LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-SR-857062

Parallel Algebraic Multigrid for Fusion and Higher-Order PDEs

S. Boileau, A. Das, K. Kanarios, L. Krajcoviechova

November 10, 2023

Disclaimer

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

Research in Industrial Projects for Students

Sponsor

Lawrence Livermore National Laboratory

Final Report

Parallel Algebraic Multigrid for Fusion and Higher-Order PDEs

Student Members

Sophie Boileau (Project Manager), *Carleton College*,
boileaus@carleton.edu

Atmik Das, *University of California San Diego*

Kellen Kanarios, *University of Michigan*

Lucia Krajčoviechová, *University of Cambridge*

Academic Mentor

Jean-Michel Maldague, jmmaldague@gmail.com

Sponsoring Mentors

Dr. Robert Falgout, falgout2@llnl.gov
Dr. Wayne Mitchell, mitchell82@llnl.gov
Dr. Daniel Osei-Kuffuor, oseikuffuor1@llnl.gov

Date: August 17, 2023

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

Abstract

Multigrid methods play a key role in large-scale scientific simulation because they are among the fastest and most scalable approaches for solving the underlying sparse linear systems of equations that arise from a wide array of Partial Differential Equation (PDE) discretizations. Algebraic multigrid (AMG) is a special type of multigrid method that depends only on the description of the linear system, giving it better portability and broader applicability than geometric multigrid, as it requires no explicit knowledge of the problem geometry. Even though these methods are widely used today, there are still applications where further development is needed. In this report, we focus on PDEs with higher-order terms (e.g., fourth order), concentrating on a PDE that arises in tokamak edge plasma simulations (a tokamak is a machine that confines a plasma using magnetic fields and is believed to be the leading plasma confinement concept for future fusion power plants). General multigrid relaxes a linear system on coarser grids and reverses this process with interpolation, but standard AMG methods struggle with the aforementioned higher-order PDEs. We investigate cyclic coarsening and interpolation heuristics, as well as new iterative approximation methods of refining the solution at each grid to improve the existing multigrid approach. To this end, we ensure that these techniques are transferable to a parallelized setting with LLNL's supercomputers.

Acknowledgments

This research was jointly supported by the Center for Applied Scientific Computing (CASC) from Lawrence Livermore National Laboratory (LLNL) and Research in Industrial Projects for Students (RIPS) from the Institute for Pure and Applied Mathematics (IPAM). We would like to thank everyone at IPAM who supported us during this project. We would also like to thank, in particular, our sponsoring mentors Robert Falgout, Wayne Mitchell and Daniel Osei-Kuffuor, as well as our academic mentor Jean-Michel Maldague for all of the guidance and patience they have provided over the span of the project.

Contents

REFERENCES

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Lawrence Livermore National Laboratory

For over 70 years, Lawrence Livermore National Laboratory (LLNL) has conducted mission driven research in the areas of science and technology. In particular, they investigate fields including nuclear deterrence, threat preparedness and response, multi-domain deterrence, and climate and energy security. This project falls into the latter category of energy security, and it is overseen by the Center of Applied Scientific Computing (CASC) branch of LLNL. It is home to some of the world's fastest parallel computers, such as the upcoming El Capitan machine.

1.2 Motivation

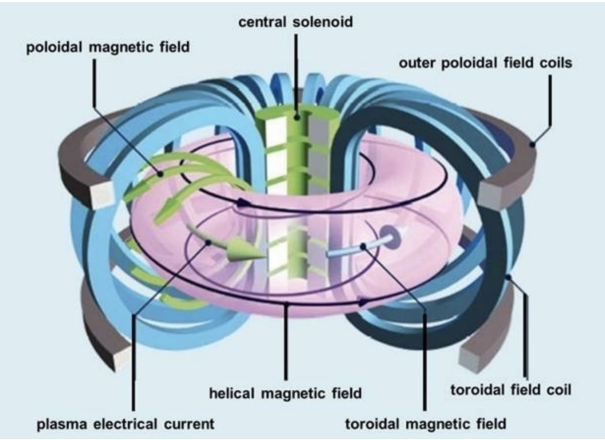


Figure 1.1: A labeled diagram of a tokamak along with its magnetic field components and plasma stream.

1.2.1 Thermonuclear Fusion and Tokamak

Our motivating examples for this project are edge plasma simulations of a tokamak for a thermonuclear fusion reaction. A tokamak is a machine that confines plasma using magnetic fields in a donut shape that scientists call a torus. It is believed to be the leading plasma confinement concept for future fusion power plants.

Under the influence of extreme heat and pressure, gaseous hydrogen fuel becomes an energized plasma, which provides an environment in which light atomic nuclei can fuse and yield energy. Magnetic coils on the outside of the tokamak cause plasma particles to spiral through it, resulting in collisions with other energized particles and breaking the electromagnetic repulsion between the positively charged nuclei. This releases high amounts of energy.

However, this also causes the flow of the plasma stream to be turbulent, especially at the edge. This is bad for the efficiency of the reactor, as one cannot have the particle trajectories interacting with the reactor wall. This motivates studies in kinetic edge models of plasma to understand the plasma confinement.

1.2.2 Fourth Order PDE and Linear Solvers

Edge plasma simulations face many challenges in terms of analytical and numerical analysis due to complex magnetic geometry, dimensionality, strong anisotropy, and multiple time scales. **CO**ntinuum **G**yrokinetic **E**dge **N**ew **T**echnology (COGENT) is a code that uses advanced numerical methods in order to overcome these challenges and analyze models with high efficiency.

Kinetic edge models give rise to complex differential equations, such as a fourth order Partial Differential Equation (PDE) of the form:

$$-au_{xx} - bu_{yy} + acu_{yyxx} = g, \quad b, c \gg a, \tag{1.1}$$

where u stands for the vorticity of the plasma.

In order to solve this PDE, we convert it into a linear form (explained in Chapter 3):

$$Au = f.$$

Pre-existing linear solvers that attempt to solve a discretization of this PDE fail at tackling the fourth order term, and such linear solvers are responsible for 60 to 70 percent of the computational cost when solving the higher-order PDEs for fusion. Solving this system by calculating A^{-1} is computationally expensive. We want to create a robust linear PDE solver that iteratively simplifies our linear system, leading to a solution in linear time. This leads us to our research in multigrid methods, namely, Algebraic MultiGrid (AMG). The COGENT code uses linear solvers such as AMG as preconditioners for iteratively solving a much more complex system pertaining to the model.

1.3 Overview of the Report

We begin the remainder of this report with Chapter 2, which details the previous and related work for our project. Next, Chapter 3 provides a comprehensive background on the mathematical basis of the problem, followed by our algorithmic advancements in Chapter 4. Next, we illustrate our advancements in the Results section (Chapter 5). Finally, we conclude in Chapter 6 by summarizing our advancements and recommendations for future directions of this problem.

Chapter 2

Literature Review

Nuclear and plasma physics play an important role in inspiring research on solving higher order PDEs, especially those resulting from tokamak edge plasma simulations [12, 18]. These works explain the physical and numerical challenges faced with confining the plasma inside a tokamak and studying the behavior of the plasma stream. The former publication discusses two approaches to solving a kinetic equation: a Particle In Cell (PIC) approach and a Continuum approach using higher order finite volume discretization. However, the PIC approach suffers from statistical particle noise and limited phase space density resolution. Therefore, the Continuum approach is a more explored avenue, and our project uses finite difference discretization for a simplified version of the gyrokinetic model explained by it.

In this report, we will utilize finite difference discretization to convert our higher order PDE into a system of linear equations (see [28] for the discretizations for 1D and 2D-Poisson Models). To solve this system, the most basic class of algorithms are known as iterative methods. Iterative methods involve decomposing our matrix into easy to solve components and updating our solution with the result of solving the easier system [23, 8]. However, as we will see in Chapter 3, these methods struggle to eliminate near kernel components in the error vector.

From the shortcomings of iterative methods, multigrid methods have arisen. Now multigrid methods are well known for being the fastest numerical methods for solving elliptic boundary-value problems [28]. These methods were developed with the main intention to solve systems corresponding to discretized PDEs. From this, they can take advantage of the structure, mainly sparsity, of the system to solve them in as fast as $O(n)$ time. Concretely, they can solve a system with n unknowns in only $O(n)$ work. Highly parallelizable versions of these algorithms [2] can solve incredibly large problems in nearly constant time.

At a high level, multigrid methods employ iterative methods to get rid of the "easy" error. They then correct the problematic near-kernel error via coarse-grid correction. The idea is to exactly solve for the error on a representative smaller system then interpolate that error back to the finer grid.

One major limitation of traditional multigrid methods is that they require extensive information about the underlying problem. In particular, knowledge of the mesh is required in almost every method. For alternative discretization schemes, such as finite elements, we may not have an associated geometrically logical mesh. This makes many of these existing methods unusable. However, Algebraic MultiGrid (AMG) tries to remedy this issue by applying traditional multigrid techniques while only requiring the linear system itself as input. AMG has been around for awhile [6, 4, 25]. Here, the authors were able to develop a heuristic to identify how to coarsen entirely from the matrix entries. To do this, they rely

on a strength measure, signifying how strongly connected two components are with respect to the matrix. From this, there have been many alternative strength measures introduced [7, 22, 26]. Other approaches to purely "algebraic" methods of solving these systems are known as compatible relaxation [8, 3]. Here they use the components of the error vector on each grid level to adapt their algorithm.

These AMG methods rely on the crucial assumption that the near-kernel error is geometrically smooth. This means that the coarse-grid correction would only need to solve for smooth error that can be easily interpolated. In an anisotropic case, a single direction of geometric smoothness is enough for AMG to be efficient. However, in this report, there are instances of no direction of smoothness. This means that existing AMG methods will break down, as we show later. From this, we must employ stronger smoothing techniques, such as line relaxation [5]. We also utilize more strategic semi-coarsening to deal with the anisotropy in the diffusion term. How to do this completely algebraically remains an open question [15, 20]. However, we propose a hybrid algorithm that constructs interpolation and coarse grids algebraically, while requiring the grid for semi-coarsening and line relaxation.

Chapter 3

Mathematical Background

3.1 Discretization of Differential Equations

In this section, we will cover the finite differences scheme for discretizing PDEs. This is the scheme that will be used for the PDE of interest.

Discretization of 1D Poisson Equation

First consider the following problem

$$-u_{xx} = f \text{ on } \Omega = [0,1], \tag{3.1}$$
$$u = g \text{ on } \Gamma = \partial\Omega = \{0,1\}, \tag{3.2}$$

where u_{xx} is a shorthand notation for $\frac{\mathrm{d}^2 u}{\mathrm{d}x^2}$. Let $n \in \mathbb{N}$, and for $0 \leq i \leq n$ define $x_i = \frac{i}{n}$. We would like to find $u_i := u(x_i)$ for all i. From Equation (3.2), we get $u_0 = g(0)$ and $u_n = g(1)$. Note

$$u'\big|_x = \lim_{h\to 0} \frac{u(x+h) - u(x)}{h} \approx \frac{u(x+h) - u(x)}{h}$$

for h sufficiently small. If we take $h > 0$, we call it the *forward difference*, if $h < 0$, it is the *backward difference*, and if we take the average of the two (with adjusted spacing of $\frac{h}{2}$ instead of h), we get

$$u'\big|_x \approx \frac{1}{2}\left(\frac{u(x+\frac{h}{2}) - u(x)}{\frac{h}{2}} + \frac{u(x-h) - u(x)}{-\frac{h}{2}}\right) = \frac{u\left(x+\frac{h}{2}\right) - u\left(x-\frac{h}{2}\right)}{h},$$

which is called the *central difference*. Thus, if n is sufficiently large, we can take $h = \frac{1}{n}$ to get

$$u'\left(x_i + \frac{h}{2}\right) \approx \frac{u\left(x_i + \frac{h}{2} + \frac{h}{2}\right) - \left(x_i + \frac{h}{2} - \frac{h}{2}\right)}{h} = \frac{u(x_{i+1}) - u(x_i)}{h},$$

and similarly

$$u'\left(x_i - \frac{h}{2}\right) \approx \frac{u(x_i) - u(x_{i-1})}{h}.$$

Substituting these into the central difference for u'', we get

$$\begin{aligned} u''(x_i) &\approx \frac{u'\left(x_i + \frac{h}{2}\right) - u'\left(x_i - \frac{h}{2}\right)}{h} \\ &\approx \frac{1}{h}\left(\frac{u(x_{i+1}) - u(x_i)}{h} - \frac{u(x_i) - u(x_{i-1})}{h}\right) \\ &\approx \frac{1}{h^2}\left(u(x_{i+1}) - 2u(x_i) + u(x_{i-1})\right). \end{aligned} \tag{3.3}$$

Thus, for each $0 < i < n$, we have

$$-f(x_i) = -u''(x_i) \approx \frac{1}{h^2}\left(-u(x_{i-1}) + 2u(x_i) - u(x_{i+1})\right).$$

These equations for all i can also be written in matrix form as

$$\underbrace{\begin{pmatrix} 2 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & -1 & 2 & -1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & -1 & 2 & -1 \\ & & & & -1 & 2 \end{pmatrix}}_{A} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_{n-2} \\ u_{n-1} \end{pmatrix} \approx \underbrace{-h^2 \begin{pmatrix} f(x_1) \\ f(x_2) \\ f(x_3) \\ \vdots \\ f(x_{n-2}) \\ f(x_{n-1}) \end{pmatrix} + \begin{pmatrix} g(0) \\ 0 \\ 0 \\ \vdots \\ 0 \\ g(1) \end{pmatrix}}_{f_d}.$$

We call the matrix on the left-hand side A and the vector on right-hand side f_d. We can also write A in *stencil* notation as

$$A \sim \begin{bmatrix} -1 & 2 & -1 \end{bmatrix},$$

where the stencil represents each row of the matrix A. From now on, we will use parentheses for matrices and square brackets for stencils. Now, we can write

$$A \begin{pmatrix} u_1 \\ \vdots \\ u_{n-1} \end{pmatrix} = f_d,$$

which is the *discretization* of Equation (3.1) with boundary conditions from Equation (3.2). We call A the *discretization operator* of $\frac{\mathrm{d}^2}{\mathrm{d}x^2}$, and multiplying a 'discretized function' by A on the left corresponds to taking the derivative of the function.

Discretization of a 2D PDE

Recall Equation (1.1):

$$-au_{xx} - bu_{yy} + acu_{yyxx} = g, \quad b, c \gg a.$$

In order to discretize this PDE, we proceed similarly as in the 1D case. We choose $n, m \in \mathbb{N}$ and let $x_{i,j} = \left(\frac{i}{n}, \frac{j}{m}\right)$ for $0 \le i \le n$, $0 \le j \le m$. We are interested in the values of $u_{i,j} = u(x_{i,j})$ for all i, j. Using the same method as in the previous section, we get

$$\frac{\partial^2}{\partial x^2} \approx M_x \sim \begin{bmatrix} -1 & 2 & -1 \end{bmatrix}$$

and also

$$\frac{\partial^2}{\partial y^2} \approx M_y \sim \begin{bmatrix} -1 \\ 2 \\ -1 \end{bmatrix},$$

where we have used a 2D stencil notation. This just means that M_y is the matrix which, if multiplied by the vector with entries $u_{i,j}$ for all $0 < i < n,\ 0 < j < m$ (in a lexicographical order), corresponds to the vector with entries

$$-u_{i,j+1} + 2u_{i,j} - u_{i,j-1}.$$

For the fourth-order term u_{yyxx}, we have

$$\left.\frac{\partial^2}{\partial y^2}\frac{\partial^2 u}{\partial x^2}\right|_{x_{i,j}} \approx -\frac{1}{h^2}\left(\left.\frac{\partial^2 u}{\partial x^2}\right|_{x_{i,j-1}} - 2\left.\frac{\partial^2 u}{\partial x^2}\right|_{x_{i,j}} + \left.\frac{\partial^2 u}{\partial x^2}\right|_{x_{i,j+1}}\right).$$

Applying Equation (3.3) again, for the x-derivative, we get

$$\frac{\partial^2}{\partial y^2}\frac{\partial^2}{\partial x^2} \approx N = M_y M_x \sim \begin{bmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{bmatrix}.$$

Then the discretization of Equation (1.1) is

$$Au_d = f_d,$$

where this time we have

$$A = -aM_x - bM_y + acN,$$

$$u_d = \begin{pmatrix} u_{1,1} \\ \vdots \\ u_{1,m-1} \\ u_{2,1} \\ \vdots \\ u_{n-1,m-1} \end{pmatrix}, f_d = -h^4 \begin{pmatrix} f(x_{1,1}) \\ \vdots \\ f(x_{1,m-1}) \\ f(x_{2,1}) \\ \vdots \\ f(x_{n-1,m-1}) \end{pmatrix} + g_B,$$

with g_B being the 'correction' for points close to boundary, i.e. those with $i = 1$, $j = 1$, $i = n - 1$ or $j = m - 1$.

3.2 Iterative Methods

Let $A \in \mathbb{R}^{n\times n}$ and $f \in \mathbb{R}^n$, where A is symmetric and positive (semi-) definite. We want to find $u \in \mathbb{R}^n$ such that

$$Au = f.$$

First, we will consider a simple approach known as an iterative method. We start with some initial guess $u^{(0)}$ and successively construct better approximations $u^{(1)}, u^{(2)}, \ldots$, which will hopefully converge to u. Let

$$e^{(t)} = u - u^{(t)}$$

be the *error* and

$$r^{(t)} = f - Au^{(t)} = A\left(u - u^{(t)}\right) = Ae^{(t)}$$

be the *residual*.

For any matrix M, we have

$$\begin{aligned} f = (M + (A - M))\, u &\approx Mu^{(t+1)} + (A - M)\, u^{(t)}, \\ f - Au^{(t)} &\approx M\left(u^{(t+1)} - u^{(t)}\right). \end{aligned}$$

If we choose M to be invertible, then we can define

$$u^{(t+1)} := u^{(t)} + M^{-1}\left(f - Au^{(t)}\right). \tag{3.4}$$

Considering the update rule in Equation (3.4), we have

$$\begin{aligned} e^{(t+1)} &= u - u^{(t+1)}, \\ &= u - \left(u^{(t)} + M^{-1}\left(f - Au^{(t)}\right)\right), \\ &= \left(u - u^{(t)}\right) - M^{-1}A\left(u - u^{(t)}\right), \\ &= e^{(t)} - M^{-1}Ae^{(t)}, \\ &= \underbrace{\left(I - M^{-1}A\right)}_{E} e^{(t)}. \end{aligned} \tag{3.5}$$

We call E the *error propagation* operator of the iterative method. If we decompose $A = L + D + U$, where L is the strictly lower triangular, D is the diagonal, and U is the strictly upper triangular part of A, then taking

- $M = D$ results in the *Jacobi* method,
- $M = L + D$ is the *forward Gauss-Seidel* method,
- $M = D + U$ is the *backward Gauss-Seidel* method.

Moreover, taking one forward Gauss-Seidel iteration followed by one backward Gauss-Seidel iteration is also known as *symmetric Gauss-Seidel* iteration.

3.2.1 Issue with Iterative Methods

Since we assume that A is a symmetric positive (semi-) definite matrix, by the Spectral Theorem, we can find an orthogonal basis $v^{(1)}, \ldots, v^{(n)}$ of $\mathbb{R}^n$ such that each $v^{(i)}$ is an eigenvector of A with eigenvalue λ_i. Therefore, we can write

$$e^{(t)} = \sum_{i=1}^{n} c_i^{(t)} v^{(i)}$$

for some $c_1^{(t)}, \ldots, c_n^{(t)} \in \mathbb{R}$. From Equation (3.5), our error propagation for Jacobi becomes

$$\begin{aligned} e^{(t+1)} &= (I - D^{-1}A) \sum_{i=1}^{n} c_i^{(t)} v^{(i)} \\ &= \sum_{i=1}^{n} c_i^{(t)} \left(v^{(i)} - D^{-1}\lambda_i v^{(i)} \right) \\ &= \sum_{i=1}^{n} c_i \begin{pmatrix} \left(1 - \frac{\lambda_i}{d_1}\right) v_1^{(i)} \\ \vdots \\ \left(1 - \frac{\lambda_i}{d_n}\right) v_n^{(i)} \end{pmatrix}, \end{aligned}$$

where $d_1, \ldots, d_n$ are the entries of D on the diagonal.

Naturally, we want our error propagation to converge to 0. However, if there is some large positive λ_j (specifically, $\lambda_j \geq 2d_j$), then the $\left(1 - \frac{\lambda_j}{d_j}\right)$ term will be less than -1. Given that the error propagation can be viewed as a geometric series, its rate exceeds 1 in magnitude. Therefore, it will not converge, which presents an obstacle.

To remedy this problem and ensure that error will converge to 0, we modify our iterative method to a weighted Jacobi approach. This involves choosing $\omega \in [0, 1]$ and taking

$$u^{(t+1)} = (1 - \omega)u^{(t)} + \omega D^{-1} \left(f - (A - D)u^{(t)} \right).$$

Note that setting $\omega = 1$ is equivalent to the original form of Jacobi. Our error propagator becomes

$$\begin{aligned} e^{(t+1)} &= u - u^{(t+1)} \\ &= (1 - \omega) \left(u - u^{(t)} \right) + \omega \left(u - D^{-1} \left(f - (A - D)u^{(t)} \right) \right) \\ &= (1 - \omega)e^{(t)} + \omega \left(u - D^{-1} \left(Au - Au^{(t)} \right) - u^{(t)} \right) \\ &= (1 - \omega)e^{(t)} + \omega \left(e^{(t)} - D^{-1}Ae^{(t)} \right) \\ &= \left(I - \omega D^{-1}A \right) e^{(t)}. \end{aligned}$$

As long as we choose ω to be $0 < \omega < 2/(\text{largest eigenvalue of } D^{-1}A)$, we obtain convergence. This solves the problem of large eigenvalues, but there are other scenarios that warrant further attention.

Alternatively, suppose that there exists some unique $\lambda_j = 0$. Then as $t \to \infty$, $e_t \to c_j v_j$. Thus, *small eigenvectors* [1] become an issue: to reduce the error for these small eigenvalues, we need to perform many iterations. To remedy this, we attempt to manually force-correct this error through a method called multigrid.

Definition 1. *Let $v^{(1)}, \ldots, v^{(m)}$ be all the eigenvectors of A such that $\max_{i \in [m]} \lambda_i < \epsilon$. Then, we define the ϵ-near-kernel as span$(v^{(1)}, \ldots, v^{(m)})$, denoted $N_\epsilon(A)$.*

Corollary 1. *Fix $\epsilon > 0$. For any iterative method $\mathcal{A}$ with error propagation of the form of Equation (3.5), there exists some T_δ such that if we run $\mathcal{A}$ for T_δ iterations then*

$$\left\| e^{(T_\delta)} - v \right\| \leq \delta$$

for some $v \in N_\epsilon(A)$.

[1] Here, we say that a small eigenvector is an eigenvector corresponding to a small eigenvalue.

3.3 Multigrid Methods

In the previous section, we explored iterative methods and saw how they may struggle for eigenvectors with small eigenvalues. The most commonly used method in practice is weighted Jacobi [28], a slightly better alternative to the method presented in Section 3.2. However, this leaves us with the following issue: we have the system

$$Au = f,$$

where it is computationally infeasible to compute A^{-1}, and iterative methods take too long to eliminate small eigenvectors. As a technical point, we will instead consider the system

$$Ae^{(t)} = r^{(t)},$$

where $r^{(t)}$ is the residual $r_t = Au - Au_t$. Note that

$$u^{(t+1)} = u^{(t)} + e^{(t)} = u^{(t)} + \left(u - u^{(t)}\right) = u,$$

so this newer problem is equivalent to solving the original system. To formalize the notion of small eigenvectors, we introduce the following terminology.

From the previous section, $\exists T_\delta \in \mathbb{N}$ such that after T_δ iterations of our iterative method $\left\|e^{(T_\delta)} - v\right\| < \delta$ for some $v \in N_\epsilon(A)$. This process of "smoothing" the error is known as *relaxation*. Suppose that we could construct some $A_c \in \mathbb{R}^{m\times m}$ and $P \in \mathbb{R}^{n\times m}$, such that for the solution, e_c, to the system

$$A_c e_c = r_c,$$

we have that $Pe_c = v$. The process of converting from A (a fine grid) to a coarser grid A_c, and transferring the solution of this system back to the initial system is called *coarse grid correction*. Introducing the update rule (this process is called *restriction*) [13],

$$u^{(t+1)} = u^{(t)} + Pe_c,$$

we have that

$$\left\|e^{(t+1)}\right\| = \left\|u - u^{(t+1)}\right\| = \left\|u - (u^{(t)} + Pe_c)\right\| = \left\|e^{(t)} - Pe_c\right\| \leq \delta.$$

Note that this algorithm can be implemented recursively. Therefore, as the iterations progress, we solve smaller and smaller systems with every restriction. Overall, this would allow us to run our iterative method for a constant number of iterations and avoid solving a computationally expensive system by solving a much smaller one. This is the main idea behind multigrid. However, we are still left with many questions. First, how do we construct such a P? For this, see Section 3.4. Second, given P, how can we construct a suitable coarse grid A' and solution r_t'? We will explore this now.

In the literature, P is known as an *interpolation operator*, where $P \in \mathbb{R}^{n\times m}$ for $m < n$. *Interpolation* is the process of transferring the solution from the coarse grid A' back to the fine grid A [13]. Recall that our main objective is to minimize $\|u^{(t+1)} - u\|$. Because we cannot calculate u exactly, we use the residual as a proxy for our objective. Note that

$$r^{(t+1)} = Au - Au^{(t+1)} = Au - A(u_t + Pe_c) = \underbrace{A(u - u_t) - Ae_t}_{0} + A(e_t - Pe_c) = A(e_t - Pe_c).$$

Therefore, we want the solution of our system to be such that $A(e_t - Pe_c)$ is minimized. Now we can introduce the A-norm as follows:

$$\|v\|_A = v^T Av.$$

Note that for this to be a norm, A must be positive definite. If we once again utilize our eigenbasis of A, we have that

$$\|v\|_A = \sum_{i=1}^{n} c_i^2 \lambda_i,$$

where $v = \sum_{i=1}^{n} c_i v_i$. If our system has a unique solution, then each $\lambda_i > 0$. Therefore, minimizing $\|v\|_A$ is equivalent to minimizing the coefficients c_i.

Corollary 2. *If A is a positive (semi-) definite matrix, then*

$$v^T Av = 0 \implies Av = 0.$$

Furthermore, if A is positive definite, then

$$\|v\|_A = 0 \implies v = 0.$$

The proof of this can be found in Appendix A.1. This means that if we want to find v such that each entry of Av is small, then a good approach would be to minimize the A-norm. For our purposes, then, a good idea would be to solve the following

$$\min_{e_c} \|e_t - Pe_c\|_A . \tag{3.6}$$

Proposition 1. *Let $e_t = u - u_t$, which implies $Ae_t = r_t$ as above. The solution e_c to the following system*

$$P^T APe_c = P^T r_t$$

is also the solution to Equation (3.6).

To see this, solve Equation (3.6) analytically. The full derivation can be found in Appendix A.2. The result of Proposition 1 is a smaller system, namely,

$$P^T APe_c = P^T r_t \tag{3.7}$$

such that with a good choice of P,

$$r^{(t+1)} = A\left(e_t - Pe_c\right)$$

should be small. An important technical distinction is that if A is positive (semi-) definite, then so is $P^T AP$. This allows us to solve Equation (3.7) recursively, providing us with the *V-Cycle* algorithm [28].

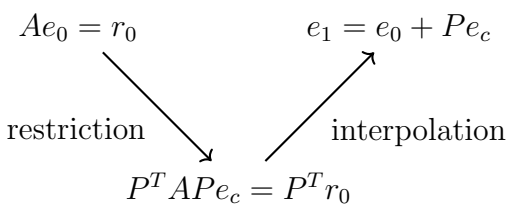


Figure 3.1: Illustration behind the name V-Cycle with only one recursive call.

In practice, we run some parameter number of V-cycles or until reaching some suitable stopping criterion. Within each V-Cycle, we recursively restrict our system until we are left with a 2×2 system [2] to solve. This results in the full multi grid algorithm (see Algorithm 1).

Algorithm 1: Full Multigrid Algorithm

input : matrix $A \in \mathbb{R}^{n \times n}$, right-hand-side vector $f \in \mathbb{R}^n$, # of relaxation steps on the "down cycle" v_1, # of relaxation steps on the "up cycle" v_2, # of V-Cycles μ

output: vector $u \in \mathbb{R}^n$

$u_0 \leftarrow$ *random initialization*;
$r_0 \leftarrow f - Au_0$;
for $t = 1, \ldots, \mu$ **do**
  $u^{(t)} \leftarrow$ `VCycle(`A, f, v_1, v_2`)`;
end
return u_t

3.4 Finding a Good Interpolation Operator

As a toy example, we will consider the following differential operator

$$Lu = -\frac{\mathrm{d}^2 u}{\mathrm{d}x^2} = f.$$

Discretizing this operator with finite differences and using Dirichlet boundary conditions yields

$$L = \begin{pmatrix} 2 & -1 & & & & & \\ -1 & 2 & -1 & & & & \\ & -1 & 2 & -1 & & & \\ & & -1 & 2 & -1 & & \\ & & & -1 & \ddots & \ddots & \\ & & & & \ddots & \ddots & -1 \\ & & & & & -1 & 2 \end{pmatrix}.$$

Next, we have

$$L \sim \begin{bmatrix} -1 & 2 & -1 \end{bmatrix}.$$

To rigorously justify future discussion, we must introduce the induced norm of a matrix.

Definition 2. *Let $A \in \mathbb{R}^{n \times m}$. Then the induced p-norm of A is defined as*

$$\|A\|_p = \sup_{v \in \mathbb{R}^m} \frac{\|Av\|_p}{\|v\|_p}.$$

As stated above, our objective is to find u such that $Lu = f$.

[2] On one CPU we recurse until $N = 2$. However, with parallelization of row-wise solving, we can stop recursion early with access to more computational resources.

Proposition 2. *Let $A \in \mathbb{R}^{n\times m}$ and $v \in \mathbb{R}^m$. For every $\epsilon > 0$, there exists some $\mathcal{E}(\epsilon)$ such that if*

$$\|v\|_2 \leq \mathcal{E}(\epsilon)$$

then we have that $\|v\|_A \leq \epsilon$.

The proof makes use of the sub-multiplicative property of the induced norm. See Appendix A.3 for the full proof.

Let $\mathcal{E}(\delta)$ be that of Proposition 2. Introducing the notation[3] e_t^N, from Corollary 1, we can pick N_δ so that $\forall t \geq 0$, $\left\|e_t^{N_\delta} - v\right\|_2 \leq \mathcal{E}(\delta)$, where $v \in N_\epsilon(A)$. Consequently, our objective in defining P is so that $v \in \text{range}(P)$. As justification, from Proposition 1 we have

$$e_c = \underset{e_c}{\text{argmin}} \left\|e_t^{N_\delta} - Pe_c\right\|_A.$$

Since $v \in \text{range}(P)$, $\exists x \in \mathbb{R}^m$ such that $Px = v$. Applying Proposition 2,

$$0 \leq \left\|e_t^{N_\delta} - Pe_c\right\|_A \leq \left\|e_t^{N_\delta} - Px\right\|_A = \left\|e_t^{N_\delta} - v\right\|_A \leq \delta. \tag{3.8}$$

From Corollary 2 as $\delta \to 0$, we have that $r_t = A(e_t - Pe_c) \to 0$. If A is full rank then $\|(u_t + Pe_c) - u\| \to 0$, as desired. Note that $\delta \to 0$ is equivalent to $e_t \to v$ for some $v \in N_\epsilon(A)$. We know that this will happen by Corollary 1.

Now, we cannot ensure that $v \in \text{range}(P)$. However, we do aim to get reasonably close. To do this, we require a key observation. Recalling that the origin of our system corresponds to a differential operator, it is reasonable to assume that its near-kernel components consist of "smooth functions." This is because the higher-order derivatives of most smooth functions vanish. Take, for example, the constant function represented as $\mathbf{1}$. It is easy to see that $L \cdot \mathbf{1} = 0$, as the second derivative of a constant function is zero. The major benefit of these near-kernel (as defined in Definition 1) components being smooth is that we can utilize interpolation to get good approximations of these components. In other words, we represent the corresponding function with much fewer degrees of freedom. Given knowledge of the underlying PDE, we can interpolate strategically to target these near-kernel components. However, in AMG, we aim to do this using only the entries of the matrix A.

3.4.1 Strength of Connection

In classical AMG [4], we attempt to leverage what is called strength of connection [13]. Once again consider N_δ from above, so that $\|e_t^{N_\delta} - v_i\|_2 < \mathcal{E}(\delta)$. By the triangle inequality, we have that

$$\left\|e_t^{N_\delta}\right\|_A \leq \|v_i\|_A + \left\|e_t^{N_\delta} - v_i\right\|_A \leq \|v_i\|_A + \delta = \lambda_i \|v_i\|_2 + \delta \ll 1.$$

We will also assume that A has zero row sum. This can be easily justified because multigrid methods are intended for differential operators. Through algebraic manipulation, it is possible to show that

$$\left\|e_t^{N_\delta}\right\|_A = \left(e_t^{N_\delta}\right)^T A e_t^{N_\delta} = \sum_{i<j} -a_{ij}(e_i - e_j)^2 \ll 1. \tag{3.9}$$

[3] Due to the varying number of iterations between relaxation steps and V-Cycles, e_t^N will be the remaining error after t V-cycles, where the relaxation step runs for some fixed N iterations.

[4] Classical AMG assumes geometrically smooth functions.

The derivation can be found in Appendix A.4. From this, we deduce that it is likely that error varies slowly in the direction of large negative matrix entries. Therefore, a good heuristic may be to coarsen in the direction of the components corresponding to these matrix entries. This leads us to the classical definition of strength:

Definition 3. *A component u_i is θ-strongly connected with a component u_j if*

$$-a_{ij} \geq \theta \max_{k \neq j}\{-a_{ik}\}.$$

It is important to note that this definition is not necessarily symmetric. For now, it will be our definition, even though there are many possible alternatives. As a brief recap, we want an operator P such that $v \in \text{range}(P)$ or something somewhat close to v. We have identified matrix entries that correspond to "smoothness" in the error, and from our intuition of differential operators, we can reasonably infer that v will be "smooth." Therefore, we hope that these entries correspond to important entries in our near-kernel components. From this, we develop another heuristic:

1. Construct a strength matrix A_s by removing all weak connections.
2. Partition the points in the strength matrix into C-points and F-points, where C-points are the points that will be in our coarser grid.
3. Construct the interpolation operator by solving for the F-points in terms of the C-points.

The first two steps are clear. However, the last step is the last key point of classical AMG.

3.4.2 Interpolation

To solidify the idea of interpolation, we consider a simple example. Let $u \in \mathbb{R}^3$ be the discretization of the linear function $f(x) = x$ on the grid $t = \{1, 3, 5\}$ – i.e.

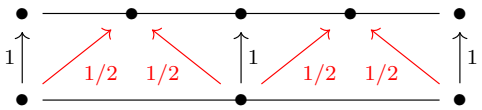
$$u = \begin{pmatrix} 1 & 3 & 5 \end{pmatrix}^T.$$

Then, if we want to interpolate our function onto the finer grid $t' = \{1, \ldots, 5\}$, we want some P such that

$$Pu = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \end{pmatrix}^T.$$

For this, we can utilize "local averaging" where we essentially just average the nearest points.

$$P = \begin{pmatrix} 1 & 0 & 0 \\ 1/2 & 1/2 & 0 \\ 0 & 1 & 0 \\ 0 & 1/2 & 1/2 \\ 0 & 0 & 1 \end{pmatrix} \quad \rightarrow$$



For AMG, the main objective of interpolation is to reconstruct a good approximation for the smooth error on the finer grid from the points in our coarse grid. However, in AMG, we have no notion of points that are "local" to one another. Therefore, we need to utilize our strength of connection heuristic to try to uncover this crucial information. Let v_s be the smallest eigenvector of A. Now pick N_δ, such that $\left\| e_t^{N_\delta} - v_s \right\|_2 \leq \delta / \|A\|_2$, where $\|A\|_2$ is the 2-induced norm (see Definition 2).

Lemma 3.1. *Let $A \in \mathbb{R}^{n\times m}$ and $v \in \mathbb{R}^m$. Then*

$$\|Av\| \leq \|A\| \cdot \|v\|,$$

where $\|A\|$ is the induced matrix norm.

This is a well-known result, so the proof will be omitted. Note that

$$\|r_t\|_2 = \left\|Ae_t^{N_\delta}\right\| \leq \left\|Ae_t^{N_\delta} - Av_s\right\| + \|Av_s\|.$$

Applying Lemma 3.1,

$$\left\|Ae_t^{N_\delta} - Av_s\right\|_2 + \|Av_s\|_2 \leq \|A\|_2 \left\|e_t^{N_\delta} - v_s\right\|_2 + \|Av_s\|_2 \leq \delta + \lambda \|v_s\|_2 \ll 1.$$

Thus, $\left\|r^{(t)}\right\| \ll 1$. Therefore, we can conclude that smooth error is also characterized by small residuals. As a heuristic, we will assume $r = 0$ and solve

$$Ae^{(t)} = 0. \tag{3.10}$$

For interpolation, we want to find a relationship for each of the fine points in terms of only the coarse grid points. To do this, we can rewrite (3.10) in terms of our strength matrix partition.

$$a_{ii}e_i = -\sum_{j\in C_i} a_{ij}e_j - \sum_{j\in F_i^s} a_{ij}e_j - \sum_{j\in N_i^w} a_{ij}e_j,$$

where

- C_i: C-points strongly connected with i,
- F_i^s: F-points strongly connected with i,
- N_i^w: points weakly connected with i - meaning that $a_{ij} \neq 0$, but j is not strongly connected to i.

Now for each $j \notin \{i, C_i\}$, we can rewrite e_j only in terms of points e_k, such that $k \in \{i, C_i\}$. Concretely,

$$e_j = \begin{cases} \sum_{k\in C_i} \left(\frac{a_{jk}}{\sum_{l\in C_i} a_{jl}}\right) e_k, & j \in F_i^s \\ e_i, & j \in N_i^w \end{cases}. \tag{3.11}$$

This gives us the interpolation operator

$$P_{ij} = -\left(a_{ij} + \sum_{k\in F_i^s} \left(\frac{a_{kj}}{\sum_{l\in C_i} a_{kl}}\right) a_{ik}\right) / \left(a_{ii} + \sum_{k\in N_i^w} a_{ik}\right), \tag{3.12}$$

where P_{ij} is the ijth entry of our interpolation operator. From Equation (3.11), we recover a "weighted" local averaging, where the finer points are interpolated by the "local" strongly connected points from the coarser grid.

3.5 Putting It All Together: An Example

Let us once again consider the discretization of the Laplacian operator with stencil

$$L \sim \begin{bmatrix} -1 & 2 & -1 \end{bmatrix}$$

on the grid $t = 1, \ldots, 5$. Since every entry on the off diagonal is negative of the same magnitude, our strength matrix will include every entry of A. To depict this, we would have the graph shown in Figure 3.2 below.

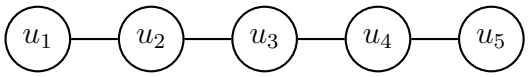


Figure 3.2: Graph representation of the strength matrix A_s for the 1-D Laplacian operator.

Now as a heuristic, we will choose our C-points as follows, which is diagramed in Figure 3.3.

1. Label each node with their measure of strength,
    - As a measure, we will just use the number of strong connections.
2. Pick the node with the highest measure as a C-point,
3. Pick all connected nodes to the selected C-point as F-points,
4. Increment the measure of all remaining points by the number of connections they had with recently allocated points,
5. Continue until every point is chosen as either an F-point or C-point.

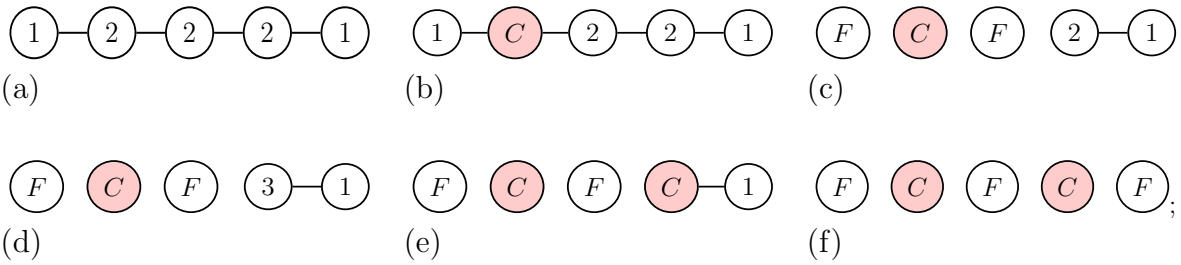


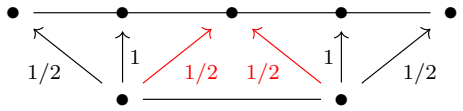
Figure 3.3: Partitioning the grid $t = 1, \ldots, 5$ for the 1-D Laplacian operator. (a) The algorithm assigns to the nodes of the strength matrix graph a weight equal to the number of off-diagonal connections. (b) A point with maximal weight is chosen as a C-point. (c) The algorithm sets the neighbors of the new C-point to be F-points. (d) For each new F-point, the algorithm increases its neighbors' weights by one to make them more likely to be chosen next. The algorithm continues in this way (e)-(f) until all points are either C- or F-points.

From this, we would get the partition $C = \{u_2, u_4\}$ and $F = \{u_1, u_3, u_5\}$ (see the figure above). With our partition, we can now construct our interpolation operator P. Using Equation (3.12), we have

$$u_1 = \frac{1}{2}u_2, \quad u_3 = \frac{u_2 + u_4}{2}, \quad u_5 = \frac{1}{2}u_4.$$

In this case, it turns out that we end up with a local averaging interpolation similar to the one proposed in Section 3.4.2. However, this is because the strongly connected matrix entries were the same magnitude. If we choose a sufficiently small θ this will most likely not be the case. We leave it as an exercise for the reader to try more complicated problems. Concretely,

$$P = \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 1/2 & 1/2 \\ 0 & 1 \\ 0 & 1 \end{pmatrix} \quad \rightarrow \quad$$



Note that the term from (3.10)

$$\sum_{k \in F_i^s} \left(\frac{a_{kj}}{\sum_{l \in C_i} a_{kl}} \right) = 0,$$

further illustrating the simplicity of this problem.

Chapter 4

Algorithm

4.1 PDE Revisited

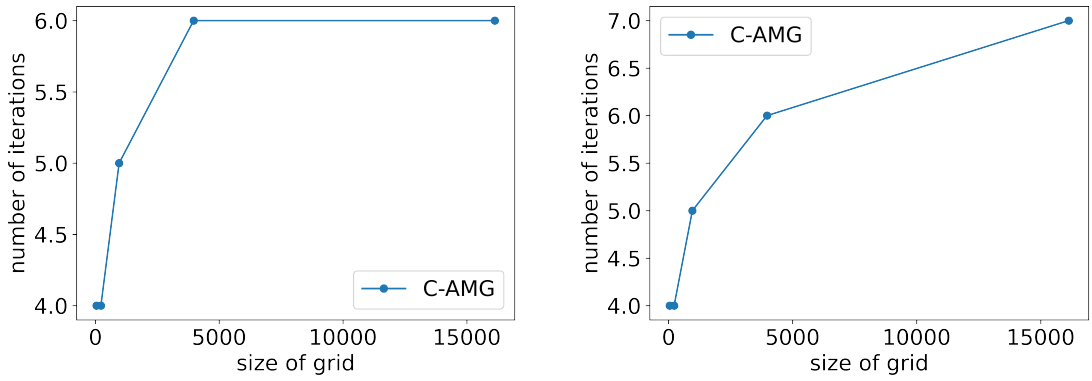
Classical AMG methods thrive with standard diffusion problems, which resemble the COGENT problem – with one major exception. These PDEs only contain second order terms. For instance, consider equations

$$-u_{xx} - u_{yy} = 0 \tag{4.1}$$

and

$$-\epsilon u_{xx} - u_{yy} = 0, \tag{4.2}$$

with $\epsilon = 10^{-5}$. As plotted below in Figure 4.1, Classical AMG (C-AMG) solves these problems with relatively large grid sizes in fewer than ten iterations. This phenomena can be attributed to the existence of geometrically smooth error in the near-kernel. For instance, all such directions are smooth for the case of isotropic diffusion, and the y direction contains smooth error for the latter example.



(a) Isotropic Diffusion: Equation (4.1)

(b) Diffusion with Anisotropy in y: Equation (4.2)

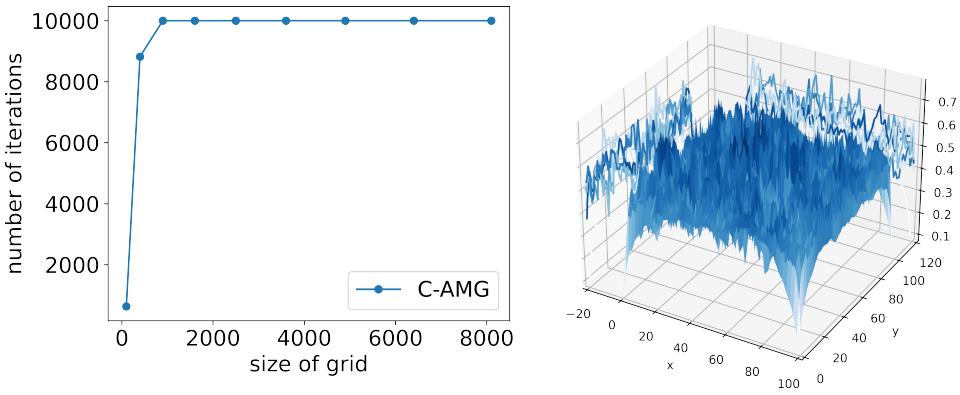
Figure 4.1: C-AMG Performance on Diffusion Problems

However, standard solving techniques break down once we add in the fourth order term because there is no longer a single direction of geometric smoothness. Since C-AMG relies

on the existence of a direction of smooth error as guidance for which direction to coarsen in, these classical techniques lose their functionality. Recall the COGENT problem:

$$-\epsilon u_{xx} - u_{yy} + u_{yyxx} = 0, \tag{4.3}$$

which clearly involves a fourth order term. It has highly oscillatory error in all directions, as shown on the right of Figure 4.2. When projected, the error on the xz and yz planes are visibly not geometrically smooth. Moreover, the plot on the left of Figure 4.2 highlights the fact that C-AMG requires over $10,000$ iterations to converge for even small grid sizes (we capped the number of iterations at $10,000$, so C-AMG in fact requires more than that).



(a) COGENT Problem: Equation (4.3)

(b) COGENT Problem's Oscillatory Error in All Directions

Figure 4.2: C-AMG Performance on a COGENT-Like Problem

Thus, C-AMG is not an adequate tool for this problem. We must develop novel AMG methods to circumvent C-AMG's deficits.

4.2 Solving the Fourth Order Term

Seeing as the fourth order term presents specific challenges, we first isolate this portion of the problem and develop tailored solving techniques for it. These are based on the multigrid reduction approach (see Section 4.2.2), and necessitate strategies for approximate F relaxation, choice of a coarse grid, and approximate ideal interpolation. In the following section, we will generalize these methods into a cohesive algorithm that can handle the mixed problem of diffusion and the fourth order term.

4.2.1 Preliminaries: Total Reduction

Let us consider the fourth order term mentioned above. The corresponding matrix operator has two kinds of near-kernel components (oscillatory in x or oscillatory in y). As a result, there is no direction in which near-kernel error varies only smoothly as required in classical AMG. However, this operator can be handled by semi-coarsening arbitrarily in either x or y and using a multigrid reduction framework. This involves decomposing the matrix A into a C/F-block matrix form [21]:

$$A = \begin{pmatrix} A_{ff} & A_{fc} \\ A_{cf} & A_{cc} \end{pmatrix}.$$

$$\begin{pmatrix} A_{ff} & A_{fc} \\ A_{cf} & A_{cc} \end{pmatrix} \begin{pmatrix} e_f \\ e_c \end{pmatrix} = 0$$

Therefore,

$$e_f = -A_{ff}^{-1} A_{fc} e_c \quad \text{and} \quad e = \begin{pmatrix} -A_{ff}^{-1} A_{fc} \\ I_c \end{pmatrix} e_c$$

Thus, the "ideal" P matrix can be defined as

$$P_* \coloneqq \begin{pmatrix} -A_{ff}^{-1} A_{fc} \\ I \end{pmatrix} \tag{4.4}$$

Define $S = \begin{pmatrix} I \\ 0 \end{pmatrix}$. Then, A and its inverse can be expressed in terms of P_* and S [27]. Namely, we have

$$A = \begin{pmatrix} A_{ff} & A_{fc} \\ A_{cf} & A_{cc} \end{pmatrix} = \begin{pmatrix} I & 0 \\ (A_{ff}^{-1} A_{fc})^T & I \end{pmatrix} \begin{pmatrix} A_{ff} & 0 \\ 0 & S_{cc} \end{pmatrix} \begin{pmatrix} I & (A_{ff}^{-1} A_{fc}) \\ 0 & I \end{pmatrix}.$$

Using P_* from Equation (4.4), we can write

$$\begin{aligned} A^{-1} &= \begin{pmatrix} I & -(A_{ff}^{-1} A_{fc}) \\ 0 & I \end{pmatrix} \begin{pmatrix} A_{ff}^{-1} & 0 \\ 0 & S_{cc}^{-1} \end{pmatrix} \begin{pmatrix} I & 0 \\ (A_{ff}^{-1} A_{fc})^T & I \end{pmatrix} \\ &= S^T \underbrace{(S^T A S)}_{A_{ff}}{}^{-1} S + P_* \underbrace{(P_*^T A P_*)}_{S_{cc}}{}^{-1} P_*^T. \end{aligned}$$

Thus, we can write the exact error propagator as

$$(I - A^{-1}A) = \underbrace{\left(I - S^T (S^T A S)^{-1} S A\right)}_{\text{F-relaxation}} \underbrace{\left(I - P_* (P_*^T A P_*)^{-1} P_*^T A\right)}_{\text{Coarse-grid correction}} = 0. \tag{4.5}$$

4.2.2 Preliminaries: Multigrid Reduction

In this section, we will see how to utilize the total reduction framework in the context of multigrid methods. In a multigrid method, we need some sort of relaxation and then some definition of coarse grids and interpolation operators. First, we will consider the relaxation method.

Suppose we utilized *F-relaxation* as our relaxation method. Namely,

$$\begin{aligned} u^{(k+1)} &= u^{(k)} + \begin{bmatrix} A_{ff}^{-1} & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} r_f \\ r_c \end{bmatrix} \\ &= u^{(k)} + \begin{bmatrix} e_f \\ 0 \end{bmatrix} \\ &= \begin{bmatrix} x_f^* \\ u_c^{(k+1)} \end{bmatrix} \end{aligned}$$

i.e. exactly solving for the F-points and keeping the same C-points. Note that

$$S(S^TAS)^{-1}S^T = \begin{bmatrix} A_{ff}^{-1} & 0 \\ 0 & 0 \end{bmatrix}$$

Thus, our error propagation for F-relaxation is

$$E_F = I - S(S^TAS)^{-1}S^TA$$

We will then use P_* as our interpolation operator and $P_*^TAP_*$ as our coarse grid. Recall from multigrid our coarse error is the solution to the system

$$P_*^TAP_*e_c = P_*^Tr.$$

Therefore,

$$e_c = (P_*^TAP_*)^{-1}Ae_k$$

Then the multigrid update can be written as

$$\begin{aligned} u_{k+1} &= u_k + P_*e_c \\ &= u_k + P_*(P_*^TAP_*)^{-1}Ae_k \end{aligned}$$

Subtracting u from both sides, it is then easy to see that our error term for this *coarse grid correction* step is

$$E_C = I - P_*(P_*^TAP_*)^{-1}P_*^TA$$

Thus, the total error for this two-grid method would be

$$E = E_FE_C = \left(I - A^{-1}A\right) = \left(I - S(S^TAS)^{-1}S^TA\right)\left(I - P_*(P_*^TAP_*)^{-1}P_*^TA\right)$$

It follow from (4.5) that $E = 0$. Thus, this method is exact. Since computing A_{ff}^{-1} is infeasible, it is common to make an approximation. While the method no longer has the theoretical guarantees, it often performs well in practice.

4.2.3 Line Relaxation

Recall from Section 4.2.2 that F-relaxation and ideal coarse grid correction are an exact method. Therefore, a good idea would be to try to approximate these processes. This is known as the MGR framework introduced in [24]. There has since been extensive applications of this framework [14, 10, 27]. However, in each of these methods, they use pointwise relaxation for their F-relaxation. As we saw in Figure 4.2, pointwise relaxation is not sufficient for our higher-order PDE.

In *SMG* [9], they introduce a MGR method with block relaxation. Here, each block corresponds to a geometric line. This choice of blocks causes each of the induced principle submatrices to be tridiagonal, allowing them to be solved exactly in $O(n)$ time. They then partition the lines into two independent sets, taking one as their "Red" lines and one as their "Black" lines. To integrate the reduction framework, they choose their red lines to be the coarse points. Then they solve for the red lines followed by the black lines to approximate F-relaxation. It is important to note that within each set the lines are independent. The independence allows for the lines to be solved in parallel. This is the inspiration for our method. The distinction with our method is that, in [9], they only coarsen in one direction as we will see later.

4.2.4 Semi Coarsening

As alluded to in the previous section. We pick our coarser grid as a subset of the lines used for line relaxation. This results in semi-coarsening in a direction (see Figure 4.3).

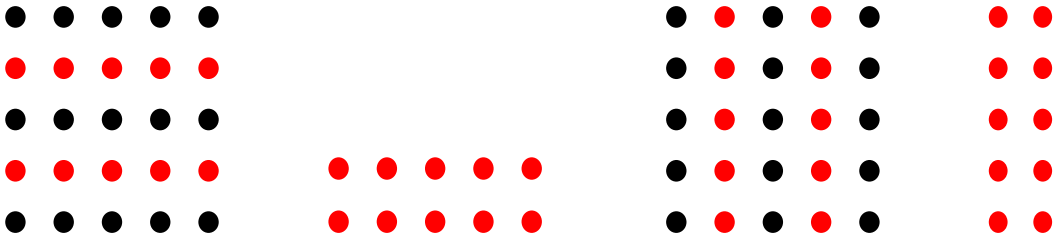


Figure 4.3: Black/Red line partitioning leads to semi-coarsening in y (left) and x (right).

Semi-coarsening has proven useful in the unstructured anisotropic regime [15], where line-relaxation is not feasible but there are oscillatory modes in the near-kernel. This is because the remaining oscillatory error can be exactly solved for efficiently on the coarsest grid, so we do not need to smooth the oscillatory error.

For our problem we have oscillatory near kernel components in all directions. Therefore, an exact solve in one direction is not sufficient. In [9], they do pair semi-coarsening with line relaxation, smoothing the error in one direction while solving for the other direction exactly. As previously mentioned, they only coarsen in the y direction.

4.2.5 Ideal Interpolation

We still need an interpolation operator. For now, suppose that we partition our mesh into "y-lines" i.e. $L_i = \{(i,j) \mid j \leq n\}$. Let J be the Jth line. It is easy to see that

$$A_{J,J-1}u_{J-1} + A_{J,J}u_J + A_{J,J+1}u_{J+1} = f.$$

This gives us

$$A_{J,J-1}e_{J-1} + A_{J,J}e_J + A_{J,J+1}e_{J+1} = 0,$$

such that

$$e_J = -A_{J,J}^{-1}A_{J,J-1}e_{J-1} - A_{J,J}^{-1}A_{J,J+1}e_{J+1}. \tag{4.6}$$

Therefore, if we want to compute the exact interpolation, then we would need to compute $A_{J,J}^{-1}A_{J,J-1}$ and $A_{J,J}^{-1}A_{J,J+1}$. In general, this is still too difficult of a task. However, we can try to exploit our knowledge of the problem to see if this may be feasible. Consider the fourth order stencil

$$A \sim \begin{bmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{bmatrix}.$$

If we look at the principle submatrices of interest, we see that

$$A_{J,J-1} \sim \begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix}, \quad A_{J,J} \sim \begin{bmatrix} -2 \\ 4 \\ -2 \end{bmatrix}, \quad A_{J,J+1} \sim \begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix}.$$

This gives us the convenient relationship

$$A_{J,J-1} = -\frac{1}{2}A_{J,J} = A_{J,J+1}. \tag{4.7}$$

Substituting (4.7) into (4.6), we see that

$$\begin{aligned} e_J &= \frac{1}{2}A_{J,J}A_{J,J}^{-1}e_{J-1} + \frac{1}{2}A_{J,J}A_{J,J}^{-1}e_{J+1} \\ &= \frac{1}{2}e_{J-1} + \frac{1}{2}e_{J+1}. \end{aligned}$$

Thus, we have the following ideal interpolation operator

$$P \sim \begin{bmatrix} 1/2 & 1 & 1/2 \end{bmatrix}_c.$$

An important thing to note is that if we use the classical AMG interpolation from Equation (3.11), then we will exactly recover this ideal interpolation operator. This is left as an exercise for the reader but will prove useful when developing a fully algebraic alternative to our algorithm.

4.2.6 An Exact Method

Now let us see what happens when we put all of these components together. To summarize, we have the following algorithm:

Algorithm 2: Fourth Order Two-Grid

Data: $A \in \mathbb{R}^{n\times n}$, $x_0 \in \mathbb{R}^n$, $b \in \mathbb{R}^n$, $G \in \mathbb{R} \times \mathbb{R}$
Result: $x \in \mathbb{R}^n$
Function `MultiGrid`(A, x, b, G)

```
/* Setup phase */
r ← Ax_0 − b;
P ← AMGinterpolation(A, C, F);
A_c ← P^T AP;
lines ← ConstructLines(A, G) ;
F, C ← PartitionLines(A, lines) ;
/* Solve phase */
r|_c ← (Ax − b)|_c;
for c ∈ C do
    e|_c ← Solve(A_cc, r|_c)
end
x|_c ← x|_c + e|_c;
r|_f ← (Ax − b)|_f;
for f ∈ F do
    e|_f ← Solve(A_ff, r|_f)
end
x|_f ← x|_f + e|_f;
e_c ← MultiGrid(A_c, e_c, P^T r, G′);
x ← x + Pe_c
```

Since the F-lines are independent, our F-line relaxation is actually exact F-relaxation

from MGR. From the previous section, we have that P is our ideal interpolation operator. Therefore, we have an exact two-grid method for the fourth order term.

While this is encouraging, a computationally feasible algorithm requires far more than just two levels. To see how our algorithm will perform when we allow more levels, we can look at how the Galerkin operator with our 2pt-linear interpolation will affect A. It can be shown that

$$P^T AP \sim \frac{1}{2}A. \tag{4.8}$$

From this, we should have an exact method at each coarser level. This can be seen in Figure 4.4, as various grid sizes consistently converge in one iteration.

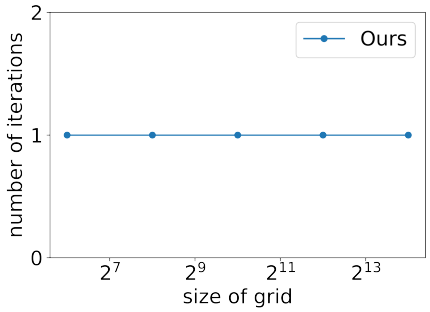


Figure 4.4: Smart Semi-Coarsening with CF-Line relaxation on fourth order problem.

4.3 Dealing with Diffusion

Now that we have an exact method for solving the fourth order term, we must reincorporate the second order terms to the problem. Let us begin in a similar way as we did before: we isolate the diffusion components of the COGENT-like problem in Equation (4.3).

To motivate our approach, consider the diffusion term from Equation (4.2), which has the stencil

$$A \sim \begin{bmatrix} 0 & -1 & 0 \\ -\epsilon & 2+2\epsilon & -\epsilon \\ 0 & -1 & 0 \end{bmatrix}.$$

From Figures 5.1 and 5.2, it is easy to see that the efficacy of solving diffusion problem depends greatly on the choice of direction in which to semi-coarsen. Given that this is anisotropic in the y direction, we choose to coarsen in that direction. Essentially, we are halving the points in the y direction, yielding the stencil

$$\begin{aligned} P^T AP &\sim \begin{bmatrix} -\epsilon/4 & -1/2+\epsilon/2 & -\epsilon/4 \\ -3\epsilon/2 & 1+3\epsilon & -3\epsilon/2 \\ -\epsilon/4 & -1/2+\epsilon/2 & -\epsilon/4 \end{bmatrix} \\ &= \begin{bmatrix} 0 & -\frac{1}{2} & 0 \\ -\epsilon & 1+2\epsilon & -\epsilon \\ 0 & -\frac{1}{2} & 0 \end{bmatrix} + \begin{bmatrix} -\epsilon/4 & \epsilon/2 & -\epsilon/4 \\ -\epsilon/2 & \epsilon & -\epsilon/2 \\ -\epsilon/4 & \epsilon/2 & -\epsilon/4 \end{bmatrix} \end{aligned}$$

To generalize this example, if we are given an expression $-au_{xx} - bu_{yy} = 0$ and semi-coarsen in the y direction, we will halve b. Conversely, if we semi-coarsen in the x direction, we will halve a. We will explore how to utilize this in the next section.

4.3.1 Smart Coarsening

From the previous section or Figures 5.1 and 5.2, we know that the direction of anisotropy should determine which direction to semi-coarsen. However, semi-coarsening on a sufficiently large grid can cause anisotropy in the wrong direction. To see this, suppose that we have the following anisotropic diffusion problem:

$$-u_{xx} - \epsilon u_{yy} = 0, \qquad \text{for } \epsilon = 1 \times 10^{-5}.$$

Then, if our grid is sufficiently large, we could feasibly semi-coarsen in x for > 14 levels. For every additional level, we would effectively be semi-coarsening in the wrong direction of strong anisotropy equivalent to our inital problem.

To prevent this semi-coarsening in the wrong direction of anisotropy, we alternate semi-coarsening between the x and y directions. This results in an isotropic diffusion term which is very well understood. We make this explicit in the algorithm: *Smart* semi-coarsening.

Algorithm 3: Smart Semi-Coarsening

```
input: Coefficients a and b
while We have not yet reached the coarsest grid do
    if a ≥ b then
        Semi-coarsen in the x direction
        a = a/2
    else
        Semi-coarsen in the y direction
        b = b/2
    end
end
```

This idea is also implemented in *PFMG* [1]. However, they only utilize pointwise relaxation. Note that this prevents the "over-correction" explained previously. To form our complete algorithm, this strategy precedes the remaining ideas discussed above.

4.4 Boundary Conditions

Before considering our complete algorithm, we must take varying boundary conditions into account. Until now, we have been operating under the assumption that the COGENT problem has the following Dirichlet boundary conditions:

$$\begin{aligned} -au_{xx} - bu_{yy} + acu_{yyxx} &= g, \quad b, c \gg a \\ u &= 0 \quad \text{on } \partial\Omega. \end{aligned}$$

In other words, the boundary values are all set to 0.

However, this is not the case in the COGENT code. Instead of Dirichlet boundary conditions, this problem actually requires boundary conditions that are periodic in the y direction. This means that this time we are not given the values of $u(x_{i,0})$ and $u(x_{i,m})$, but instead we have additional equations relating these values. In particular, for $\frac{\partial^2}{\partial y^2}$, we get the equations

$$\begin{aligned} -u_{i,m} + 2u_{i,0} - u_{i,1} &= -h^2 f(x_{i,0}), \\ -u_{i,m-1} + 2u_{i,m} - u_{i,0} &= -h^2 f(x_{i,m}), \end{aligned}$$

for all $0 \leq i \leq n$. The discretization of the full COGENT Equation (1.1) with these new boundary conditions then includes linear equations relating the first and last row of our grid. An illustration of these relations is shown in Figure 4.5.

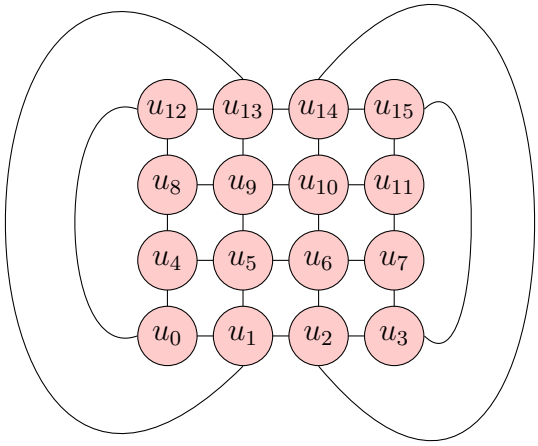


Figure 4.5: Grid with Periodic Boundary Conditions

Adjusting the boundary conditions brings about a new challenge: the A matrix is singular, which presents difficulties in the coarsest grid of our algorithm because the kernel will no longer be empty. To fix this issue, we simply stop one level early.

Chapter 5

Results

In this chapter, we will show the results we obtained by implementing our algorithms in Python and PyAMG, see [3].

5.1 Semi Coarsening

Before implementing boundary conditions that are periodic in y (described above in Chapter 4), we investigate SMG methods for a simpler scenario. Thus, we use Dirichlet conditions, where the boundary is set to zero. Operating under this framework, we compare three semi coarsening methods against our own algorithm.

Abbreviated as *x-SMG*, we investigate illustrative examples where we only coarsen in the x direction. Likewise, *y-SMG* represents coarsening only in the y direction. To synthesize the two, *alt-SMG* denotes a coarsening method where the direction of semi coarsening alternates between x and y with each iteration. For each of these, we do exact C/F-line relaxation in the direction corresponding with the direction of coarsening. Lastly, our algorithm – *CF-SSMG* – uses the same C/F-line relaxation with smart semi coarsening (see Section 4.2.3).

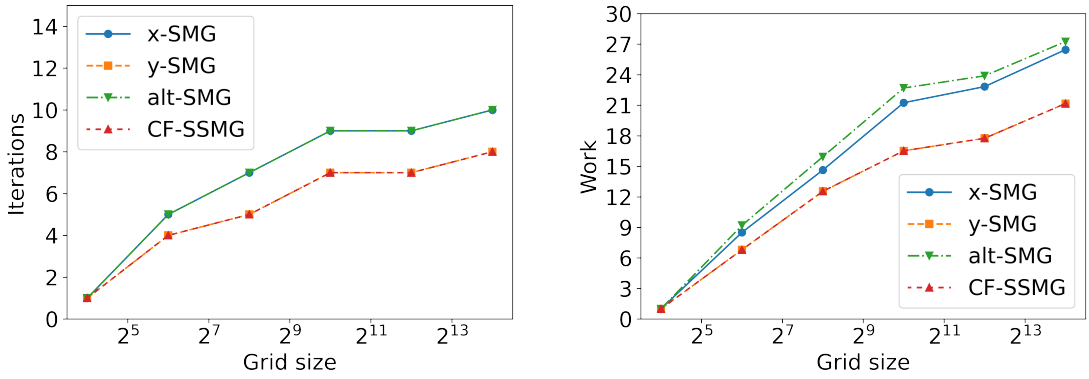
In the following three sections, we demonstrate that our algorithm (CF-SSMG) does at least as well as these other semi coarsening methods when we measure the number of iterations and work required for an example problem to converge. Here, we define work as the product of operator complexity and iteration count, and we consider the problem to be converged when the norm of the relative residual has reached below the threshold of $\frac{\|r\|}{\|r_0\|} < 10^{-5}$. Throughout these cases, we use exact F relaxation and test five square grids of size $(2^n - 1) \times (2^n - 1)$, where $n = \{3, 4, ..., 7\}$.

5.1.1 Diffusion with Anisotropy in y

As a first example, consider the problem

$$-\epsilon u_{xx} - u_{yy} = 0,$$

with $\epsilon = 0.005$, which is highly anisotropic. The error is smooth in the y direction but highly oscillatory in the x direction. Therefore, the swiftest algorithm should semi coarsen in the y direction, which is reflected in the results shown below.



(a) Iteration Count against Grid Size (b) Work against Grid Size

Figure 5.1: Results for Diffusion with Anisotropy in the y Direction

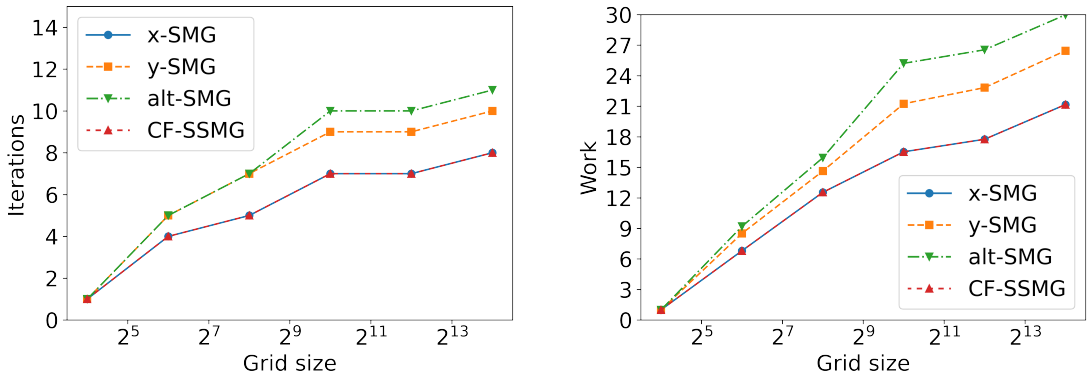
As expected, Figure 5.1 confirms that coarsening in the y direction yields the best solution. Conversely, semi coarsening in the x direction performs the worst for work and iteration calculations. Alternating, intuitively, produces results that are between the two. Most importantly, the CF-SSMG algorithm performs as well as coarsening in the y direction, as needed.

5.1.2 Diffusion with Anisotropy in x

Next, we consider a similar example given by

$$-u_{xx} - \epsilon u_{yy} = 0,$$

which is also anisotropic. However, unlike the scenario examined above, the smooth error is found in the x direction, while the oscillatory error appears in the y direction. Thus, we expect that the best solver would coarsen in the x direction.



(a) Iteration Count against Grid Size (b) Work against Grid Size

Figure 5.2: Results for Diffusion with Anisotropy in the x Direction

The results shown in Figure 5.2 depict opposite trends as does Figure 5.1. In other words, coarsening in the x direction performs the best, while coarsening in the y direction gives

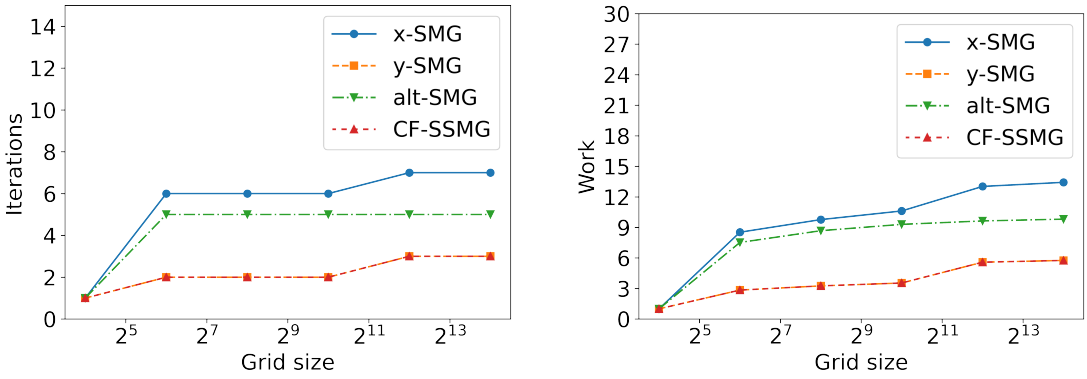
the slowest convergence. However, the graphs are similar in that the CF-SSMG algorithm does as well as the best of the three other semi coarsening methods.

5.1.3 COGENT-Like Problem

Finally, we combine the two problems above into a problem similar to the one given by the COGENT problem. This is

$$-\epsilon u_{xx} - 2u_{yy} + 2u_{yyxx} = 0.$$

Here, we will no longer maintain a single direction of oscillatory error throughout the entire problem, so it is no longer obvious in which direction to coarsen for the entire solving process. Without the fourth order term, the problem emulates the first one given, where there was diffusion with anisotropy in the y direction. Intuitively, then, we would expect that coarsening in the y direction would perform the best.



(a) Iteration Count against Grid Size

(b) Work against Grid Size

Figure 5.3: Results for COGENT-Like Problem

This prediction is confirmed in the figure above, and the CF-SSMG algorithm matches the best performance. Thus, we find that in all of our scenarios, the CF-SSMG coarsening technique is the only option that always gives the most optimal solution. Hence, our algorithm is preferable. As we exhibit in the following section, the same is true when swapping Dirichlet boundary conditions for periodic boundary conditions.

5.2 Comparison of Algorithms

In this section, we will compare three newly developed algorithms and classical AMG. All three new algorithms use Smart Semicoarsening Multigrid, as described in section 5.1. Therefore, we call these algorithms X-SSMG, where X defines the presmoother, postsmoother and F-relaxation, as follows:

- CF-SSMG: The first algorithm uses exact CF-line relaxation from Section 4.2.3 as a presmoother and doesn't have a postsmoother nor F-relaxation.
- VL-SSMG: The next one has CF-line relaxation as a presmoother and FC-line relaxation as a postsmoother, where in both cases we run a V-cycle. Again, no F-relaxation is used.

- P-SSMG: In this algorithm, P stands for pointwise, since Gauss-Seidel is used as both pre- and postsmoother. This time we use V-cycle as F-relaxation.

We compare all of these algorithms against the classical AMG (C-AMG), which uses classical coarsening, Gauss-Seidel as both a pre- and postsmoother, and has no F-relaxation. The definition of these four algorithms can also be seen in Table 5.1. In all of them, we use LU factorization on the coarsest level.

	CF-SSMG	VL-SSMG	P-SSMG	C-AMG
Coarsening	Smart	Smart	Smart	Classical
Presmoother	E-CF-line	V-CF-line	GS	GS
Postsmoother	None	V-FC-line	GS	GS
F-relaxation	None	None	V-Cycle	None

Table 5.1: Comparison of Algorithms

We will compare the number of iterations for the different algorithms, as well as work as defined in Section 5.1. We illustrate our results on two problems – one where the fourth-order term dominates and another with the COGENT coefficients. The boundary conditions are Dirichlet in x-direction and periodic in y-direction. In each case, we set the right-hand size to 0, start with a random initial guess, and stop when the norm of the residual is less than 10^{-5}.

5.2.1 Fourth-Order Dominant Problem

Here, we consider the equation

$$-\epsilon u_{xx} - \epsilon u_{yy} + u_{yyxx} = 0 \tag{5.1}$$

with $\epsilon = 10^{-5}$, so the fourth-order term is dominant. In Table 5.2, we show the convergence factor for the fourth order term. We computed the convergence factor as

$$\frac{\left\|r^{(20)}\right\|}{\left\|r^{(19)}\right\|},$$

for a random initialization on a 127×127 grid.

Algorithm	Convergence factor
CF-SSMG	2.03e−08
VL-SSMG	9.12e−04
P-SSMG	2.06e−03
C-AMG	9.92e−01

Table 5.2: Comparison of Results for Fourth-Order Dominant Problem: Equation (5.1)

We can see that all three algorithms using SSMG perform a lot better than C-AMG. In particular, we see that C-AMG performs incredibly poorly with a convergence factor of nearly 1. This means that after each iteration, the normed residual is barely decreasing. Alternatively, the SSMG-based algorithms all drastically outperform C-AMG with CF-SSMG having a convergence factor on the order of 10^{-8}.

Next, we run the four algorithms on grids of sizes $(2^k-1)\times(2^k-1)$ for all $k \in \{2,3,\ldots,8\}$. The number of iterations and work are shown in Figure 5.4. However, C-AMG needs more than 1000 iterations, so its graph is not visible in these plots.

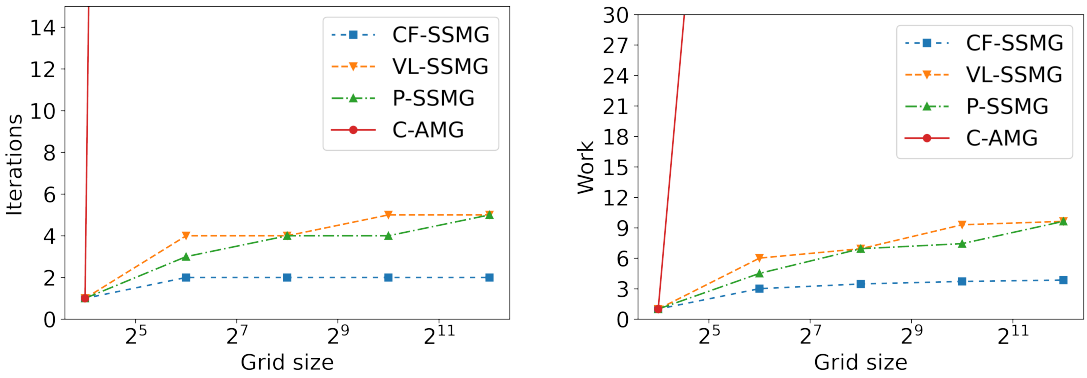


Figure 5.4: Results for Fourth Order Dominant Problem (Equation (5.1))

Again, all three new algorithms perform much better than C-AMG, always requiring at most 9 iterations. Like before, CF-SSMG got the lowest number of iterations and work, with the maximum iterations being 3.

5.2.2 COGENT Problem

Now consider

$$-0.0003u_{xx} - 2u_{yy} + 2u_{yyxx} = 0. \tag{5.2}$$

In Table 5.3, we once again consider the convergence factors of each of the algorithm.

Algorithm	Convergence factor
CF-SSMG	2.86e−08
VL-SSMG	5.14e−04
P-SSMG	1.81e−03
C-AMG	9.91e−01

Table 5.3: Results for COGENT Problem: Equation (5.2)

These results for the COGENT problem are very similar to those for the fourth order dominant problem in Table 5.2, with CF-SSMG having the best convergence rate and C-AMG once again having a convergence rate near 1.

Again, we looked at how the number of iterations and amount of work change as the size of the grid increases. Grids of the same sizes as for Figure 5.4 were used in Figure 5.5.

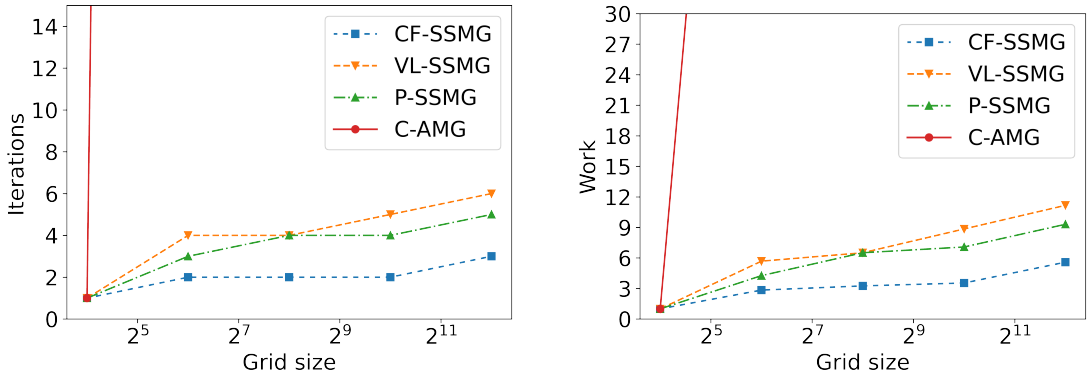


Figure 5.5: Results for COGENT Problem (Equation (5.2))

Each of the three SSMG algorithms require at most 9 iterations before the norm of the relative residual is less than 10^{-5}. For C-AMG, the number of iterations is again more than 1000, so it is not visible in the plots. The best algorithm in terms of both number of iterations and in terms of work is CF-SSMG, with VL-SSMG and P-SSMG only having marginally more iterations.

Chapter 6

Conclusion

6.1 Work Done

We have conducted an extensive literature review in order to orient ourselves with the mathematics necessary to embark on this project. Once we felt equipped to begin researching new methods, we pieced together an algorithm to solve the fourth order term, then developed the prerequisite diffusion method (described in Chapter 4). Furthermore, the results in Chapter 5 confirm that our algorithm is preferable for the COGENT problem. Along the way, we made advancements and implemented many approaches that did not prove to be optimal for the COGENT problem, but may become practical for other sorts of problems.

6.2 Value of Work

The algorithms and approaches suggested hope to make current Algebraic Multigrid Methods more robust in terms of being able to solve higher order PDEs with high efficiency. They cater to a vast set of complex physics problems, but more specifically to kinetic edge models pertaining to plasma confinement inside a tokamak. We believe that if these approaches are further researched, one can develop a fully algebraic solver that does not depend on the problem geometry and would be able to solve the system based solely on its matrix entries. The main algorithm suggested in Chapter 4 provides positive results for the cases tested by C-AMG on the simplified COGENT problem. If it is expanded for higher dimensions, higher order discretization, and parallel computing, it could be tested as a preconditioner for the COGENT code used to solve the kinetic equation resulting from an edge plasma simulation.

6.3 Future Directions

In the beginning of this report, we briefly discussed fully algebraic methods to solve linear systems [7, 22, 26]. Additionally, there have been previous attempts to algebraically identify lines in cases when there is no geometrical associated grid [19, 16]. However, these methods require other information specified, such as desired direction and location of "edges." There has been success in the anisotropic regime [20, 23] with heuristics based on strong couplings. Although, in the fourth order term, we do not have this anisotropy.

Due to this, our proposed algorithm relies heavily on knowledge of the underlying grid. This is highly undesirable in pursuit of a general solver, so we have begun developing some preliminary ideas to circumnavigate this issue. The first is the definition of a "algebraic" line. We take inspiration from graph based methods for defining permutation matrices [11].

Definition 4. *A set of points L forms a **line** if the principal sub-matrix induced by L, $A_{L,L}$, is tri-diagonal.*

While this allows us to identify lines algebraically, there is still the issue of identifying the "best" lines. In the case of a structured grid, we hope to recover the geometric lines considered in our algorithm. To do this, we adopt the following heuristic:

1. Pick the point with the smallest number of connections (corner point).
2. Pick the point with the most connections connected to the corner point.
    - Decrement all connected non-chosen points.
3. If possible, pick the point with the same number of connections as chosen point (points in same line should have same number of connections).
    - Otherwise pick point with minimal number of connections (end corner point).
4. Increment all connected non-chosen points.
5. Continue until all points are allocated.

From this, we can recover the geometric lines, even in the case of the pure fourth order term. However, this heuristic breaks down in the presence of periodic boundary conditions. As a future direction, we will attempt to adapt this heuristic to utilize the matrix entries as a proxy for varying levels of strength.

To continue developing our algorithm, assume we have our desired lines. We must still assign them to our C/F partitioning as in Chapter 4. To do this, we form an adjacency matrix where each entry corresponds to a line. Two lines are then connected if points in the lines are connected. To form a C/F partitioning we take our F points to be a maximally independent set of this matrix. After we have our partitioning, we can use classical AMG interpolation to interpolate from our coarse lines to our finer lines. In preliminary experiments, this fails to sufficiently coarsen our grid. Therefore, more strategic interpolation should be considered, such as the line interpolation introduced in [9].

Appendix A

Math Background

A.1 Proof of Corollary 2

Corollary 2. *If A is a positive (semi-)definite matrix, then*

$$v^T Av = 0 \implies Av = 0.$$

Furthermore, if A is positive definite, then

$$\|v\|_A = 0 \implies v = 0.$$

Proof. Since A is a positive (semi-) definite matrix, we can find an eigenbasis of $\mathbb{R}^n$ with respect to A: $(v_i)_{i=1}^n$. We then have that

$$v^T Av = \sum_{i=1}^{n} c_i^2 \lambda_i$$

for some $c_1, \ldots, c_n \in \mathbb{R}$. Once again using that A is a PSD matrix, we know that each $\lambda_i \geq 0$. This gives us the following

$$c_i = \begin{cases} \geq 0, & \text{if } \lambda_i = 0 \\ 0, & \text{if } \lambda_i > 0 \end{cases}$$

Now if A is positive definite every $\lambda_i > 0$. Thus, each $c_i = 0$ and $v = 0$ as desired. Otherwise,

$$v = \sum_{i \in I} c_i v_i,$$

where $I = \{i \in 1, \ldots, n \mid \lambda_i = 0\}$. Thus,

$$Av = \sum_{i \in I} \lambda_i c_i v_i = 0$$

as desired. □

A.2 Proof of Proposition 1

Proposition 1. *Let $e_t = u - u_t$, which implies $Ae_t = r_t$ as above. The solution e_c to the following system*

$$P^T A P e_c = P^T r_t$$

is also the solution to Equation (3.6).

Proof. To find the minimizer of Equation 3.6, we can solve

$$\nabla_{e_c} \|e_t - P e_c\|_A = 0$$

From matrix calculus, this yields

$$\begin{aligned} \nabla_{e_c} \|e_t - P e_c\|_A &= \nabla_{e_c} \left(e_t A e_t + (P e_c)^T A (P e_c) - e_t^T A P e_c - (P e_c)^T A e_t \right) \\ &= 2 P^T A P e_c - 2 P^T A e_t \\ &= 2 P^T A P e_c - 2 P^T r_t \qquad (\star) \end{aligned}$$

Setting $\star$ equal to 0, yields

$$P^T A P e_c = P^T r_t$$

as desired. □

A.3 Proof of Proposition 2

Proposition 2. *Let $A \in \mathbb{R}^{n \times m}$ and $v \in \mathbb{R}^m$. For every $\epsilon > 0$, there exists some $\mathcal{E}(\epsilon)$ such that if*

$$\|v\|_2 \leq \mathcal{E}(\epsilon)$$

then we have that $\|v\|_A \leq \epsilon$.

Proof. Let $\delta = \frac{\sqrt{\epsilon}}{\sqrt{\|A\|_2}}$ and $v \in \mathbb{R}^m$ such that $\|v\|_2 < \delta$. Then we have by Cauchy-Schwartz

$$\|v\|_A = v^T A v \leq \|v\|_2 \|Av\|_2$$

Using Lemma 3.1, we have that

$$\|v\|_A \leq \|v\|_2 \|Av\|_2 \leq \|A\|_2 \|v\|_2^2 = \epsilon$$

as desired. □

A.4 Equation A.4 Derivation

Equation 2.14

$$\|e_t^{N_\delta}\|_A = \left(e_t^{N_\delta}\right)^T A e_t^{N_\delta} = \sum_{i<j} -a_{ij}(e_i - e_j)^2 \ll 1.$$

Proof. The following derivation is purely algebraic.

$$\mathbf{e}^T A\mathbf{e} = \sum_i e_i \left(a_{ii} e_i + \sum_{j \neq i} a_{ij} e_j \right)$$

Making use of the fact that $a_{ii} = -\sum_{j \neq i} a_{ij}$,

$$\begin{aligned}
\mathbf{e}^T A\mathbf{e} &= \sum_i e_i \left(\sum_{j \neq i} (-a_{ij}) (e_i - e_j) \right) \\
&= \sum_{i<j} (-a_{ij}) e_i (e_i - e_j) + \sum_{i>j} (-a_{ij}) e_i (e_i - e_j) \\
&= \sum_{i<j} (-a_{ij}) e_i (e_i - e_j) - \sum_{i<j} (-a_{ji}) e_j (e_i - e_j) \\
&= \sum_{i<j} (-a_{ij}) (e_i - e_j)^2 \ll 1
\end{aligned}$$

□

Selected Bibliography Including Cited Works

[1] S. F. Ashby and R. D. Falgout, *A parallel multigrid preconditioned conjugate gradient algorithm for groundwater flow simulations*, Nuclear science and engineering, 124 (1996), pp. 145–159.

[2] A. H. Baker, R. D. Falgout, T. V. Kolev, and U. M. Yang, *Scaling hypre's multigrid solvers to 100,000 cores*, in High-performance scientific computing: algorithms and applications, Springer, 2012, pp. 261–279.

[3] N. Bell, L. N. Olson, and J. Schroder, *PyAMG: Algebraic multigrid solvers in python*, Journal of Open Source Software, 7 (2022), p. 4142.

[4] A. Brandt, *Algebraic multigrid theory: The symmetric case*, Applied mathematics and computation, 19 (1986), pp. 23–56.

[5] A. Brandt and O. E. Livne, *Interior Relaxation and Smoothing Factors*, Society for Industrial and Applied Mathematics, 2011, pp. 31–45.

[6] A. Brandt, S. McCormick, and J. Ruge, *Algebraic multigrid (amg) for sparse matrix equations*, Sparsity and its Applications, 257 (1984).

[7] J. Brannick, M. Brezina, S. MacLachlan, T. Manteuffel, S. McCormick, and J. Ruge, *An energy-based amg coarsening strategy*, Numerical linear algebra with applications, 13 (2006), pp. 133–148.

[8] J. J. Brannick and R. D. Falgout, *Compatible relaxation and coarsening in algebraic multigrid*, Society for Industrial and Applied Mathematics Journal on Scientific Computing, 32 (2010), pp. 1393–1416.

[9] P. N. Brown, R. D. Falgout, and J. E. Jones, *Semicoarsening multigrid on distributed memory machines*, Society for Industrial and Applied Mathematics Journal on Scientific Computing, 21 (2000), pp. 1823–1834.

[10] Q. M. Bui, D. Osei-Kuffuor, N. Castelletto, and J. A. White, *A scalable multigrid reduction framework for multiphase poromechanics of heterogeneous media*, Society for Industrial and Applied Mathematics Journal on Scientific Computing, 42 (2020), pp. B379–B396.

[11] E. Cuthill and J. McKee, *Reducing the bandwidth of sparse symmetric matrices*, in Proceedings of the 1969 24th national conference, 1969, pp. 157–172.

[12] M. Dorf and M. Dorr, *Continuum gyrokinetic simulations of edge plasmas in single-null geometries*, Physics of Plasmas, 28 (2021).

[13] R. D. Falgout, *An introduction to algebraic multigrid*, Computing in Science & Engineering, 8 (2006), p. 24–33.

[14] R. D. Falgout, S. Friedhoff, T. V. Kolev, S. P. MacLachlan, and J. B. Schroder, *Parallel time integration with multigrid*, Society for Industrial and Applied Mathematics Journal on Scientific Computing, 36 (2014), pp. C635–C661.

[15] J. Francescatto and A. Dervieux, *A semi-coarsening strategy for unstructured multigrid based on agglomeration*, International journal for numerical methods in fluids, 26 (1998), pp. 927–957.

[16] O. Hassan, K. Morgan, and J. Peraire, *An implicit finite-element method for high-speed flows*, International journal for numerical methods in engineering, 32 (1991), pp. 183–205.

[17] J. J. Hu, C. M. Siefert, and R. S. Tuminaro, *Smoothed aggregation for difficult stretched mesh and coefficient variation problems*, Numerical Linear Algebra with Applications, 29 (2022), p. e2442.

[18] ITER, *What is a tokamak?* https://www.iter.org/mach/Tokamak, 2023.

[19] D. Martin and R. Loehner, *An implicit, linelet-based solver for incompressible flows*, in 30th Aerospace Sciences Meeting and Exhibit, 1992, p. 668.

[20] D. J. Mavriplis, *Multigrid strategies for viscous flow solvers on anisotropic unstructured meshes*, Journal of Computational Physics, 145 (1998), pp. 141–165.

[21] W. Mitchell, R. Falgout, and D. Osei-Kuffuor, *Parallel algebraic multigrid for fusion and higher-order pdes*, Lawrence Livermore National Laboratory Internal Report, (2023).

[22] L. N. Olson, J. Schroder, and R. S. Tuminaro, *A new perspective on strength measures in algebraic multigrid*, Numerical Linear Algebra with Applications, 17 (2010), pp. 713–733.

[23] B. Philip and T. P. Chartier, *Adaptive algebraic smoothers*, Journal of Computational and Applied Mathematics, 236 (2012), pp. 2277–2297.

[24] M. Ries, U. Trottenberg, and G. Winter, *A note on mgr methods*, Linear Algebra and its Applications, 49 (1983), pp. 1–26.

[25] J. W. Ruge and K. Stüben, *Algebraic multigrid*, (1987), pp. 73–130.

[26] P. Vanek, J. Mandel, and M. Brezina, *Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems*, Computing, 56 (1996), pp. 179–196.

[27] L. Wang, D. Osei-Kuffuor, R. Falgout, I. Mishev, and J. Li, *Multigrid reduction for coupled flow problems with application to reservoir simulation*, in SPE Reservoir Simulation Conference, OnePetro, 2017.

[28] I. Yavneh, *Why multigrid methods are so efficient*, Computing in Science & Engineering, 8 (2006), pp. 12–22.