviour of the method remains almost the same by replacing the original coarse-level operators with the sparsified operators.

Table 6.2: The average number of iterations for the 3-level multigrid method to converge with different coarse-grid operators when solving (6.2) under different sets of parameters. The mesh size is set to be 511×511. The parameters are chosen so that $\xi \in \{100, 200, 300, 400\}$ is fixed and θ is randomly sampled from a uniform distribution on each interval listed below. The average is taken over at least 10 different sampled θ's. $\mathcal{A}_g$ denotes the coarse-grid operators obtained from Galerkin products; and $\mathcal{A}_c$ corresponds to coarse-grid operators sparsified by the proposed algorithm.

$\xi = 100, \theta$	$(\frac{2\pi}{12}, \frac{3\pi}{12})$	$(\frac{3\pi}{12}, \frac{4\pi}{12})$	$(\frac{6\pi}{12}, \frac{7\pi}{12})$
$\mathcal{A}_g$	92.1	102.8	126.9
$\mathcal{A}_c$	89.0	93.0	135.2
$\xi = 200, \theta$	$(\frac{2\pi}{12}, \frac{3\pi}{12})$	$(\frac{3\pi}{12}, \frac{4\pi}{12})$	$(\frac{6\pi}{12}, \frac{7\pi}{12})$
$\mathcal{A}_g$	191.7	196.6	203.1
$\mathcal{A}_c$	174.2	177.8	204.9
$\xi = 300, \theta$	$(\frac{2\pi}{12}, \frac{3\pi}{12})$	$(\frac{3\pi}{12}, \frac{4\pi}{12})$	$(\frac{6\pi}{12}, \frac{7\pi}{12})$
$\mathcal{A}_g$	248.0	269.7	342.3
$\mathcal{A}_c$	246.5	231.4	356.2
$\xi = 400, \theta$	$(\frac{2\pi}{12}, \frac{3\pi}{12})$	$(\frac{3\pi}{12}, \frac{4\pi}{12})$	$(\frac{6\pi}{12}, \frac{7\pi}{12})$
$\mathcal{A}_g$	337.1	351.1	438.2
$\mathcal{A}_c$	326.3	327.7	441.5

In the second set of experiments, we fix θ and assume ξ follows a uniform distribution on a selected interval. We again conduct 12 experiments in total where each $\theta \in \{\frac{\pi}{6}, \frac{\pi}{4}, \frac{\pi}{3}, \frac{5*\pi}{12}\}$ is paired up with a $\xi-$interval from $\{(5, 10), (80, 100), (100, 200)\}$ and use a 3-level V-cycle for each of them. The configurations (the coarsening scheme, the relative-error tolerance, the number of non-zero elements in the sparsified stencil, etc.) remain the same as in the previous set of experiments.

The training and testing process is also similar: for each experiment, we train the model with 5 different instances evenly distributed on the selected interval and test it for 10 randomly drawn ξs on the interval. The size of the fine-level linear system

in the training instance is set to be 31×31 while the size in each testing instance is set to be 511×511. The average performance is presented in Table 6.3.

Table 6.3: The average number of iterations for the 3-level multigrid method to converge with different coarse-grid operators when solving (6.2) under different sets of parameters. The mesh size is set to be 511×511. The parameters are chosen so that $\theta \in \{\frac{\pi}{6}, \frac{\pi}{4}, \frac{\pi}{3}, \frac{5*\pi}{12}\}$ is fixed and ξ is randomly sampled from a uniform distribution on each interval listed below. The average is taken over at least 10 different sampled θ's. $\mathcal{A}_g$ denotes the coarse-grid operators obtained from Galerkin products; and $\mathcal{A}_c$ corresponds to coarse-grid operators sparsified by the proposed algorithm.

$\theta = \frac{\pi}{6}, \xi$	(100, 200)	(80,100)	(5,10)
$\mathcal{A}_g$	90.4	72.1	13.5
$\mathcal{A}_c$	100.2	84.4	13.8
$\theta = \frac{\pi}{4}, \xi$	(100, 200)	(80,100)	(5,10)
$\mathcal{A}_g$	172.5	105.2	14.1
$\mathcal{A}_c$	123.1	79.0	15.9
$\theta = \frac{\pi}{3}, \xi$	(100, 200)	(80,100)	(5,10)
$\mathcal{A}_g$	99.4	80.9	14.3
$\mathcal{A}_c$	79.1	88.8	15.4
$\theta = \frac{5*\pi}{12}, \xi$	(100, 200)	(80,100)	(5,10)
$\mathcal{A}_g$	92.5	76.4	16.5
$\mathcal{A}_c$	107.4	88.2	16.6

6.4 The 2-Dimensional Linear Elasticity Problem

Consider the 2-D linear elasticity problem in an isotropic homogeneous medium:

$$\mu \nabla^2 u + (\mu + \lambda)(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 v}{\partial x \partial y}) + f_x = 0, \tag{6.3}$$

$$\mu \nabla^2 v + (\mu + \lambda)(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 u}{\partial x \partial y}) + f_y = 0, \tag{6.4}$$

where u and v are the solution in the direction of $x-$ and $y-$ axis respectively. f_x and f_y are forcing terms. μ and λ are Lame coefficients which are determined by Young's modulus E and Poisson's ratio ν:

$$\mu = \frac{E}{2(1+\nu)}, \quad \lambda = \frac{E\nu}{(1+\nu)(1-2\nu)}.$$

Following [15], we use the following 9-point discretization stencils with a mesh size h in the direction of $x-$axis and mesh size $b_y h$ in the direction of $y-$axis:

$$\mathcal{A}_{uu} = \begin{bmatrix} \frac{-\lambda b_y^2 - 2\mu b_y^2 + \lambda + \mu}{4(2\lambda b_y^2 + \lambda + 4(b_y^2+1)\mu)} & \frac{(b_y^2-1)\lambda + 2(b_y^2-2)\mu}{2(2\lambda b_y^2 + \lambda + 4(b_y^2+1)\mu)} & \frac{-\lambda b_y^2 - 2\mu b_y^2 + \lambda + \mu}{4(2\lambda b_y^2 + \lambda + 4(b_y^2+1)\mu)} \\ -\frac{2\lambda b_y^2 + 4\mu b_y^2 + \lambda + \mu}{2(2\lambda b_y^2 + \lambda + 4(b_y^2+1)\mu)} & 1 & -\frac{2\lambda b_y^2 + 4\mu b_y^2 + \lambda + \mu}{2(2\lambda b_y^2 + \lambda + 4(b_y^2+1)\mu)} \\ \frac{-\lambda b_y^2 - 2\mu b_y^2 + \lambda + \mu}{4(2\lambda b_y^2 + \lambda + 4(b_y^2+1)\mu)} & \frac{(b_y^2-1)\lambda + 2(b_y^2-2)\mu}{2(2\lambda b_y^2 + \lambda + 4(b_y^2+1)\mu)} & \frac{-\lambda b_y^2 - 2\mu b_y^2 + \lambda + \mu}{4(2\lambda b_y^2 + \lambda + 4(b_y^2+1)\mu)} \end{bmatrix}$$

$$\mathcal{A}_{uv} = \begin{bmatrix} \frac{3b_y(\lambda+\mu)}{8(2\lambda b_y^2 + \lambda + 4(b_y^2+1)\mu)} & 0 & -\frac{3b_y(\lambda+\mu)}{8(2\lambda b_y^2 + \lambda + 4(b_y^2+1)\mu)} \\ 0 & 0 & 0 \\ -\frac{3b_y(\lambda+\mu)}{8(2\lambda b_y^2 + \lambda + 4(b_y^2+1)\mu)} & 0 & \frac{3b_y(\lambda+\mu)}{8(2\lambda b_y^2 + \lambda + 4(b_y^2+1)\mu)} \end{bmatrix}$$

where $\mathcal{A}_{uu}$ and $\mathcal{A}_{uv}$ are stencils defining the linear system

$$\boldsymbol{A}\boldsymbol{x} := \begin{bmatrix} \boldsymbol{A}_{uu} & \boldsymbol{A}_{uv} \\ \boldsymbol{A}_{uv}^\top & \boldsymbol{A}_{uu}^\top \end{bmatrix} \begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{v} \end{bmatrix} = \begin{bmatrix} \boldsymbol{f}_x \\ \boldsymbol{f}_y \end{bmatrix}. \tag{6.5}$$

The interpolation operator for linear elasticity problem is also defined in block form as

$$\boldsymbol{P} := \begin{bmatrix} \boldsymbol{P}_{uu} & \boldsymbol{P}_{uv} \\ \boldsymbol{P}_{uv}^\top & \boldsymbol{P}_{uu} \end{bmatrix},$$

and the restriction operator $\boldsymbol{R}$ is defined as $\boldsymbol{R} = \boldsymbol{P}^\top$. We use red-black coarsening scheme for each block. The restriction and interpolation stencils for $u-u$ and $v-v$

connections associated with this coarsening scheme are defined by the stencils

$$\frac{1}{4}\begin{bmatrix} & 1 & \\ 1 & 4 & 1 \\ & 1 & \end{bmatrix} \quad \text{and} \quad \frac{1}{4}\begin{bmatrix} & 1 & \\ 1 & 4 & 1 \\ & 1 & \end{bmatrix},$$

the restriction and interpolation stencils for $u - v$ connection are given by

$$\frac{1}{4}\begin{bmatrix} & 1 & \\ -1 & 0 & -1 \\ & 1 & \end{bmatrix} \quad \text{and} \quad \frac{1}{4}\begin{bmatrix} & 1 & \\ -1 & 0 & -1 \\ & 1 & \end{bmatrix},$$

and the restriction and interpolation stencils for $v - u$ connection are given by

$$\frac{1}{4}\begin{bmatrix} & -1 & \\ 1 & 0 & 1 \\ & -1 & \end{bmatrix} \quad \text{and} \quad \frac{1}{4}\begin{bmatrix} & -1 & \\ 1 & 0 & 1 \\ & -1 & \end{bmatrix}.$$

Note that as stated in [6], multigrid method requires having row sums of one for the $u - u$ and $v - v$ interpolation weights and row sums of zero for the $u - v$ and $v - u$ weights in order to converge.

We set Young's modulus as $E = 10^{-5}$ vary Poisson's ratio ν. We choose to use the Gauss-Seidel method discussed in Section 3.1 for both pre-smoothing and post-smoothing. The relative-error tolerance is set as 10^{-6}. The sparsification ratio is set as 0.5. We train the model on 4 different instances with $\nu \in \{0.1, 0.2, 0.3, 0.4\}$. The size of the fine-level linear system is set as 9×9. We test the model on instances with randomly drawn ν from each interval in $\{(0.1, 0.2), (0.2, 0.3), (0.3, 0.4)\}$. The size of the linear system associated with each testing instance is set as 65×65. The averaged performance is presented in Table 6.4. Again we see that the convergence behavior of the two-grid method is not affected much by replacing with the learned sparsified

Table 6.4: Averaged numbers of iterations required for the two-grid scheme to converge (in the sense of reaching the relative-error tolerance) when solving (6.5) of size 65×65 with random ν on each interval using $\mathcal{A}_g$ and $\mathcal{A}_c$ as coarse grid operator stencils.

ν	(0.1,0.2)	(0.2,0.3)	(0.3,0.4)
$\mathcal{A}_g$	10.1	10.2	10.6
$\mathcal{A}_c$	11.0	10.7	11.5

stencil.

We make a quick note that for linear elasticity problems, we have four blocks of stencils on the coarse grid and two distinct stencils because of symmetry. The two stencils are combined together before being fed into the neural networks so that the model can learn a better balance between $u - u$ connection and $u - v$ connection rather than learning them separately.

We take a specific problem with $\nu = 0.3$ and $E = 10^{-5}$ as an example. We show the eigenvalues of $\boldsymbol{A}_c^{-1}\boldsymbol{A}_g$ for different mesh sizes in Figure **??**. We can see that the distribution of spectrum is mesh independent indicating spectral equivalence of $\boldsymbol{A}_c$ and $\boldsymbol{A}_g$.

An example of the approximate solutions after the same number of iterations of using two-grid method with $\mathcal{A}_g$ and $\mathcal{A}_c$ as coarse grid stencils is presented in Figure **??** for direct evaluation.

6.5 Comparison with the Sparsified Smooth Aggregation Method

Here we report the comparison between the performance of our method and that of the method proposed in [32] on the rotated Laplacian problems. The method in [32] is based on the idea of sparsified smooth aggregation of nodes, which we will refer to as SpSA. The experiment setup for the testing of our method remains the same. We input the same set of linear systems to the SpSA solver. To make sure of a fair

comparison, we enforce the number of non-zero entries per row in the coarse-level operators after SpSA to be at least as many as that in the operators sparsified by our algorithm. We set the relative tolerance for both methods as 10^{-6}.

Table 6.5: The average number of iterations for the 3-level multigrid method accelerated with GMRES on the top level to converge with different coarse-grid operators when solving (6.2) under different sets of parameters. The mesh size is set to be 511×511. The parameters are chosen so that $\theta \in \{\frac{\pi}{6}, \frac{\pi}{4}, \frac{\pi}{3}, \frac{5*\pi}{12}\}$ is fixed and ξ is randomly sampled from a uniform distribution on each interval listed below. The average is taken over at least 10 different sampled θ's. $\mathcal{A}_g$ denotes the coarse-grid operators obtained from Galerkin products; $\mathcal{A}_c$ corresponds to coarse-grid operators sparsified by the proposed algorithm; and $SpSA$ corresponds to the coarse-grid operator obtained by the sparsified smooth aggregation algorithm proposed in [32].

$\xi = 100, \theta$	$(\frac{2\pi}{12}, \frac{3\pi}{12})$	$(\frac{3\pi}{12}, \frac{4\pi}{12})$	$(\frac{6\pi}{12}, \frac{7\pi}{12})$
$\mathcal{A}_g$	11.3	11.1	11.5
$\mathcal{A}_c$	16.5	16.9	19.9
$SpSA$	40.4	36.9	11.5
$\xi = 200, \theta$	$(\frac{2\pi}{12}, \frac{3\pi}{12})$	$(\frac{3\pi}{12}, \frac{4\pi}{12})$	$(\frac{6\pi}{12}, \frac{7\pi}{12})$
$\mathcal{A}_g$	15.9	15.3	14.5
$\mathcal{A}_c$	20.5	19.6	29.8
$SpSA$	50.4	46.7	9.1
$\xi = 300, \theta$	$(\frac{2\pi}{12}, \frac{3\pi}{12})$	$(\frac{3\pi}{12}, \frac{4\pi}{12})$	$(\frac{6\pi}{12}, \frac{7\pi}{12})$
$\mathcal{A}_g$	18.1	21.5	17.9
$\mathcal{A}_c$	25.4	33.1	25.6
$SpSA$	53.0	52.1	12.3
$\xi = 400, \theta$	$(\frac{2\pi}{12}, \frac{3\pi}{12})$	$(\frac{3\pi}{12}, \frac{4\pi}{12})$	$(\frac{6\pi}{12}, \frac{7\pi}{12})$
$\mathcal{A}_g$	21.1	20.2	19.9
$\mathcal{A}_c$	27.2	30.9	26.2
$SpSA$	59.7	56.3	12.9

Table 6.6: The average number of iterations for the 3-level multigrid method accelerated with GMRES on the top level to converge with different coarse-grid operators when solving (6.2) under different sets of parameters. The mesh size is set to be 511×511. The parameters are chosen so that $\xi \in \{100, 200, 300, 400\}$ is fixed and θ is randomly sampled from a uniform distribution on each interval listed below. The average is taken over at least 10 different sampled θ's. $\mathcal{A}_g$ denotes the coarse-grid operators obtained from Galerkin products; $\mathcal{A}_c$ corresponds to coarse-grid operators sparsified by the proposed algorithm; and $SpSA$ corresponds to the coarse-grid operator obtained by the sparsified smooth aggregation algorithm proposed in [32].

$\theta = \frac{\pi}{6}, \xi$	(100, 200)	(80,100)	(5,10)
$\mathcal{A}_g$	11.6	10.4	4.4
$\mathcal{A}_c$	19.9	16.4	7.1
$SpSA$	30.3	27	12.7
$\theta = \frac{\pi}{4}, \xi$	(100, 200)	(80,100)	(5,10)
$\mathcal{A}_g$	14.2	11.3	4.6
$\mathcal{A}_c$	18.2	15.5	10.0
$SpSA$	41.4	34.7	13.5
$\theta = \frac{\pi}{3}, \xi$	(100, 200)	(80,100)	(5,10)
$\mathcal{A}_g$	11.2	10.2	4.7
$\mathcal{A}_c$	18.8	16.4	7.1
$SpSA$	31.7	28.9	13.4
$\theta = \frac{5*\pi}{12}, \xi$	(100, 200)	(80,100)	(5,10)
$\mathcal{A}_g$	11.2	10.1	4.8
$\mathcal{A}_c$	28.8	19.1	9.1
$SpSA$	16.5	15.1	9.1

Chapter 7

Discussion and Future Work

The multigrid method we are considering in this work is fully defined with matrix-vector products without taking advantage of the underlying geometric information. Therefore it is a legitimate algebraic multigrid method. However, this does not mean that our sparsification algorithm is applicable to every kind of AMG methods. Our biggest limitation is that we are assuming for a constant stencil representation for both the fine-level matrix and the coarse-level matrix. Strictly speaking, problems with more complicated meshes would not fit into this framework. Although this does cover a lot of the territory in both practices and theories, it is not in the most generalized version. As such, future work can be devoted to generalizing our framework to unstructured meshes by using graph convolutional neural networks [17]. This will bring up many difficulties for sure. To name a few, one has to consider how complicated it is to set the meshes of the training and testing instances, how to choose the dataset, how to define the feature vector attached to each node, etc.

Chapter 8

Conclusion

In this work we propose a machine learning framework for sparsifying the coarse-grid operators $\boldsymbol{A}_g^{(l)}$ in multigrid methods to improve parallel efficiency. The sparsification process has two major steps: choosing the sparsity pattern and deciding the values. For each step we utilize one neural network and combine the results from the two steps to construct $\boldsymbol{A}_c^{(l)}$ which has much fewer number of non-zero entries than $\boldsymbol{A}_g^{(l)}$. We train the model on small-size problems while testing it on problems of much larger sizes, and by following multigrid convergence theory during the training process, our method does not trade algorithmic scalability with the convergence behaviour of the algorithm.

We conducted numerical experiments on rotated Laplacian problems and linear elasticity problems. We found that for both of the challenging problems, our framework can sparsify the coarse grid operator by at least 44% while maintaining the performance of the method. This marks an important step towards fully boosting the multigrid methods with the help of machine learning models. We also hope our method can provide new perspectives on how to improve classical solvers with modern data-driven techniques.

Bibliography

[1] O Axelsson. On multigrid methods of the two-level type. In *Multigrid methods*, pages 352–367. Springer, 1982.

[2] Nathan Bell, Luke N Olson, and Jacob Schroder. Pyamg: algebraic multigrid solvers in python. *Journal of Open Source Software*, 7(72):4142, 2022.

[3] Amanda Bienz, Robert D Falgout, William Gropp, Luke N Olson, and Jacob B Schroder. Reducing parallel communication in algebraic multigrid through sparsification. *SIAM Journal on Scientific Computing*, 38(5):S332–S357, 2016.

[4] Amanda Bienz, William D Gropp, and Luke N Olson. Reducing communication in algebraic multigrid with multi-step node aware communication. *The International Journal of High Performance Computing Applications*, 34(5):547–561, 2020.

[5] Matthias Bolten and Andreas Frommer. Structured grid amg with stencil-collapsing for d-level circulant matrices. 2007.

[6] Marian Brezina, Andrew J Cleary, Robert D Falgout, Van Emden Henson, Jim E Jones, Thomas A Manteuffel, Stephen F McCormick, and John W Ruge. Algebraic multigrid based on element interpolation (amge). *SIAM Journal on Scientific Computing*, 22(5):1570–1592, 2001.

[7] Marian Brezina, R Falgout, Scott MacLachlan, T Manteuffel, S McCormick, and

J Ruge. Adaptive smoothed aggregation (α sa). *SIAM Journal on Scientific Computing*, 25(6):1896–1920, 2004.

[8] William L Briggs, Van Emden Henson, and Steve F McCormick. *A multigrid tutorial*. SIAM, 2000.

[9] Robert D Falgout. An introduction to algebraic multigrid. Technical report, Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), 2006.

[10] Robert D Falgout and Jacob B Schroder. Non-galerkin coarse grids for algebraic multigrid. *SIAM Journal on Scientific Computing*, 36(3):C309–C334, 2014.

[11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[12] Anne Greenbaum. *Iterative methods for solving linear systems*. SIAM, 1997.

[13] Daniel Greenfeld, Meirav Galun, Ronen Basri, Irad Yavneh, and Ron Kimmel. Learning to optimize multigrid pde solvers. In *International Conference on Machine Learning*, pages 2415–2423. PMLR, 2019.

[14] Ru Huang, Ruipeng Li, and Yuanzhe Xi. Learning optimal multigrid smoothers via neural networks. *arXiv preprint arXiv:2102.12071*, 2021.

[15] Alexander Idesman and Bikash Dey. Compact high-order stencils with optimal accuracy for numerical solutions of 2-d time-independent elasticity equations. *Computer Methods in Applied Mechanics and Engineering*, 360:112699, 2020.

[16] Alexandr Katrutsa, Talgat Daulbaev, and Ivan Oseledets. Black-box learning of multigrid parameters. *Journal of Computational and Applied Mathematics*, 368:112524, 2020.

[17] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[18] Dmitry Kuznichov. Learning relaxation for multigrid. *arXiv preprint arXiv:2207.11255*, 2022.

[19] Jörg Liesen and Zdenek Strakos. *Krylov subspace methods: principles and analysis*. Oxford University Press, 2013.

[20] Ilay Luz, Meirav Galun, Haggai Maron, Ronen Basri, and Irad Yavneh. Learning algebraic multigrid using graph neural networks. In *International Conference on Machine Learning*, pages 6489–6499. PMLR, 2020.

[21] Yvan Notay. Convergence analysis of perturbed two-grid and multigrid methods. *SIAM journal on numerical analysis*, 45(3):1035–1044, 2007.

[22] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[23] Simon J.D. Prince. *Understanding Deep Learning*. MIT Press, 2023.

[24] Alfio Quarteroni, Riccardo Sacco, and Fausto Saleri. *Numerical mathematics*, volume 37. Springer Science & Business Media, 2010.

[25] Alfio Quarteroni and Alberto Valli. *Numerical approximation of partial differential equations*, volume 23. Springer Science & Business Media, 2008.

[26] Youcef Saad and Martin H Schultz. Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on scientific and statistical computing*, 7(3):856–869, 1986.

[27] Yousef Saad. *Iterative methods for sparse linear systems*. SIAM, 2003.

[28] Gilbert Strang. *Linear algebra and its applications*. Belmont, CA: Thomson, Brooks/Cole, 2006.

[29] Klaus Stüben. A review of algebraic multigrid. *Numerical Analysis: Historical Developments in the 20th Century*, pages 331–359, 2001.

[30] Ali Taghibakhshi, Scott MacLachlan, Luke Olson, and Matthew West. Optimization-based algebraic multigrid coarsening using reinforcement learning. *Advances in Neural Information Processing Systems*, 34:12129–12140, 2021.

[31] James William Thomas. *Numerical partial differential equations: finite difference methods*, volume 22. Springer Science & Business Media, 2013.

[32] Eran Treister and Irad Yavneh. Non-galerkin multigrid based on sparsified smoothed aggregation. *SIAM Journal on Scientific Computing*, 37(1):A30–A54, 2015.

[33] Henk A Van der Vorst. *Iterative Krylov methods for large linear systems*. Number 13. Cambridge University Press, 2003.

[34] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[35] Roman Wienands and Irad Yavneh. Collocation coarse approximation in multigrid. *SIAM Journal on Scientific Computing*, 31(5):3643–3660, 2009.

[36] Jinchao Xu and Ludmil Zikatanov. Algebraic multigrid methods. *Acta Numerica*, 26:591–721, 2017.