Efficient Iterative and Multigrid Solvers with Applications

A Dissertation Presented

by

Cao Lu

to

The Graduate School

in Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

in

Applied Mathematics and Statistics

Stony Brook University

August 2016

Stony Brook University

The Graduate School

Cao Lu

We, the dissertation committe for the above candidate for the

Doctor of Philosophy degree, hereby recommend

acceptance of this dissertation

Xiangmin Jiao
Associate Professor, Department of Applied Mathematics and Statistics

Roman Samulyak
Professor, Department of Applied Mathematics and Statistics

Matthew Reuter
Assistant Professor, Department of Applied Mathematics and Statistics

Marat Khairoutdinov
Associate Professor, School of Marine and Atmospheric Sciences

Minghua Zhang
Professor, School of Marine and Atmospheric Sciences

This dissertation is accepted by the Graduate School

Nancy Goroff
Interim Dean of the Graduate School

Abstract of the Dissertation

Efficient Iterative and Multigrid Solvers with Applications

by

Cao Lu

Doctor of Philosophy

in

Applied Mathematics and Statistics

Stony Brook University

2016

Iterative methods are some of the most important techniques in solving large scale linear systems. Compared to direct methods, iterative solvers have the advantage of lower storage cost and better scalability as the problem size increases. Among the methods, multigrid solvers and multigrid preconditioners are often the optimal choices for solving systems that arise from partial differential equations (PDEs). In this dissertation, we introduce several efficient algorithms for various scientific applications. Our first algorithm is a specialized geometric multigrid solver for ill-conditioned systems from Helmholtz equations. Such equations appear in climate models with pure Neumann boundary condition and small wave numbers. In numerical linear algebra, ill-conditioned and even singular systems are inherently hard to solve. Many standard methods are either slow or non-convergent. We demonstrate that our solver delivers accurate solutions with fast convergence. The second algorithm, HyGA, is a general hybrid geometric+algebraic multigrid framework for elliptic type PDEs. It leverages the rigor, accuracy and efficiency of geometric multigrid (GMG) for hierarchical unstructured meshes, with the flexibility of algebraic multigrid (AMG) at the coarsest level. We conduct numerical experiments using Poisson equations in both 2-D and 3-D, and demonstrate the advantage of HyGA over classical GMG and AMG.

Besides the aforementioned algorithms, we introduce an orthogonally projected implicit null-space method (OPINS) for saddle point systems. The traditional null-space method is inefficient because it is expensive to find the null-space explicitly. Some alternatives, notably constraint-preconditioned or projected Krylov methods, are relatively efficient, but they can suffer from numerical instability. OPINS is equivalent to the null-space method with an orthogonal projector, without forming the orthogonal basis of the null space explicitly. Our results show that it is more stable than projected Krylov methods while achieving similar efficiency.

Contents

List of Figures

List of Tables

Acknowledgements

First, I would like to thank Prof. Xiangmin Jiao for making solving linear systems such an interesting topic. It has been a great pleasure and valuable experience working with him. I have learned a lot from his unconventional way of thinking and vast knowledge in software development. I express my gratitude for all his guidance.

Next, I would like to thank Prof. Roman Samulyak and Prof. Marat Khairoutdinov of Stony Brook University, Dr. Vijay Mahadevan of Argonne National Laboratory and Dr. Daniel Einstein of Pacific Northwest Laboratory for the opportunities to work on interesting projects. The collaborations greatly broaden and further motivate my research.

I would also like to thank all the present and past members of my research group, including Tristan Delaney, Wei Li, Aditi Ghai and Dr. Navamita Ray for some of the experimental results, and Hongxu Liu, Xinglin Zhao, Rebecca Conley, Oliver Young and Xuebin Wang for the helpful discussions. Thank you all for the encouragement and friendship.

Finally, I would like to thank my parents for their endless love and support. I dedicate this dissertation to you.

Chapter 1

Introduction

Since the early nineties, there has been a strongly increasing demand for efficient methods to solve large and sparse linear systems

$$\mathbf{Ax} = \mathbf{b}. \tag{1.0.1}$$

Direct methods and stationary iterative methods either have large memory requirement or very slow convergence. Krylov subspace methods while effective with a good preconditioner, do not perform very well by themselves.

Nowadays multigrid methods are considered the most advanced, flexible and computationally optimal tool for numerical solutions. With a carefully designed scheme, the multigrid method either as a standalone solver or a preconditioner can achieve linear complexity by utilizing the complementarity between smoothing and coarse-grid correction. Currently there are two major classes of the multigrid method: *geometric multigrid* (GMG) and *algebraic multigrid* (AMG). Compared to GMG, AMG does not require explicit knowledge of geometry and can be applied to more general problems involving unstructured meshes and discontinuity [15]. On the other hand, GMG, when applicable, enjoys fast convergence and low storage cost. Programers and users of multigrid methods often must face difficult choices among these variants. To make matters worse, GMG algorithms are often highly problem-dependent. Different coefficients or even different discretization methods can result in significant changes of the algorithm.

In this dissertation, we seek to generalize the GMG algorithms to solve more complicated problems. We first introduce a specialized solver for anisotropic Helmholtz equations on nonuniform structured grids. It utilizes a line-smoother in the direction of strong connection to smooth out errors geometrically. With a smooth error, we then apply bi-linear interpolation with ghost cells along the boundary to transfer the errors accurately. For unstructured meshes, we propose a hybrid *geometric+algebraic (HyGA) multigrid* framework. At a high-level, our overall approach may be summarized as follows. Our hierarchical mesh generator starts from a good-quality coarse unstructured mesh that is a sufficiently accurate representation of the geometry. It iteratively refines the mesh with guaranteed mesh quality (by uniform refinements) and geometric accuracy (by high-order boundary reconstruction [45]). We apply GMG with a multilevel weighted-residual formulation on these hierarchical meshes, and employ the AMG at the coarsest level. We use a semi-iterative method, namely the Chebyshev-Jacobi method, as the smoother at both the GMG and AMG levels, and utilize Krylov-subspace methods as coarse-grid solvers. In addition, we introduce a unified derivation of restriction and prolongation operators for multilevel weighted-residual methods for linear PDEs with hierarchical basis functions. It allows a more systematic derivation of geometric multigrid methods for finite element methods as well as weighted least squares based methods over unstructured meshes.

Our third main contribution is the orthogonally *projected implicit null-space method (OPINS)* for saddle point systems. The standard null-space method [10] can be effective, but computing an orthonormal basis is very expensive. More recent methods such as the constraint preconditioning [61, 36] and the projected Krylov methods [37] are equivalent to the null-space method with an orthonormal basis, but they do not require computing the basis explicitly. However these methods suffer from rounding errors and numerical instability. OPINS is a more stable variant of the implicit null-space methods. It uses the range-space of the constraint matrix to compute an orthogonal projector onto the null-space. This approach is much cheaper than computing the null-space directly especially if the number of constraints is small. At the same time, the explicit projection makes it more stable than the projected Krylov methods. Another advantage of OPINS is that it can be applied to singular and compatible systems. In particular, it finds the minimum-norm solution in terms of the solution variables.

The dissertation is organized as follows. We first give an overview of iterative solvers with an emphasis on the Krylov subspace methods in Chapter 2. Next, we outline the standard multigrid methods in Chapter 3 and introduce the specialized geometric multigrid solver for anisotropic Helmholtz equations in Chapter 4. We propose a general hybrid geometric+algebraic multigrid framework (HyGA) for elliptic PDEs in Chapter 5. Chapter 6 introduces the OPINS method for saddle point systems. Chapter 7 applies the algorithms to various applications such as climate modeling and elasticity with brittle fracture. Extension of HyGA to WLS-based discretization methods are also explorered. Chapter 8 concludes the paper with discussions of future work.

Chapter 2

Overview of Iterative Methods

The methods discussed in this chapter serve as the foundation for the development of new algorithms. We first introduce the stationary iterative methods as they are the simplest. These methods are used as smoothers in the more powerful multigrid methods whose details will be discussed in Chapter 3. Next we introduce some of the most widely used methods, namely the Krylov subspace methods for symmetric and nonsymmetric systems. These methods are more effective than the simple stationary iterative methods. Additionally their performance can be greatly improved by preconditioning. In fact, the multigrid preconditioner is one of the best preconditioners used in practice. Section 2.3 discusses some popular preconditioners and demonstrates their effectiveness in accelerating the convergence of Krylov subspace methods. Finally, we explore some methods for solving singular systems in Section 2.4.

2.1 Stationary Iterative Methods

Stationary iterative methods can be interpred as a fixed point iteration obtained by matrix splitting. Let $\mathbf{A} = \mathbf{M} - \mathbf{N}$ and the system can then be written as

$$\mathbf{Mx} = \mathbf{N}x + \mathbf{b}. \tag{2.1.1}$$

Assuming $\mathbf{M}$ is invertibe, we have

$$\mathbf{x}_{k+1} = \mathbf{M}^{-1}\mathbf{N}\mathbf{x}_k + \mathbf{b}. \tag{2.1.2}$$

This is the basic form of the stationary iterative method. Alternatively, it can be interpreted as a preconditioned Richardson iteration. Let $\mathbf{r}_k = \mathbf{b} - \mathbf{A}\mathbf{x}_k$, then the Richardson iteration has the following form

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{r}_k. \tag{2.1.3}$$

Left multiplying $\mathbf{M}^{-1}$on both sides of the linear system (1.0.1), the Richardson iteration for $\mathbf{M}^{-1}\mathbf{A}\mathbf{x} = \mathbf{M}^{-1}\mathbf{b}$ becomes

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{M}^{-1}\mathbf{r}_k. \tag{2.1.4}$$

It is easy to show that equations (2.1.4) and (2.1.2) are equivalent.
Different choices of splitting will lead to various schemes:

- If $\mathbf{M} = \mathbf{D}$, where $\mathbf{D}$ is the diagonal part of matrix $\mathbf{A}$, we have the Jacobi iteration. It is guaranteed to converge if $\mathbf{A}$ is diagonally dominant;
- If $\mathbf{M} = \mathbf{D} + \mathbf{L}$ where $\mathbf{L}$ is the lower triangular part of matrix $\mathbf{A}$ we have the Gauss-Seidel iteration. It is guaranteed to converge if $\mathbf{A}$ is diagonally dominant or symmetric positive definite;
- If $\mathbf{M} = \mathbf{D} + \omega\mathbf{L}$ where $0 < \omega < 2$, we have SOR iteration. Its applicability depends on ω.

These iteration schemes work for a wide range of problems, and they can often be implemented in a matrix-free fashion. However, this comes at the price of slow convergence and poor scalability. In particular, the ratio of the error reduction is measured by $\rho\left(\mathbf{M}^{-1}\mathbf{N}\right)$, which is the spectral radius of the iteration matrix. For many PDE problems, it can be infinitely close to one as the mesh size increases. Therefore, stationary iterative methods are not very effective in general despite their simplicity and low cost per iteration.

2.2 Krylov Subspace Methods

Krylov subspace (KSP) methods are widely used for solving large and sparse linear systems. For symmetric systems, conjugate gradient (CG) [42] and MINRES [56] are well recognized as the best KSP methods [28]. For nonsymmetric systems, various KSP methods have been developed, such as GMRES [60], CGS [65], QMR [30], TFQMR [29], BiCGSTAB [69], QMRCGSTAB [18], etc. Each method has its own advantages and disadvantages. In this section, We focus on the Krylov subspaces and the procedure in constructing the basis vectors of the subspaces, which are often the determining factors in the overall performance of different types of KSP methods. We defer more detailed discussions and analysis to [5, 59, 70].

Given a matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ and a vector $\mathbf{v} \in \mathbb{R}^n$, the kth *Krylov subspace* generated by them, denoted by $\mathscr{K}_k(\mathbf{A},\mathbf{v})$, is given by

$$\mathscr{K}_k(\mathbf{A},\mathbf{v}) = \operatorname{span}\{\mathbf{v}, \mathbf{A}\mathbf{v}, \mathbf{A}^2\mathbf{v}, \ldots, \mathbf{A}^{k-1}\mathbf{v}\}. \tag{2.2.1}$$

Let $\mathbf{x}_0$ be some initial guess to the solution, and $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$ be the initial residual vector. A Krylov subspace method incrementally finds approximate solutions within $\mathscr{K}_k(\mathbf{A},\mathbf{v})$, sometimes through the aid of another Krylov subspace $\mathscr{K}_k(\mathbf{A}^T,\mathbf{w})$, where $\mathbf{v}$ and $\mathbf{w}$ typically depend on $\mathbf{r}_0$. To construct the basis of the subspace $\mathscr{K}(\mathbf{A},\mathbf{v})$, two procedures are commonly used: the (restarted) *Arnoldi iteration* [3], and the *bi-Lanczos iteration* [48, 70] (a.k.a. Lanczos biorthogonalization [59] or tridiagonal biorthogonalization [66]).

2.2.1 The Arnoldi/Lanczos Iteration

The Arnoldi iteration is a procedure for constructing orthogonal basis of the Krylov subspace $\mathscr{K}(\mathbf{A},\mathbf{v})$. Starting from a unit vector $\mathbf{q}_1 = \mathbf{v}/\|\mathbf{v}\|$, it iteratively constructs

$$\mathbf{Q}_{k+1} = [\mathbf{q}_1 \mid \mathbf{q}_2 \mid \cdots \mid \mathbf{q}_k \mid \mathbf{q}_{k+1}] \tag{2.2.2}$$

with orthonormal columns by solving

$$h_{k+1,k}\mathbf{q}_{k+1} = \mathbf{A}\mathbf{q}_k - h_{1k}\mathbf{q}_1 - \cdots - h_{kk}\mathbf{q}_k, \tag{2.2.3}$$

where $h_{ij} = \mathbf{q}_i^T \mathbf{A}\mathbf{q}_j$ for $j \leq i$, and $h_{k+1,k} = \|\mathbf{A}\mathbf{q}_k - h_{1k}\mathbf{q}_1 - \cdots - h_{kk}\mathbf{q}_k\|$, i.e., the norm of the right-hand side of (2.2.3). This is analogous to Gram-Schmidt orthogonalization. If $\mathscr{K}_k \neq \mathscr{K}_{k-1}$, then the columns of $\mathbf{Q}_k$ form an orthonormal basis of $\mathscr{K}_k(\mathbf{A}, \mathbf{v})$, and

$$\mathbf{A}\mathbf{Q}_k = \mathbf{Q}_{k+1}\tilde{\mathbf{H}}_k, \tag{2.2.4}$$

where $\tilde{\mathbf{H}}_k$ is a $(k+1) \times k$ upper Hessenberg matrix, whose entries h_{ij} are those in (2.2.3) for $i \leq j+1$, and $h_{ij} = 0$ for $i > j+1$.

The KSP method GMRES [60] is based on the Arnoldi iteration, with $\mathbf{v} = \mathbf{r}_0$. At the kth iteration, it minimizes $\|\mathbf{r}_k\|_2$ in $\mathscr{K}_k(\mathbf{A}, \mathbf{b})$. Equivalently, it finds the optimal degree-k polynomial $\mathscr{P}_k(\mathbf{A})$ such that $\mathbf{r}_k = \mathscr{P}_k(\mathbf{A})\mathbf{r}_0$ and $\|\mathbf{r}_k\|$ is minimized. The Arnoldi iteration has a k-term recurrence, so its computational cost increases as k increases. For this reason, one almost always need to restart the Arnoldi iteration in practice, for example after every 30 iterations, to build a new Krylov subspace from $\mathbf{v} = \mathbf{r}_k$ at restart. Unfortunately, restarts also undermine the convergence of GMRES as the previous basis vectors are lost.

If $\mathbf{A}$ is symmetric, the Hessenberg matrix $\tilde{\mathbf{H}}_k$ reduces to a tridiagonal matrix, and the Arnoldi iteration reduces to the Lanczos iteration, which CG and MINRES are based on. At the kth iteration, MINRES minimizes $\|\mathbf{r}_k\|_2$ in $\mathscr{K}_k(\mathbf{A}, \mathbf{b})$ while CG minimizes the energy norm $\|\mathbf{r}_k\|_{\mathbf{A}} = \sqrt{\mathbf{r}_k^T \mathbf{A}\mathbf{r}_k}$. Unlike Arnoldi iteration, Lanczos iteration enjoys a three-term recurrence. Therefore CG and MINRES are able to preserve the complete Krylov subspace, making them the optimal choices for symmetric systems.

2.2.2 The Bi-Lanczos Iteration

The *bi-Lanczos* iteration, also known as *Lanczos biorthogonalization* or *tridiagonal biorthogonalization*, offers an alternative for constructing the basis of the Krylov subspaces of $\mathscr{K}(\mathbf{A}, \mathbf{v})$. Unlike Arnoldi iterations, the bi-Lanczos iterations enjoy a three-term recurrence. However, the basis will no longer be orthogonal, and we need to use two matrix-vector multiplications per iteration, instead of just one.

The bi-Lanczos iterations can be described as follows. Starting from the vector

$\mathbf{v}_1 = \mathbf{v}/\|\mathbf{v}\|$, we iteratively construct

$$\mathbf{V}_{k+1} = [\mathbf{v}_1 \mid \mathbf{v}_2 \mid \cdots \mid \mathbf{v}_k \mid \mathbf{v}_{k+1}], \tag{2.2.5}$$

by solving

$$\beta_k \mathbf{v}_{k+1} = \mathbf{A}\mathbf{v}_k - \gamma_{k-1}\mathbf{v}_{k-1} - \alpha_k \mathbf{v}_k, \tag{2.2.6}$$

analogous to (2.2.3). If $\mathscr{K}_k \neq \mathscr{K}_{k-1}$, then the columns of $\mathbf{V}_k$ form a basis of $\mathscr{K}_k(\mathbf{A}, \mathbf{v})$, and

$$\mathbf{A}\mathbf{V}_k = \mathbf{V}_{k+1}\tilde{\mathbf{T}}_k, \tag{2.2.7}$$

where

$$\tilde{\mathbf{T}}_k = \begin{bmatrix} \alpha_1 & \gamma_1 & & & \\ \beta_1 & \alpha_2 & \gamma_2 & & \\ & \beta_2 & \alpha_3 & \ddots & \\ & & \ddots & \ddots & \gamma_{k-1} \\ & & & \beta_{k-1} & \alpha_k \\ & & & & \beta_k \end{bmatrix} \tag{2.2.8}$$

is a $(k+1) \times k$ tridiagonal matrix. To determine the α_i and γ_i, we construct another Krylov subspace $\mathscr{K}(\mathbf{A}^T, \mathbf{w})$, whose basis is given by the column vectors of

$$\mathbf{W}_{k+1} = [\mathbf{w}_1 \mid \mathbf{w}_2 \mid \cdots \mid \mathbf{w}_k \mid \mathbf{w}_{k+1}], \tag{2.2.9}$$

subject to the biorthogonality condition

$$\mathbf{W}_{k+1}^T \mathbf{V}_{k+1} = \mathbf{V}_{k+1}^T \mathbf{W}_{k+1} = \mathbf{I}_{k+1}. \tag{2.2.10}$$

Since

$$\mathbf{W}_{k+1}^T \mathbf{A}\mathbf{V}_k = \mathbf{W}_{k+1}^T \mathbf{V}_{k+1}\tilde{\mathbf{T}}_k = \tilde{\mathbf{T}}_k, \tag{2.2.11}$$

it then follows that

$$\alpha_k = \mathbf{w}_k^T \mathbf{A}\mathbf{v}_k. \tag{2.2.12}$$

Suppose $\mathbf{V} = \mathbf{V}_n$ and $\mathbf{W} = \mathbf{W}_n = \mathbf{V}^{-T}$ form complete basis vectors of $\mathscr{K}_n(\mathbf{A}, \mathbf{v})$ and

$\mathscr{K}_n(\mathbf{A}^T, \mathbf{w})$, respectively. Let $\mathbf{T} = \mathbf{V}^{-1}\mathbf{A}\mathbf{V}$ and $\mathbf{S} = \mathbf{T}^T$. Then,

$$\mathbf{W}^{-1}\mathbf{A}^T\mathbf{W} = \mathbf{V}^T\mathbf{A}^T\mathbf{V}^{-T} = \mathbf{T}^T = \mathbf{S}, \tag{2.2.13}$$

and

$$\mathbf{A}^T\mathbf{W}_k = \mathbf{W}_{k+1}\tilde{\mathbf{S}}_k, \tag{2.2.14}$$

where $\tilde{\mathbf{S}}_k$ is the leading $(k+1) \times k$ submatrix of $\mathbf{S}$. Therefore,

$$\gamma_k \mathbf{w}_{k+1} = \mathbf{A}^T\mathbf{w}_k - \beta_{k-1}\mathbf{w}_{k-1} - \alpha_k \mathbf{w}_k. \tag{2.2.15}$$

Starting from $\mathbf{v}_1$ and $\mathbf{w}_1$ with $\mathbf{v}_1^T\mathbf{w}_1 = 1$, and let $\beta_0 = \gamma_0 = 1$ and $\mathbf{v}_0 = \mathbf{w}_0 = \mathbf{0}$. Then, α_k is uniquely determined by (2.2.12), and β_k and γ_k are determined by (2.2.6) and (2.2.15) by up to scalar factors, subject to $\mathbf{v}_{k+1}^T\mathbf{w}_{k+1} = 1$. A typical choice is to scale the right-hand sides of (2.2.6) and (2.2.15) by scalars of the same modulus [59, p. 230].

If $\mathbf{A}$ is symmetric and $\mathbf{v}_1 = \mathbf{w}_1 = \mathbf{v}/\|\mathbf{v}\|$, then the bi-Lanczos iteration reduces to the classical Lanczos iteration for symmetric matrices. Therefore, it can be viewed as a different generalization of the Lanczos iteration to nonsymmetric matrices. Unlike the Arnoldi iteration, the cost of bi-Lanczos iteration is fixed per iteration, which may be advantageous in some cases. Some KSP methods, in particular BiCG [26] and QMR [30], are based on bi-Lanczos iterations. At *k*th iteration, QMR minimizes the pseudonorm $\|\mathbf{r}_k\|_{\mathbf{W}_{k+1}^T}$ while the behavior of BiCG is more random. A potential issue of bi-Lanczos iteration is that it suffers from *breakdown* if $\mathbf{v}_{k+1}^T\mathbf{w}_{k+1} = 0$ or *near breakdown* if $\mathbf{v}_{k+1}^T\mathbf{w}_{k+1} \approx 0$. These can be resolved by a *look-ahead* strategy to build a block-tridiagonal matrix $\mathbf{T}$. Fortunately, breakdowns are rare in practice, so look-ahead is rarely implemented.

A disadvantage of the bi-Lanczos iteration is that it requires the multiplication with $\mathbf{A}^T$. Although $\mathbf{A}^T$ is in principle available in most applications, multiplication with $\mathbf{A}^T$ leads to additional difficulties in performance optimization and preconditioning. Fortunately, in bi-Lanczos iteration, $\mathbf{V}_k$ can be computed without forming $\mathbf{W}_k$ and vice versa. This observation leads to the transpose-free variants of the KSP methods, such as TFQMR [29], which is a transpose-free variant of QMR, and CGS [65], which is a transpose-free variant of BiCG. Two other examples include

Table 2.1: Comparisons of KSP methods.

Method	Iteration	Matrix-Vector Prod.		Recurrence
		A^T	A	
GMRES [60]	Arnoldi	0	1	k
BiCG [26]	bi-Lanczos	1	1	3
QMR [30]				
CGS [65]	TF bi-Lanczos 1	0	2	
TFQMR [29]				
BiCGSTAB [69]	TF bi-Lanczos 2			
QMRCGSTAB [18]				

BiCGSTAB [69], which is more stable than CGS, and QMRCGSTAB [18], which is a hybrid of QMR and BiCGSTAB, with smoother convergence than BiCGSTAB. These transpose-free methods enjoy three-term recurrences and require two multiplications with $\mathbf{A}$ per iteration.

2.2.3 Comparison of the Iteration Procedures

Both the Arnoldi iteration and the bi-Lanczos iteration are based on the Krylov subspace $\mathscr{K}(\mathbf{A}, \mathbf{r}_0)$. However, these iteration procedures have very different properties, which are inherited by their corresponding KSP methods, as summarized in Table 2.1. These properties, for the most part, determine the cost per iteration of the KSP methods. For KSP methods based on the Arnoldi iteration, at the kth iteration the residual $\mathbf{r}_k = \mathscr{P}_k(\mathbf{A})\mathbf{r}_0$ for some degree-k polynomial $\mathscr{P}_k$, so the asymptotic convergence rates depend on the eigenvalues and the generalized eigenvectors in the Jordan form of $\mathbf{A}$ [54, 59]. For methods based on transpose-free bi-Lanczos, in general $\mathbf{r}_k = \hat{\mathscr{P}}_k(\mathbf{A})\mathbf{r}_0$, where $\hat{\mathscr{P}}_k$ is a polynomial of degree $2k$. Therefore, the convergence of these methods also depend on the eigenvalues and generalized eigenvectors of $\mathbf{A}$, but at different asymptotic rates. Typically, the reduction of error in one iteration of a bi-Lanczos-based KSP method is approximately equal to that of two iterations in an Arnoldi-based KSP method. Since the Arnoldi iteration requires only one matrix-vector multiplication per iteration, compared to two per iteration for the bi-Lanczos iteration, the cost of different KSP methods are comparable in terms of the number of matrix-vector multiplications. Theoretically, the Arnoldi iteration is more robust because of its use of orthogonal basis, whereas the bi-Lanczos iteration may break down if $\mathbf{v}_{k+1}^T \mathbf{w}_{k+1} = 0$. However, the Arnoldi iteration typically requires restarts, which can undermine convergence.

In general, if the iteration count is small compared to the average number of nonzeros per row, the methods based on the Arnoldi iteration may be more efficient; if the iteration count is large, the cost of orthogonalization in Arnoldi iteration may become higher than that of bi-Lanczos iteration. On the other hand, the apparent disadvantages of each KSP method may be overcome by effective preconditioners: For Arnoldi iterations, if the KSP method converges before restart is needed, then it may be the most effective method; for bi-Lanczos iterations, if the KSP method converges before any break down, it is typically more robust than the methods based on restarted Arnoldi iterations.

Note that some KSP methods use a Krylov subspace other than $\mathscr{K}(\mathbf{A}, \mathbf{r}_0)$. The most notable examples are LSQR [57] and LSMR [27], which use the Krylov subspace $\mathscr{K}(\mathbf{A}^T\mathbf{A}, \mathbf{A}^T\mathbf{r}_0)$. These methods are mathematically equivalent to applying CG or MINRES to the normal equation, respectively, but with better numerical properties. An advantage of these methods is that they are applicable to least squares systems without modification. However, they tend to converge slowly for square linear systems, and they require special preconditioners.

2.3 Preconditioners

The convergence of KSP methods can be improved significantly by the use of preconditioners. Various preconditioners have been proposed for the Krylov subspace methods over the past few decades. We focus on the three commonly used preconditioners: SOR, incomplete LU factorization, and algebraic multigrid.

2.3.1 Left and Right Preconditioners

Roughly speaking, a preconditioner is a matrix or transformation $\mathbf{M}$, whose inverse $\mathbf{M}^{-1}$ approximates $\mathbf{A}^{-1}$, and $\mathbf{M}^{-1}\mathbf{v}$ can be computed efficiently. For nonsymmetric linear systems, a preconditioner may be applied either to the left or the right of $\mathbf{A}$. With a left preconditioner, one solves the modified system

$$\mathbf{M}^{-1}\mathbf{A}\mathbf{x} = \mathbf{M}^{-1}\mathbf{b} \tag{2.3.1}$$

by utilizing the Krylov subspace $\mathscr{K}(\mathbf{M}^{-1}\mathbf{A}, \mathbf{M}^{-1}\mathbf{b})$ instead of $\mathscr{K}(\mathbf{A}, \mathbf{b})$. For a right preconditioner, one solves the linear system

$$\mathbf{A}\mathbf{M}^{-1}\mathbf{y} = \mathbf{b} \tag{2.3.2}$$

by utilizing the Krylov subspace $\mathscr{K}(\mathbf{A}\mathbf{M}^{-1}, \mathbf{b})$, and then $\mathbf{x} = \mathbf{M}^{-1}\mathbf{y}$. The convergence of a preconditioned KSP method is then determined by the eigenvalues of $\mathbf{M}^{-1}\mathbf{A}$, which are the same as those of $\mathbf{A}\mathbf{M}^{-1}$. Qualitatively, $\mathbf{M}$ is a good preconditioner if $\mathbf{M}^{-1}\mathbf{A}$ is not too far from normal and its eigenvalues are more clustered than those of $\mathbf{A}$ [66].

Although the left and right-preconditioners have similar asymptotic behavior, they can behave drastically differently in practice. This is because the termination criterion of a Krylov subspace method is typically based on the norm of the residual of the unpreconditioned system. With a left preconditioner, the preconditioned residual $\|\mathbf{M}^{-1}\mathbf{r}_k\|$ may differ significantly from the true residual $\|\mathbf{r}_k\|$ if $\|\mathbf{M}^{-1}\|$ is far from 1, which unfortunately is often the case. This in turn leads to erratic behavior, such as premature termination or false stagnation of the preconditioned KSP method, unless the true residual is calculated explicitly at the cost of additional matrix-vector multiplications. In contrast, a right preconditioner does not alter the residual, so the stopping criteria can use the true residual with little or no extra cost.

Note that a preconditioner may also be applied to both the left and right of $\mathbf{A}$, leading to the so-called *symmetric preconditioners*. Such preconditioners are more commonly used for preserving the symmetry of symmetric matrices so that CG and MINRES can be applied. Like left preconditioners, they also alter the norm of the residual.

2.3.2 SOR

The SOR preconditioner is simply the iteration matrix $\mathbf{M}$ defined in the SOR iteration. When $\omega = 1$, the SOR reduces to Gauss-Seidel; when $\omega > 1$ and $\omega < 1$, it corresponds to over-relaxation and under-relaxation, respectively. It also has a symmetric variant, called SSOR, with the following definition

$$\mathbf{M} = \frac{1}{2-\omega}\left(\frac{1}{\omega}\mathbf{D}+\mathbf{L}\right)\left(\frac{1}{\omega}\mathbf{D}\right)^{-1}\left(\frac{1}{\omega}\mathbf{D}+\mathbf{L}\right)^{T}. \tag{2.3.3}$$

Like their counterparts in iterative solvers, the SOR type preconditioners are simple to implement but their convergence is usually poor. Unless a good ω is available, a safe choice is to use the Gauss-Seidel preconditioner. One thing to note here is that SOR type preconditioners require the diagonal entries of $\mathbf{A}$ to be nonzero.

2.3.3 Incomplete LU Factorization

Incomplete LU factorization (ILU) is one of the most widely used black-box preconditioners. It performs an approximate factorization

$$\mathbf{A} \approx \mathbf{LU}, \tag{2.3.4}$$

where $\mathbf{L}$ and $\mathbf{U}$ are far sparser than those in the true LU factorization of $\mathbf{A}$. When $\mathbf{A}$ is symmeric and positive definite, the factorization becomes the incomplete Cholesky factorization. In its simplest form, ILU does not introduce any fill, so that $\mathbf{L}$ and $\mathbf{U}$ preserve the sparsity patterns of the lower and upper triangular parts of $\mathbf{A}$, respectively. In this case, the diagonal entries of $\mathbf{A}$ must be nonzero. The ILU may be extended to preserve relatively large fills, and to use partial pivoting. These improve the stability of the factorization, but also increases the computational cost and storage. In general, the ILU factorization is more effective than SOR type preconditioners.

2.3.4 Algebraic Multigrid

Multigrid methods, including geometric multigrid (GMG) and algebraic multigrid (AMG), are the most sophisticated preconditioners. These methods are typically based on stationary iterative methods, and they accelerate the convergence by constructing a series of coarser representations. Compared to SOR and ILU, multigrid preconditioners are far more difficult to implement. Fortunately, AMG preconditioners are easily accessible through software libraries. There are primarily two types of AMG methods: the classical AMG and smoothed aggregation. An efficient implementation of the former is available in Hyper [25], and that of the latter is available in ML [32]. Both packages are accessible through PETSc.

Computationally, AMG is more expensive than SOR and ILU in terms of both setup time and cost per iteration, but they are also more scalable in problem size. Therefore, they are beneficial only if they can accelerate the convergence of KSP methods significantly, and the problem size is sufficiently large. In general, the classical AMG is more expensive than smoothed aggregation in terms of cost per iteration, but it tends to converge much faster. Depending on the types of the problems, the classical AMG may outperform smoothed aggregation and vice versa.

Figure 2.1 shows the residual plots of various preconditioned nonsymmetric Krylov subspace methods. The test matrices are generated from applying the finite element method to 2-D and 3-D convection-diffusion equations with unstuctured meshes. The Krylov subspace methods used are GMRES that restarts every 30 iterations, BiCGSTAB, TFQMR and QMRCGSTAB. Regardless of the underlying solver, we can observe that ILU is more effective than Gauss-Seidel while the ML preconditioner is significantly better than the two. In addition, ML also make GMRES(30) comparable or even faster than the methods based on bi-Lanczos iteration. The timing result shown in Figure 2.2 is consistent with the convergence behaviors. The superior performance of the algebraic multigrid preconditioner motivates us to develop more efficient geometric and hybrid multigrid preconditioners.

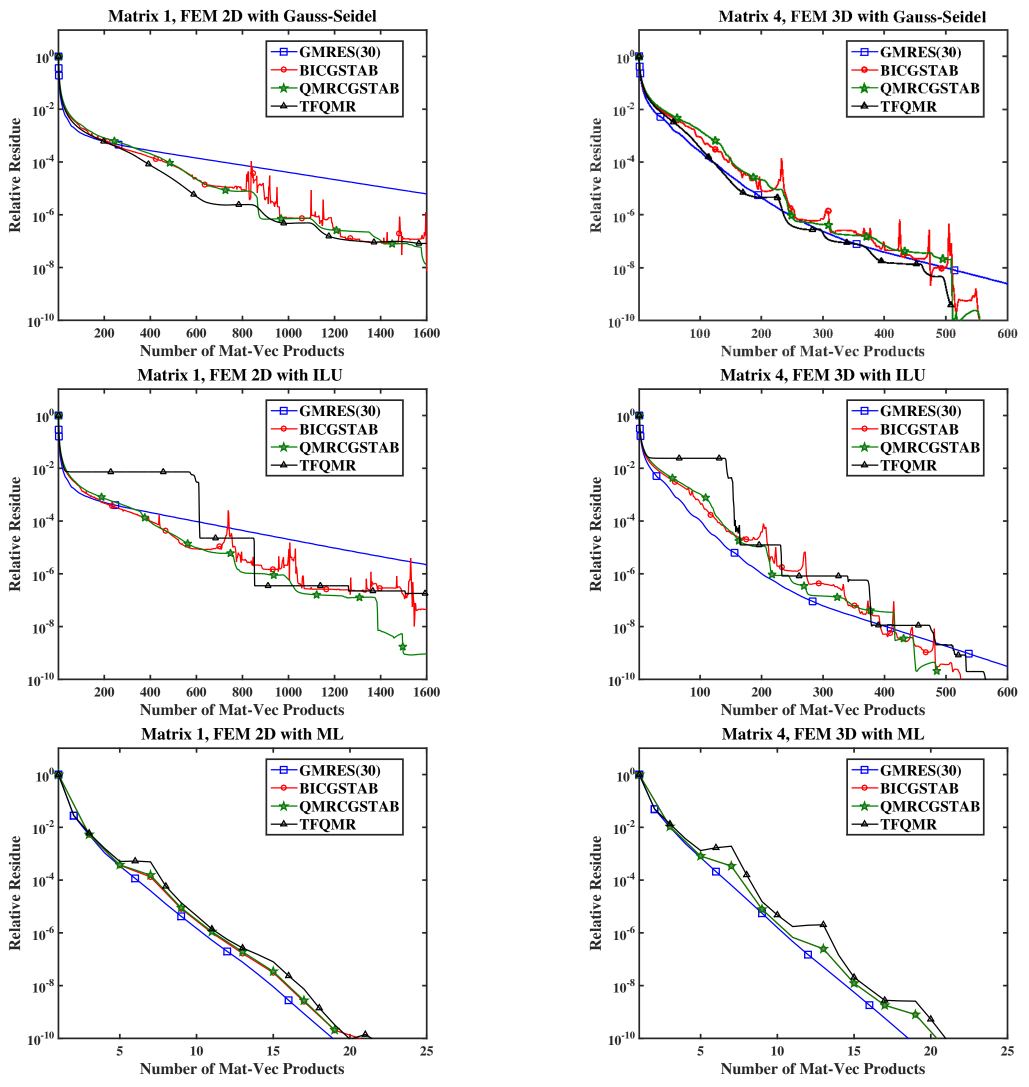
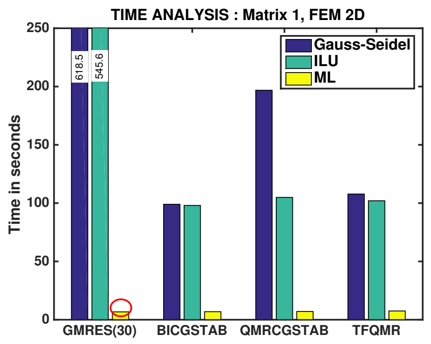


Figure 2.1: Relative residuals versus numbers of matrix-vector multiplications for Gauss-Seidel, ILU and ML preconditioners.



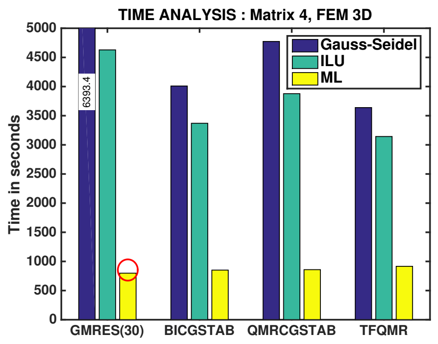


Figure 2.2: Timing results for the preconditioned solvers.

2.4 Methods for Singular Systems

Singular systems are inherently more difficult to solve than nonsingular systems. First, they are typically ill-conditioned in terms of the nonzero eigenvalues. When a system is singular, chances are the smallest nonzero eigenvalues are close to zero, which leads to ill-conditioning. Second, the solution does not exist if **b** has a component in $\text{null}\left(\mathbf{A}^T\right)$. In some applications, **A** is symmetric and the system can be made compatible by projecting $\text{null}(\mathbf{A})$ off **b**. However we do not know about $\text{null}(\mathbf{A})$ or $\text{null}\left(\mathbf{A}^T\right)$ in general. In this case, we usually seek a solution **x** that minimizes the 2-norm of the residual, i.e. a least-squares solution. There are infinitely many least-squares solution as the addition of any vector in $\text{null}(\mathbf{A})$ preserves the residual. In many applications, an additional constraint is added to ensure that the solution has minimum norm:

$$\min_{\mathbf{x}} \|\mathbf{x}\| \text{ subject to } \min_{\mathbf{x}} \|\mathbf{b} - \mathbf{A}\mathbf{x}\|. \tag{2.4.1}$$

Under this condition, the solution is uniquely determined. It is physically meaningful as it is free of spurious motion or energy. In the literature, the minimum-norm least-squares solution is also referred as the pseudo-inverse solution. An interesting observation is that when $\|\mathbf{b} - \mathbf{A}\mathbf{x}\|$ is minimized, $\mathbf{b} - \mathbf{A}\mathbf{x} \in \text{null}\left(\mathbf{A}^T\right)$. If **x** has minimum norm, $\mathbf{x} \perp \text{null}(\mathbf{A})$, which means $\mathbf{x} \in \text{range}\left(\mathbf{A}^T\right)$. Therefore the pseudo-inverse solution also satisfies the following conditions

$$\mathbf{x} \in \operatorname{range}\left(\mathbf{A}^T\right) \text{ and } \mathbf{A}^T\mathbf{A}\mathbf{x} = \mathbf{A}^T\mathbf{b}. \tag{2.4.2}$$

Finding the pseudo-inverse solution is quite challenging, especially when $\mathbf{A}$ is large and sparse. For direct methods, truncated SVD finds the pseudo-inverse solution regardless of the shape of the matrix. QR with column pivoting on $\mathbf{A}$ is able to find the least-squares solution. However it generally does not find the minimum-norm solution since we only have $\operatorname{range}(\mathbf{A})$. Among the iterative methods, most existing Krylov-subspace methods are applicable only to some sub-classes of linear systems. In particular, if $\mathbf{A}$ is Hermitian, then MINRES and MINRES-QLP [19] can be applied to consistent and inconsistent systems, respectively. The reason that they are able to find the minimum-norm solution is that the Krylov subspace $\mathscr{K}_k(\mathbf{A},\mathbf{b})$ coincides with $\mathscr{K}_k(\mathbf{A}^T,\mathbf{b})$. For nonsymmetric systems, GMRES can produce a least-squares solution for some consistent or inconsistent systems, but it may breakdown for more general problems [58]. Other nonsymmetric Krylov-subspace methods, including CGS, BiCG-STAB, QMR and TFQMR are unable to find the least-squares solution as they do not minimize the 2-norm of the residual. Some more general methods, such as LSQR [57] and LSMR [27], are mathematically equivalent to solving the normal equation $\mathbf{A}^T\mathbf{A}\mathbf{x} = \mathbf{A}^T\mathbf{b}$ and hence can always find the pseudo-inverse solution. These methods are remarkably robust, but their asymptotic convergence rates may be slow in practice, so they are used typically as the last resort when other methods fail to converge. More comprehensive discussions on these methods can be found in textbooks such as [38] and [59].

Chapter 3

Multigrid Methods

In this chapter, we review one of the most efficient iterative solvers, the multigrid methods. These methods make not only good standalone solvers but also effective preconditioners. They utilize stationary iterative methods such as Jacobi, Gauss-Seidel, or SOR to smooth out high-frequency errors, and accelerate the convergence of solution by transferring the residual and correction vectors across different levels via the *interpolation* and *restriction* operators. Depending on how the components are constructed, the multigrid methods can be further classified into geometric multigrid (GMG) and algebraic multigrid (AMG). We will go over the some of the key ideas behind the design of multigrid algorithms and present the data structure for software implementations.

3.1 Principles of Multigrid Methods

For multigrid methods, one of the most frequently asked questions is why they work so well. Before delving into details, we first present an important property about the stationary iterative methods: the smoothing effect.

3.1.1 Smoothing Effect

As we mentioned in the previous section, stationary iterative methods have a poor convergence rate. If we plot errors versus the number of iterations, we can see that

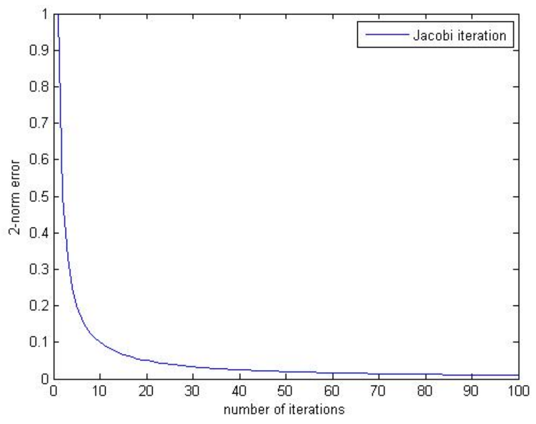


Figure 3.1: After 5 Jacobi iterations on a Poisson equation, error decreases very slowly.

errors decrease very quickly in the first several iterations, but they tend to change little after that. The reason behind this phenomenon is what we call the smoothing property of stationary iterative methods.

To illustrate the idea, let us apply weighted-Jacobi iteration to a 1-D Poisson equation where the iteration scheme and coefficient matrix $\mathbf{A}$ are given as

$$\mathbf{x}_{m+1} = \left(\mathbf{I} - \omega \mathbf{D}^{-1}\mathbf{A}\right)\mathbf{x}_m + \omega \mathbf{D}^{-1}\mathbf{b}, \qquad 0 \leqslant \omega \leqslant 1 \tag{3.1.1}$$

$$\mathbf{A} = \frac{1}{h^2}\begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & .. & .. & & \\ & & .. & .. & \\ & & & -1 & 2 \end{bmatrix}. \tag{3.1.2}$$

It can be calculated that the eigenvalues and eigenvectors of $\mathbf{A}$ are

$$\lambda_k(\mathbf{A}) = 4\sin^2\left(\frac{k\pi}{2n}\right), \qquad 1 \leqslant k \leqslant n-1 \tag{3.1.3}$$

$$(\omega_k)_j = \sin\left(\frac{jk\pi}{n}\right). \qquad 1 \leqslant k \leqslant n-1, \quad 1 \leqslant j \leqslant n-1 \tag{3.1.4}$$

The iteration matrix $\mathbf{R}_\omega = \left(\mathbf{I} - \omega \mathbf{D}^{-1}\mathbf{A}\right)$ has the same eigenvectors of $\mathbf{A}$ and its eigenvalues are

$$\lambda_k(\mathbf{R}_\omega) = 1 - 2\omega \sin^2\left(\frac{k\pi}{2n}\right). \qquad 1 \leqslant k \leqslant n-1 \tag{3.1.5}$$

With these results we can examine the effect of weighted Jacobian iteration on error $\mathbf{e}_m = \mathbf{x} - \mathbf{x}_m$ where $\mathbf{x}$ is the real solution. Subtracting $\mathbf{x} = \mathbf{R}_\omega \mathbf{x} + \omega \mathbf{D}^{-1}\mathbf{b}$ with (2.3), we obtain

$$\mathbf{e}_{m+1} = \mathbf{R}_\omega \mathbf{e}_m. \tag{3.1.6}$$

Using this recursive relation we can write

$$\mathbf{e}_m = \mathbf{R}_\omega^m \mathbf{e}_0. \tag{3.1.7}$$

Since the eigenvectors of $\mathbf{R}_\omega$ are given, $\mathbf{e}_0$ can be expressed as

$$\mathbf{e}_0 = \sum_{k=1}^{n-1} c_k \omega_k. \tag{3.1.8}$$

Here we can see that the error can be expanded using Fourier modes of high and low frequencies. After m iterations, the error $\mathbf{e}_m$ becomes

$$\mathbf{e}_m = \mathbf{R}_\omega^m e_0 = \sum_{k=1}^{n-1} \lambda_k^m c_k \omega_k. \tag{3.1.9}$$

This expansion of $\mathbf{e}_m$ shows that after m iterations, the k th component of the initial error has been reduced by a factor of $\lambda_k^m(\mathbf{R}_\omega)$. By letting $\omega = \frac{1}{2}$ in (2.4) we can observe that for $\frac{n}{2} \leqslant k \leqslant n-1$, $\lambda_k(\mathbf{R}_\omega) \leqslant \frac{1}{2}$ and this value is independent of h. On the other hand, for small k, λ_k can be very close to one. Take λ_1 for instance, we have the following estimate

$$\lambda_1 = 1 - 2\omega \sin^2\left(\frac{\pi}{2n}\right) = 1 - 2\omega \sin^2\left(\frac{\pi}{2n}\right) \approx 1 - \frac{\omega \pi^2 h^2}{2}. \tag{3.1.10}$$

This fact implies that λ_1, eigenvalue associated with the smoothest mode, will always be close to 1. In addition, the smaller the grid spacing h, the closer λ_1 is to 1. Therefore after several iterations, the impact of high frequency components in the error is not significant and the error becomes “smooth”.

Now we present an intuitive explanation for the smoothing effect. First let us introduce the idea of the residual equation. Let $\mathbf{x}_m$ be an approximate of the real solution $\mathbf{x}$ and $\mathbf{e}_m = \mathbf{x} - \mathbf{x}_m$. Then we have the following

$$\mathbf{A}\mathbf{e}_m = \mathbf{A}\left(\mathbf{x} - \mathbf{x}_m\right) = \mathbf{b} - \mathbf{A}\mathbf{x}_m = \mathbf{r_m}, \tag{3.1.11}$$

where $\mathbf{r}_m = \mathbf{b} - \mathbf{A}\mathbf{x}_m$ is the residual and we call (3.1.11) the residual equation.

With this definition, let us examine the jth component of the residual. If the error is geometrically smooth, $\left(\mathbf{e}_m\right)_{j-1}$, $\left(\mathbf{e}_m\right)_j$ and $\left(\mathbf{e}_m\right)_{j+1}$ should not differ much. Then we have

$$\left(\mathbf{r}_m\right)_j = \left(\mathbf{A}\mathbf{e}_m\right)_j = 2\left(\mathbf{e}_m\right)_j - \left(\mathbf{e}_m\right)_{j-1} - \left(\mathbf{e}_m\right)_{j+1} \approx 0. \tag{3.1.12}$$

This implies that smooth errors has small residuals. Subtracting (2.1.4) with $\mathbf{x}$ on both sides, we further obtain:

$$\mathbf{e}_{m+1} = \mathbf{e}_m + \omega\mathbf{D}^{-1}\mathbf{r}_m. \tag{3.1.13}$$

Since the weighted residual $\omega\mathbf{D}^{-1}r_m$ is small, we can conclude that the error will stay relatively unchanged.

3.1.2 Coarse-grid Correction

So far we established that many stationary iterative methods are effective in eliminating oscillatory components of the error while leaving out the smooth ones. This limitation keeps them from achieving fast convergence. On the other hand, we can make an interesting observation about the smooth error:

If the error is smooth, can we somehow solve a smaller problem and interpolate the solution?

In fact, the answer is yes, and this leads us to the famous correction scheme [16]. To be more specific, let $\mathbf{A}^{(1)} = \mathbf{A}$ be the coefficient matrix on the finer level $\Omega^{(1)}$, and $\mathbf{A}^{(2)}$ be that on the coarser level $\Omega^{(2)}$. A two-grid method can be outlined as follows:

Pre-smoothing: starting from the initial solution $\mathbf{x}_0$, run the *smoother* on $\mathbf{A}^{(1)}\mathbf{x}^{(1)} = \mathbf{b}$ for a few iterations to obtain $\mathbf{v}^{(1)}$;

Restriction: compute the residual vector $\mathbf{r}^{(1)} = \mathbf{b} - \mathbf{A}^{(1)}\mathbf{v}^{(1)}$ and $\mathbf{r}^{(2)} = \mathbf{R}\mathbf{r}^{(1)}$, where $\mathbf{R}$ is the *restriction matrix*;

Coarse-solve: construct the *coarse-grid operator* $\mathbf{A}^{(2)}$ and solve $\mathbf{A}^{(2)}\mathbf{e}^{(2)} = \mathbf{r}^{(2)}$;

Interpolation: compute the correction vector $\mathbf{e}^{(1)} = \mathbf{P}\mathbf{e}^{(2)}$, where $\mathbf{P}$ is the *interpolation matrix,* and update solution $\mathbf{v}^{(1)} \leftarrow \mathbf{v}^{(1)} + \mathbf{e}^{(1)}$;

Post-smoothing: run the smoother on $\mathbf{A}^{(1)}\mathbf{x}^{(1)} = \mathbf{b}^{(1)}$ for a few iterations with initial guess $\mathbf{v}^{(1)}$.

Although only two grids are presented here, one can easily extend the scheme to multiple grids, hence the name "multigrid". Now we will go through the process and explain the motivation behind each step:

Step 1: We run the smoother on the original problem. This process is usually referred to as presmoothing. Due to the smoothing effect, we expect the error $\mathbf{e}^{(1)} = \mathbf{x} - \mathbf{v}^{(1)}$ to be relatively smooth. If we can approximate the residual equation (3.1.11) on the coarse grid, $\mathbf{e}^{(1)}$ will appear oscillatory due to downsampling, which makes the smoothers more effective. In addition, solving the equation on $\Omega^{(2)}$ is cheaper.

Step 2: To approximate the residual equation on the coarse grid, we first compute the residual $\mathbf{r}^{(1)}$ on $\Omega^{(1)}$ and use a restriction operator to move it to $\Omega^{(2)}$.

Step 3: We then construct a coarse-grid operator $\mathbf{A}^{(2)}$ on $\Omega^{(2)}$. In principle, the coarse-grid operator should be a good approximation of $\mathbf{A}^{(1)}$. $\mathbf{A}^{(2)}\mathbf{e}^{(2)} = \mathbf{r}^{(2)}$ forms the residual equation on $\Omega^{(2)}$ and $\mathbf{e}^{(2)}$ is the coarse grid version of $\mathbf{e}^{(1)}$. To solve this equation we can use direct solvers, the Krylov subspace methods or smoothers with a zero initial guess. All approaches will benefit from the fact that $\Omega^{(2)}$ is smaller than $\Omega^{(1)}$. In addition, as $\mathbf{e}^{(1)}$ appears more oscillatory on $\Omega^{(2)}$, smoothers become more effective at reducing errors[1].

[1]In practice, more levels are used to make the coarsest grid sufficiently small

Step 4: Assume we have solved $\mathbf{A}^{(2)}\mathbf{e}^{(2)} = \mathbf{r}^{(2)}$ accurately, we can then interpolate $\mathbf{e}^{(2)}$ to obtain $\mathbf{e}^{(1)}$ and update the solution $\mathbf{v}^{(1)} \leftarrow \mathbf{v}^{(1)} + \mathbf{e}^{(1)}$. If the error is smooth, we expect the interpolation to be relatively accurate. Therefore we essentially solved $\mathbf{A}^{(1)}\mathbf{e}^{(1)} = \mathbf{r}^{(1)}$ at a much lower cost.

Step 5: Finally with the updated solution, we may want to iterate a few steps to further reduce errors. This process is referred as postsmoothing.

As we can see the reason why the multigrid method works so well is the *complementarity between the smoothers and the coarse grid correction*. Iterating on the fine grid leaves smooth errors $\mathbf{e}^{(1)}$ and they appear more oscillatory on the coarse grid as $\mathbf{e}^{(2)}$. We can then solve for $\mathbf{e}^{(2)}$ effectively, and $\mathbf{e}^{(2)}$ will become a good approximation of $\mathbf{e}^{(1)}$ after interpolation. Finally with the correction step $\mathbf{v}^{(1)} \leftarrow \mathbf{v}^{(1)} + \mathbf{e}^{(1)}$, we will obtain a solution very close to $\mathbf{x}$. In other words, errors that cannot be eliminated effectively by smoothers are removed by the coarse grid correction.

The correction scheme also introduces the essential components of the multigrid algorithm:

1. **Grid hierarchy**: A hierarchy of *grids (levels)* with various resolutions.
2. **Coarse-grid operator**s: Smaller problems to be solved on coarser levels.
3. **Smoother**: A stationary iterative method.
4. **Coarse-grid solver**: A solver employed at the coarsest level, and its choices include stationary iterative methods, Krylov-subspace methods, and direct methods.
5. **Interpolation operator**s: Operators that transfer vectors to finer levels.
6. **Restriction operator**s: Operators that transfer vectors to coarser levels.

With these components defined, a multigrid algorithm typically repeats the process in a V-cycle, W-cycle or full-multigrid (FMG) cycle, as shown in Figure 3.2. In the following sections, we will provide more details about the construction of these components.

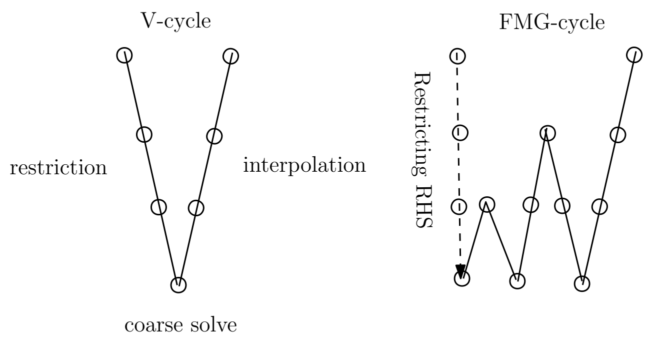


Figure 3.2: Illustration of V-cycle and FMG-cycle.

3.2 Geometric Multigrid

Geometric multigrid, by its name, utilizes the geometry of the PDEs. Many classical theories, such as Fourier analysis and polynomial approximation, can be used to aid in the construction of the multigrid components. For standard problems, it is guaranteed that GMG achieves linear time complexity regardless of the mesh size. The disadvantage is that the methods are often problem dependent. Different equations or even different discretization methods for the same equation can result in significant changes of the algorithm. Nevertheless, GMG is one of the most efficient iterative solvers if properly implemented.

Mesh Hierarchy. For structured meshes, the hierarchy is typically obtained by coarsening the original mesh until the mesh size is sufficiently small. The coarsening process can be done by merging adjacent cells to form bigger cells of the same element. For unstructured meshes, it is more difficult to aggregate cells while maintaining the element shape and quality. A dual approach is to refine a coarse initial mesh and solve the problem on the finest mesh [53, 44, 12]. This approach is substantially simpler and more effective, but it is less general and requires tighter integration with mesh generation. Figure 3.3 illustrates the process of mesh coarsening and refinement.

Smoother and Coarse-grid Operators. Once the hierarchy is generated, we need to define the problems to be solved on intermediate levels. The most natural way to approximate the original problem is to *rediscretize the PDEs*. It is not

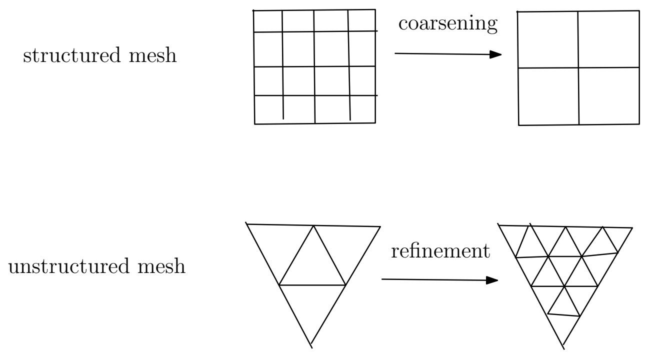


Figure 3.3: Generation of mesh hierarchy for GMG.

necessary to recompute the right hand side vectors, but they can be helpful in the initial FMG cycle. In terms of the smoothers, they should leave out errors that are *geometrically smooth*. Fortunately, it is usually sufficient to use standard methods like Jacobi and Gauss-Seidel, as long as the PDE has smooth coefficients and the meshes are close to uniform. For anisotropic problems, we need to consider more specialized methods, such as the line smoother in Chapter 4.

Transfer Operators. The interpolation operator is typically constructed by spatial interpolation. For example, we may use linear interpolation for triangles and bi-linear interpolation for quadrilaterals. For the case shown in Figure 3.4, the interpolation formula for the coarse grid vertex p can be written as

$$f(x,y) \approx w_1 f(q_1) + w_2 f(q_2) + w_3 f(q_3) + w_4 f(q_4), \tag{3.2.1}$$

$$\begin{aligned} w_1 &= \tfrac{1}{(x_2-x_1)(y_2-y_1)}(x_2-x)(y_2-y) \\ w_2 &= \tfrac{1}{(x_2-x_1)(y_2-y_1)}(x-x_1)(y_2-y) \\ w_3 &= \tfrac{1}{(x_2-x_1)(y_2-y_1)}(x-x_1)(y-y_1) \\ w_4 &= \tfrac{1}{(x_2-x_1)(y_2-y_1)}(x_2-x)(y-y_1). \end{aligned} \tag{3.2.2}$$

We refer w_i as the interpolation weights. If the grid hierarchy is generated by uniform refinement, the computation of weights are greatly simplified. For example, the weights for the case shown here are simply 0.25 for each q_i. Equivalently, the weights can be computed from the finite element basis functions, in particular the Lagrange basis functions. For a given vertex q_i, its Lagrange basis function ϕ_i has

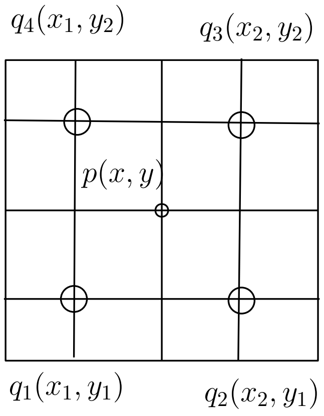


Figure 3.4: vertex p is enclosed by $q_1 - q_4$.

the following property

$$\phi_i(q) = \begin{cases} 1 & q = q_i \\ 0 & q \in \{q_j\}, \quad j \neq i \end{cases}. \tag{3.2.3}$$

Within element $\{q_1, q_2, q_3, q_4\}$, any bi-linear polynomial can be expressed as

$$f \approx \phi_1 f(q_1) + \phi_2 f(q_2) + \phi_3 f(q_3) + \phi_4 f(q_4). \tag{3.2.4}$$

A closer look at equation (3.2.2) reveals that the weights w_i are exactly $\phi_i(x,y)$.

The restriction operator can be defined in two ways. One approach is to interpolate from the fine grid points. The simplest example is the injection operator where the value is interpolated from the nearest fine grid point. Alternatively, we can use the scaled transpose of the interpolation operator, such as the *full-weighting* schemes in [67, 16]. For finite element methods, the transpose of interpolation coincides with the restriction of the basis functions. For finite difference methods, similar principles can be applied but the transposed matrix needs to be scaled so that the row sums are ones. More guidelines on the construction of the transfer operators are discussed in Chapter 5.

3.3 Algebraic Multigrid

Unlike GMG, where the geometry of problem plays a significant role, algebraic multigrid is based entirely on the linear system. It requires neither knowledge about the geometry nor how the PDE is solved. Due to this reason, AMG is an excellent black-box solver, and it has been implemented in various packages [25, 32, 8]. There are two variants of AMG: classical AMG [15] and smoothed aggregation [71]. Their differences mainly lie in the coarsening algorithm and the construction of the interpolation operator. We will focus on the derivation of classical AMG.

Algebraic Smoothness. Like any multigrid algorithm, AMG utilizes the idea of the complementarity between smoothers and coarse grid corrections. However, the smoothness of the error is now measured *algebraically*: the error at the $k+1st$ step is not much smaller than that of the kth step. From equation (2.1.4), we can loosely interpret this condition as *smooth errors have small residuals*, i.e.

$$\|\mathbf{M}(\mathbf{e}_{k+1}-\mathbf{e}_k)\| = \|\mathbf{r}_k\| \approx 0. \tag{3.3.1}$$

With a smooth error, we need to define a way to interpolate it from the coarse grid points, which leads us to the idea of *strong influence*. Since smooth errors have small residuals, we have

$$\mathbf{Ae} = \mathbf{r} \approx \mathbf{0}. \tag{3.3.2}$$

The ith row of the equation can be expressed as

$$e_i \approx -\frac{1}{a_{ii}} \sum_{j \neq i} a_{ij} e_j. \tag{3.3.3}$$

Of course, we cannot compute e_i directly with formula (3.3.3) since $\{e_j\}$ are available only from the coarse grid. However, we can expect a good approximation if the set includes variables with large a_{ij}, in other words, $\{e_j\}$ that strongly influence e_i. This observation suggests the following definition:

Definition 1. With $0 \leqslant \theta \leqslant 1$, the variable j strongly influences the variable i if

$$\left\|a_{ij}\right\| \geqslant \theta \max_{k \neq i} \left\|a_{ik}\right\|. \tag{3.3.4}$$

Similarly, the criterion of strong influence in smoothed aggregation is defined as $\left\|a_{ij}\right\| \geqslant \theta \sqrt{\left\|a_{ii}a_{jj}\right\|}$.

Algebraic Coarsening. Our goal is to select variables that strongly influence the other variables as the coarse grid points. This process is referred as *algebraic coarsening*. There are various algorithms available such as the classical Ruge-Stueben coarsening and the aggregation based coarsening. These algorithms represent the matrix via a graph with edges connecting vertices i and j if a_{ij} is nonzero. They typically include the following steps

1. For each vertex, decide its strongly influenced neighbors by definition 1.
2. Loop through all vertices in a certain order and choose an unprocessed vertex as a coarse grid point.
3. Once a coarse grid point is chosen, mark all its strongly influenced neighbors as fine grid points.
4. Repeat steps 2 and 3 until all points are processed.

Depending on the algorithm, there may be a post processing step that improves the distribution of the coarse grid points. Figure 3.5 illustrates a simple strategy. We first pick vertex 1 as a coarse grid point and mark vertices 2 and 4 as its fine grid neighbors. Then we move on to vertex 3 and mark vertex 6 as the fine grid point. The algorithm is repeated until we reach vertex 9. Since all of its neighbors are already processed, we mark vertex 9 as a coarse grid point and terminate the algorithm. For more complicated strategies, we refer the reader to [41, 71] .

Multigrid Operators. Depending on the AMG variant, the interpolation operator can be based on the classical interpolation [15] or the smoothed piece-wise constant

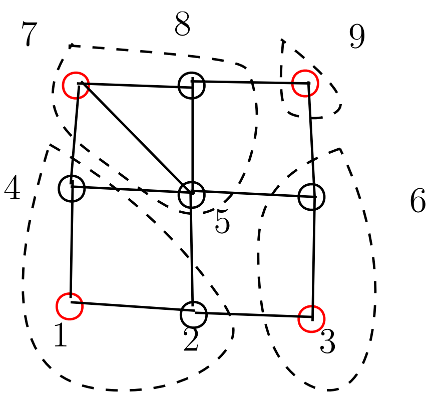


Figure 3.5: Example of algebraic coarsening.

interpolation [71]. Let e_i denotes the error associated with vertex i. For classical AMG, the interpolation operator is defined by:

$$P(e_i) = \begin{cases} e_i & \text{if } i \text{ is also in the coarse grid} \\ \sum w_{ij} e_j & \text{otherwise.} \end{cases} . \tag{3.3.5}$$

The weights $\{w_{ij}\}$ are computed by

$$w_{ij} = -\frac{a_{ij} + \sum_{m \in D_i^s} \left(\frac{a_{im} a_{mj}}{\sum_{k \in C_i} a_{mk}} \right)}{a_{ii} + \sum_{n \in D_i^w} a_{in}}, \tag{3.3.6}$$

where D_i^s, C_i and D_i^w are three sets partitioned by strong influence [16]:

- C_i denotes the neighboring coarse-grid points that strongly influence i;
- D_i^s denotes the neighboring fine-grid points that strongly influence i;
- D_i^w denotes the weakly connected neighbors.

The interpolation operator is expressed as a matrix $\mathbf{P}$, and the restriction operator $\mathbf{R}$ is defined as $\mathbf{P}^T$. Unlike GMG, the coarse grid operators for AMG are constructed algebraically by the *Galerkin formulation*. Let $\mathbf{P}$ be the interpolation operator, the coarse grid matrix $\mathbf{A}^{(2)}$ can then be written as

$$\mathbf{A}^{(2)} = \mathbf{P}^T \mathbf{A} \mathbf{P}. \tag{3.3.7}$$

Table 3.1: Computational times of GMG and PCG.

	GMG			PCG		
	Iter	Error	Time	Iter	Error	Time
x component	4	9.1e-7	0.7 sec	44	8.9e-7	2.8 sec
y component	4	6.6e-7		31	9.4e-7	
z component	6	7.3e-7		45	7.9e-7	

This formulation is much simpler than rediscretization as it only requires matrix-matrix multiplication with known matrices $\mathbf{A}$ and $\mathbf{P}$. In fact the Galerkin formulation is equivalent to rediscretization for some problems with finite element discretization. However $\mathbf{A}^{(2)}$ is in general much denser than that obtained by rediscretization. Recent efforts on AMG are focused on reducing operator complexity by truncating either the interpolation operator or the Galerkin product $\mathbf{P}^T\mathbf{AP}$ [22, 24].

3.4 Comparison of Iterative Solvers

One of the main motivations for the study of multigrid methods is its superior performance over the standard solvers. As an example, we consider the computation for the gradient vector flow field [74]

$$-\mu\triangle u(x) + |\nabla f(x)|^2 u(x) = |\nabla f(x)|^2 \nabla f(x), \qquad (3.4.1)$$

where $f(x)$ is the edge map provided as input. We applied GMG and PCG with SSOR to a $60\times 60\times 60$ test image with tolerance 10^{-6}. The results are summarized in Table 3.1. It can be observed that the multigrid method is about 4 times as fast as PCG. For larger data set, this advantage can be more significant. Other examples include the comparison of preconditioned solvers for nonsymmetric systems in Figure 2.2. In those examples, AMG is used as a preconditioner for the Krylov subspace methods. Its performance is significantly better than the alternatives such as GS and ILU.

Among the multigrid methods themselves, GMG and AMG have their own advantages and disadvantages, which we summarize in Table 3.2. The main advantages of GMG include its better convergence with smooth solution, better efficiency in terms

Table 3.2: Advantages and disadvantages of GMG and AMG, along with the design goals for HyGA.

	geometric MG	algebraic MG	HyGA
memory requirement	**low**	high	**low**
operator complexity	**less costly**	more costly	**less costly**
matrix quality	**high**	less control	**high at finer levels**
algorithm design	hard	**easier**	**intermediate**
user friendliness	less friendly	**more friendly**	**intermediate**

of computational time, and better efficiency in storage, especially with a matrix-free implementation. However, GMG algorithms are very problem dependent. It can be difficult to design the appropriate components for problems with anisotropic or discontinuous coefficients. In addition, it is hard to implement GMG to scale with unstructured meshes as there is no effective method for geometric coarsening. The advantages of AMG include its generality and flexibility. It can be used as a black-box solver without any knowledge about the PDE or its geometry. A single AMG algorithm can be applied to a wide range of problems, even ones with discontinuous coefficients. However, AMG tends to be more expensive and it heavily relies on the choice of parameters. It also tends to generate denser matrices than GMG, so it has higher memory and computational costs. We seek to propose a hybrid geometric+algebraic multigrid method that combines the advantages and overcome the disadvantages of the GMG and AMG. The resulting algorithm is HyGA, which will be discussed in Chapter 5.

3.5 Implementation

Compared to stationary iterative methods and the Krylov subspace methods, multigrid algorithms are more difficult to implement. Besides the original problem, the algorithm also solves a collection of smaller systems and transfers solutions back and forth. Therefore, we first need to define the appropriate data structure for the necessary operations. On each level, we solve a residual equation and transfer the solution. This requires storage of the solution vector, the right hand side vector, the residual vector, the coarse-grid operator, the interpolation operator and possibly the restriction operator. We can encapsulate them into a structure variable called Grid. In C, the Grid variable can be defined in the following fashion:

```
struct Grid {
  int    *row_ptr;   //coefficient matrix
  int    *col_ind;
  double *vals;

  int    *Rrow_ptr;  //restriction matrix
  int    *Rcol_ind;
  double *Rvals;

  int    *Prow_ptr;  //prolongation matrix
  int    *Pcol_ind;
  double *Pvals;

  double *u;        //Solution vector
  double *b;        //RHS      vector
  double *r;        //Residual vector
};
```

The operators are stored as matrices in the *compressed row storage (CRS)* format. For some geometric multigrid algorithms, one may implement the transfer operators in a matrix-free fashion. If direct methods are used as the coarse grid solver, we may need additional data structure at the coarsest level for the factorization matrices. Since there are multiple levels, we can use an array of Grid, named "Grids", to represent the whole multigrid data structure. On the finest level, we only need the coefficient matrix and the right hand side vector of the original system. They can be stored seperately from Grids or wrapped into a Grid variable with empty transfer operators.

Figure 3.6 shows the workflow of the algorithm. There are two main steps: a setup stage and the main loop. The advantage of seperating the steps is that we can use

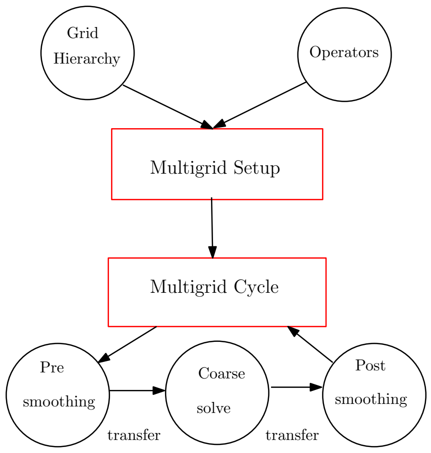


Figure 3.6: Multigrid workflow.

the same loop for GMG and AMG. The pseudo-codes are presented in Algorithms 1, 2 and 3. For convenience, the matrices are denoted by single letters instead of the CRS format.

Setup. The setup for AMG is very straightforward as everything can be done by algebraic coarsening. Since its restriction operator is the transpose of interpolaton, there is no need to store it in Grid. There are two types of setup for GMG. One is based on geometric coarsening for structured meshes, and the other is based on mesh refinement. The construction of the transfer operators requires parent-child information, which can be provided by the coarsening or refinement routines. The coarse grid operators are computed by the rediscretization routines, which need to be provided by users. Otherwise, one may use the Galerkin product if no other information is available. Note that for the GMG refinement setup, the problem to be solved is defined on the finest level. The input parameters are the original mesh, which corresponds to the bottom level in the multigrid structure.

Main Loop. Once the setup stage is complete, the main algorithm may follow a V-cycle, W-cycle or FMG-cycle. These cycles can be implemented via the same codes for AMG and GMG. Algorithm 4 gives the pseudo code for the V-cycle. Each V-cycle consists of an upward stroke and a downward stroke. In the downward stroke, we smooth the solutions, compute the residuals, restrict the residuals to a

Algorithm 1 AMG coarsening setup

```
input: $A$, $b$: the linear system
       $N$: number of levels
       $\theta$: coarsening strength
output: $Grids$: multigrid structure
1: [$Grids(1).A$, $Grids(1).P$] = coarsen ($A$,$\theta$)
2: for $k = 2, ..., N-1$
3:     [$Grids(k).A$, $Grids(k).P$] = coarsen ($Grids(k-1).A$, $\theta$)
4: end for
```

Algorithm 2 GMG coarsening setup

```
input: $A$, $b$: the linear system
       $xs$: vertex coordinates
       $elems$: element connectivity
       $N$: number of levels
output: $Grids$: multigrid structure
1: [$xs$,$elems$, $Grids(1).P$, $Grids(1).R$] = coarsen ($xs$,$elems$)
2: $Grids(1).A$ = rediscretize ($xs$,$elems$)
3: for $k = 2, ..., N-1$
4:     [$xs_c$, $elems_c$, $Grids(k).P$, $Grids(k).R$] = coarsen ($xs$,$elems$)
5:     $Grids(k).A$ = rediscretize ($xs$,$elems$)
6: end for
```

Algorithm 3 GMG refinement setup

```
input: $xs$: vertex coordinates
       $elems$: element connectivity
       $N$: number of levels
output: $Grids$: multigrid structure
        $A$: coefficient matrix on the finest level
        $b$: right hand side on the finest level
1: for $k = N-1, ..., 1$
2:     $Grids(k).A$ = rediscretize ($xs$,$elems$)
3:     [$xs$,$elems$, $Grids(k).P$, $Grids(k).R$] = refine ($xs$, $elems$)
4: end for
5: [$A$, $b$] = rediscretize ($xs$,$elems$)
```

lower level and initialize the solution there to be zeroes. At the bottom level, a coarse grid solver is applied to compute the solution with sufficient accuracy.[2] In the upward stroke, we interpolate the coarse grid solution, update the solution at the current level and smooth it again. The cycles are repeated until the norm of the residual is sufficiently small. FMG-cycles and W-cycles can be implemented in a similar fashion.

Algorithm 4 Multigrid V-cycle

```
input: $A$, $b$: the linear system
       $Grids$: multigrid structure
       $N$: number of levels
       $tol$: tolerance
output: $u$: the solution
 1: while $resnorm \leqslant tol$
 2:     $u$= smoother $(A, u, b)$
 3:     $r = b - A \cdot u$ and check $norm(r)$
 4:     $Grids(1).b = Grids(1).R \cdot r$
 5:     $Grids(1).u = \mathbf{0}$
 6:     for $k = 1, ..., N-2$
 7:         $[Grids(k).u]$ = smoother $(Grids(k).A, Grids(k).u, Grids(k).b)$
 8:         $Grids(k).r = Grids(k).b - Grids(k).A \cdot Grids(k).u$
 9:         $Grids(k+1).b = Grids(k+1).R \cdot Grids(k).r$
10:         $Grids(k+1).u = \mathbf{0}$
11:     end for
12:     $Grids(N-1).u$=solve $(Grids(N-1).A, Grids(N-1).u, Grids(N-1).b)$
13:     for $k = N-2, ..., 1$
14:         $Grids(k).r = Grids(k+1).P \cdot Grids(k+1).u$
15:         $Grids(k).u = Grids(k).u + Grids(k).r$
16:         $[Grids(k).u]$ = smoother $(Grids(k).A, Grids(k).u, Grids(k).b)$
17:     end for
18:     $r = Grids(1).P \cdot Grids(1).u$
19:     $u = u + r$
20:     $u$= smoother $(A, u, b)$
21: end while
```

[2] Although it is not necessary to apply direct solvers, one must ensure that the solution is accurate enough. Otherwise, the multigrid convergence can deteriorate significantly.

3.6 Performance Diagnosis

Sometimes the multigrid solvers may converge slowly or even diverge. There are several things we can check to diagnose the problem.

Smoother. The smoother is the most likely component to cause slow convergence or divergence. It also occupies the majority of the computational time in the main loop. Take the Gauss-Seidel iteration for example. It converges for symmetric and positive definite systems as well as systems with strong diagonal dominance. For other types of problems, it is not clear whether GS converges or not. If GS diverges, the multigrid solver diverges as well unless it is used as a preconditioner. Another potential problem is that the smoothing effect of GS gets worse for more complicated equations, such as the one in Chapter 4. As a result, the multigrid solver may converge at a slower rate. A quick fix is to increase the number of pre and post smoothing steps, particularly those at coarser levels. If the algorithm still does not work well, we need to consider switching to another smoother.

Coarse-grid Solver. Strictly speaking, the multigrid algorithms require the solution to be computed exactly on the coarsest level. This is not necessary in practice as a few iterations of smoothers or Krylov subspace methods are often enough. However, when the multigrid method starts to converge slowly, we need to check if the solution at the coarsest level is sufficiently accurate. If not, the convergence rate of the multigrid method deteriorates significantly. An example can be found in Chapter 4. In the worst case, we should always use a direct solver as the coarse-grid solver.

Interpolation. The synergy between smoothers and coarse-grid corrections also relies on the accuracy of the transfer operators. Even if the error is smooth, multigrid algorithms may converge slowly if the transfer operators are not accurate. For GMG, we can check the accuracy by multiplying the operator matrix with a vector on a coarser level. For example, if we use bi-linear interpolation, we can construct a bi-linear function and evaluate its values at coarse-grid points. We then multiply the interpolation matrix with the coarse-grid vector and see whether the resulting vector recovers the values on the fine grid exactly.

Parameters. This is more related to AMG. In AMG, there are a few parameters that can change the behavior of the algorithm significantly. First, there are various coarsening strategies to choose from. Some may be better than the others in some cases. Second, there are a few parameters in the coarsening algorithm. For classical Ruge-Stueben coarsening, there is a parameter θ that defines the threshold of strong influence. In 2-D, a good choice is $\theta = 0.25$. However in 3-D, it needs to be changed to a higher value such as $\theta = 0.7$. Third, we may want to consider different interpolation algorithms. One of the problems with AMG is that the operators are typically dense, which can be expensive in terms of both computational time and storage. One may try to reduce the complexity by combining aggressive coarsening with a more accurate interpolation matrix and vice versa.

Number of Levels. It is possible that adding more levels slows down the multi-grid convergence. This is particularly true if the coarse-grid operator is not a good approximation of its fine-grid counterpart. The best convergence rate of any multi-grid algorithm is actually that of a two-grid scheme with a direct solver on the second level. Of course, it is not practical to have only two levels. In general, we should coarsen the mesh until the problem size on the coarsest level is manage-able by the coarse-grid solver. If the coarse-grid solver does not affect the overall performance much, there is no reason to coarsen the mesh further.

Chapter 4

Geometric Multigrid for Anisotropic Helmholtz Equations

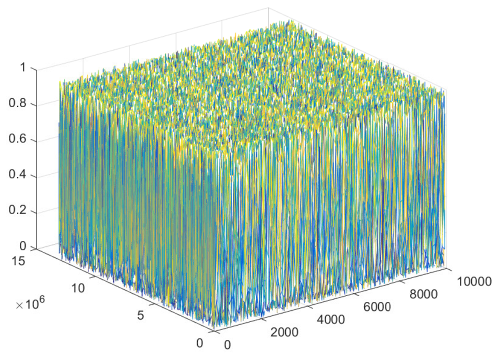
We present a specialized geometric multigrid solver for the 2-D anisotropic Helmholtz equation on structured grids. The general form of the equation can be expressed as:

$$-\nabla \cdot (K \nabla u) + \alpha u = f. \tag{4.0.1}$$

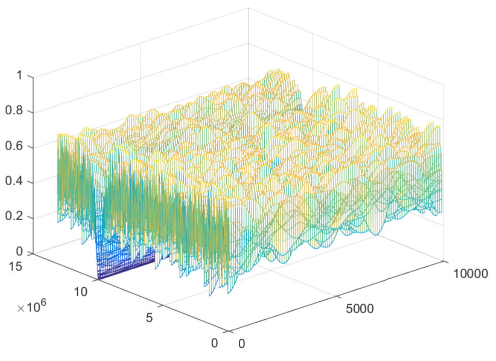
We consider cell-centered approximation with pure Neumann boundary condition. The grid spacing can be non-uniform and anisotropic. It is easy to see that when $\alpha = 0$, the problem is reduced to a singular Poisson equation. When α is small, the system is ill-conditioned. Special care must be taken when dealing with these cases as the solutions tend to be inaccurate. In the followng sections, we will discuss about the adaptation of the multigrid components in detail.

4.1 Line Smoother

The first change we need to make about the algorithm is the smoother. In Chapter 3, we mentioned that standard stationary iterative methods usually work well for GMG and AMG. For GMG, this is because the errors tend to be geometrically smooth. However, this is no longer the case when the grid spacing is anisotropic. As an example, let us apply standard Gauss-Seidel to a Helmholtz equation with $N_x =$



(a) Initial error.



(b) GS.

Figure 4.1: Initial error (left) and error of Gauss-Seidel (right).

3300, $N_y = 100$, $dx = 4000$, $dy = 100$ and $\alpha = 10^{-8}$. To evaluate the effectiveness of Gauss-Seidel, we solve the following homogeneous system:

$$\mathbf{Au} = \mathbf{0}. \tag{4.1.1}$$

$\mathbf{A}$ comes from discretizing equation (4.0.1) with a finite difference scheme. The initial guess is an oscillatory random vector. If the smoother works as intended, the approximated solution $\mathbf{u}_k$, which coincides with $\mathbf{e}_k$, should be geometrically smooth. Figure 4.1 plots the initial vector and the solution after 10 iterations. We can see that the error is not smooth at all! Without a smooth error, the other multigrid components will not work well either. Fortunately, the treatment of anisotropic problems is well studied in the literature [16, 67], and one of the solutions is to use line smoothers.

The line smoother, or plane smoother in 3-D, is a specialized block stationary iterative method for the Poisson type problem. To illustrate the idea, we express the structure of the matrix $\mathbf{A}$ as

$$\begin{pmatrix} \mathbf{A}_1 & \mathbf{B}_1 & & & \\ \mathbf{C}_1 & \mathbf{A}_2 & \mathbf{B}_2 & & \\ & \mathbf{C}_2 & \mathbf{A}_3 & \mathbf{B}_3 & \\ & & \dots & \dots & \\ & & & \mathbf{C}_{N_y-1} & \mathbf{A}_{N_y} \end{pmatrix}. \tag{4.1.2}$$

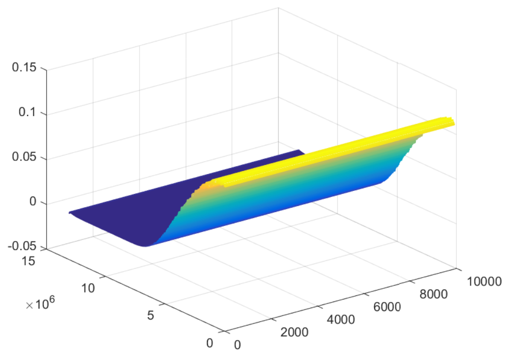
It is a block tri-diagonal matrix. A block smoother is essentially a standard smoother applied to these blocks. Algorithm 5 shows the details of the block Gauss-Seidel iteration. At each iteration, we solve N_y smaller systems of the size $N_x \times N_x$. Each coefficient matrix $\mathbf{A}_i$ is tri-diagonal, and the variables within that block are updated simultaneously. The right hand sides are computed by a combination of the current values and the updated values, just like the point-wise Gauss-Seidel. These smoothers turn out to be very effective for anisotropic problem when *strongly connected unknowns are updated together*. In our previous example, the coefficients in the x direction are much smaller than those in the y direction. This indicates that variables are strongly connected in the vertical direction. Therefore, we should group variables in y unlike Algorithm 5 which processes x first. The block smoothers in this case are also referred as y-line smoothers. Figure 4.2 shows the errors of y-line GS and x-line GS after 10 iterations. We can see that the error of y-line GS is indeed smooth while that of x-line GS is still oscillatory.

Algorithm 5 Block Gauss-Seidel Iteration

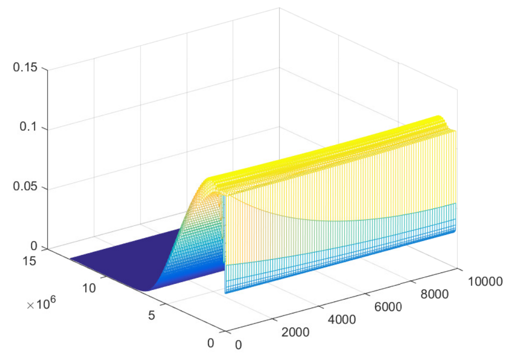
```
input: $\{\mathbf{A}_i\}$, $\{\mathbf{B}_i\}$, $\{\mathbf{C}_i\}$, $N_x$, $N_y$, $\mathbf{b}$
output: $u$
1: solve $\mathbf{A}_1\mathbf{u}_{1:N_x} = \mathbf{b}_{1:N_x} - \mathbf{B}_1\mathbf{u}_{N_x+1:2N_x}$
2: for $k = 2, \ldots, N_y - 1$
3:     $\mathbf{r} = \mathbf{b}_{(k-1)N_x+1:kN_x} - \mathbf{B}_k\mathbf{u}_{kN_x+1:(k+1)N_x} - \mathbf{C}_{k-1}\mathbf{u}_{(k-2)N_x+1:(k-1)N_x}$
4:     solve $\mathbf{A}_k\mathbf{u}_{(k-1)N_x+1:kN_x} = \mathbf{r}$
5: end for
6: solve $\mathbf{A}_{N_y}\mathbf{u}_{(N_y-1)N_x+1:N_xN_y} = \mathbf{b}_{(N_y-1)N_x:N_xN_y} - \mathbf{C}_{N_x-1}\mathbf{u}_{(N_y-2)N_x+1:(N_y-1)N_x}$
```

4.2 Multigrid Setup

With smooth errors, we can then define the coarsening strategy and its corresponding transfer operators. Many algorithms in the literature [16, 67] are developed for nodal-based schemes and Dirichlet boundary conditions. The problem we are interested utilizes cell-centered discretization and the Neumann boundary conditions, which are necessary for the conservation of physical quantities. These differences require changes in the construction of the transfer operators. In addition, unlike



(a) y-line GS.



(b) x-line GS.

Figure 4.2: Error of y-line GS (left) and error of x-line GS (right).

the common choice of the full-weighting scheme, we will see that the cell-centered discretization provides a natural way to interpolate the fine-grid values to the coarse-grid points.

Coarsening. For the generation of the grid hierarchy, we can apply power-two coarsening. Figure 4.3 demonstrates two examples of such strategy. Let N_x denote the number of cells in the x direction and N_y denote that in the y direction. In the first example, N_x and N_y are both divisible by 2. We simply merge every two cells in each direction into a bigger cell. In the second example, N_y is divisible by 2 but N_x is not. In this case, we first merge every two cells in both directions until there is only one cell in x. After that, we proceed with merge every two cells only in the y direction. Note that it will be good to have at least two cells in each direction on the coarsest level. If there is a large difference between N_x and N_y, one may consider semi-coarsening, i.e. coarsening in one direction, when reaching the coarsest level to reduce the grid size further.

Interpolation. There are three cases to consider when constructing the interpolation operator. The first one is to interpolate fine grid points that lie in the interior as shown in Figure 4.4. Since each point is surrounded by four coarse grid points that form a rectangle, we can simply apply the bi-linear interpolation described by formula (3.2.2). The nonstandard cases are the interpolation of boundary points

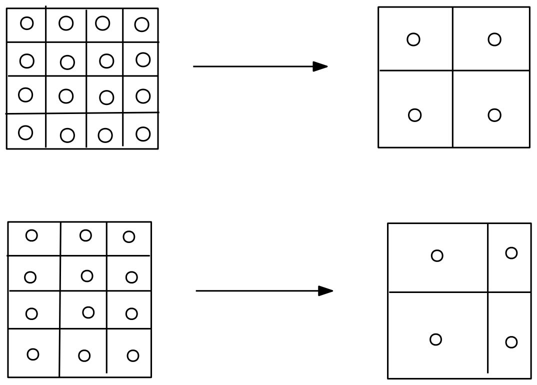


Figure 4.3: 2-D geometric coarsening.

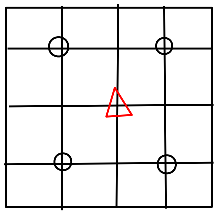
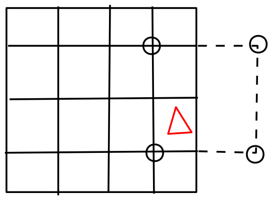


Figure 4.4: Interpolation of interior points.

shown in Figure 4.5. The trick here is use ghost cells. For homogeneous Neumann boundary condition, the ghost point has the same value as its parent. Therefore, the interpolation scheme for case (a) is linear interpolation in y and the scheme for case (b) is piece-wise constant interpolation.

Restriction. The construction of the restriction operator includes two main cases as illustrated in Figure 4.6. For the first case, each interior coarse grid point is surrounded by four fine grid points. Therefore, we can simply use the bi-linear interpolation as opposed to the full-weighting scheme. It is both accurate and cheap in cost. The second case only occurs when N_x is not divisible by 2. Similarly if N_y is not divisible by 2, we need to treat points near the ceiling. Since each coarse grid point is adjacent to only two fine grid points, we can use linear interpolation.


(a) points near edges.

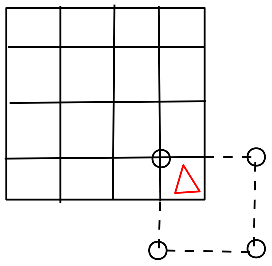
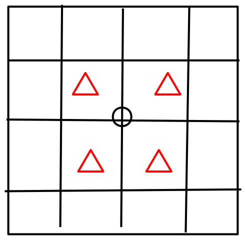

(b) corner points.

Figure 4.5: Interpolation of boundary points.


(a) interior points.

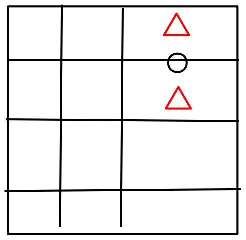

(b) points near edges.

Figure 4.6: Restriction.

4.3 Coarse-grid Solver

Finally we need to choose the appropriate coarse-grid solver. The choice of the coarse-grid solver usually does not introduce any problems. In fact, the multigrid algorithm converges nicely for many problems even with only a few iterations of smoothers applied at the bottom level. The complication here is that the system becomes very ill-conditioned when α is close to zero. When the system is ill-conditioned, neither smoothers nor Krylov subspace methods are sufficient to guarantee the accuracy of the coarse grid solution hence undermining the multigrid convergence. Even if the algorithm does converge quickly in terms of the residuals, the actual error may still be large due to the large condition number. To illustrate the point, let us consider the same equation used in the previous section with $N_x = 3300$, $N_y = 100$, $dx = 2000$, $dy = 100$ and $\alpha = 10^{-14}$. We use $f = \sin\left(\frac{2\pi x}{N_x \cdot dx}\right) \sin\left(\frac{2\pi y}{N_y \cdot dy}\right)$ as the reference solution and generate a matching right hand side vector. We compare a direct solver with 30 iterations of y-line GS in the same multigrid algorithm. The matrix size on the coarsest level is 417. Table 4.1 shows the convergence results. The residuals and errors are measured by the relative 2-norm. Even though

Table 4.1: Comparison of coarse-grid solvers.

Coarse-grid Solver	MG Iter	Residual	Error
LU Factorization	4	4.2e-7	2.0e-4
y-line GS	9	8.9e-7	3.9e-1

y-line GS and LU factorization has similar residual and iteration count, their errors are significantly different. This indicates that direct solvers are much more robust than iterative methods for ill-conditioned systems.

Chapter 5

HyGA:A Hybrid Geometric+Algebraic Multigrid Framework

Geometric multigrid methods are very efficient but the algorithms are often problem-dependent. We have shown in Chapter 4 how to adapt the standard algorithms to solve anisotropic problems on structured grids. In this chapter, we will generalize the algorithm further to deal with unstructured meshes and the finite element discretization. At the same time, we seek to combine the advantages of GMG with AMG.

With these goals in mind, we introduce a new framework of multigrid method for solving elliptic PDEs over hierarchical unstructured meshes [51]. Figure 5.1 shows the schematic of our proposed method, which integrates various components in a systematic fashion. At a high-level, our overall approach may be summarized as follows. Our hierarchical mesh generator starts from a good-quality coarse unstructured mesh that is sufficiently accurate representation and allows accurate high-order reconstruction of the geometry [45]. It iteratively refines the mesh with guaranteed mesh quality (by uniform refinements) and geometric accuracy (by high-order boundary reconstruction). We apply GMG with a multilevel weighted-residual formulation on these hierarchical meshes, and employ the AMG at the coarsest level. We use a semi-iterative method, namely the Chebyshev-Jacobi

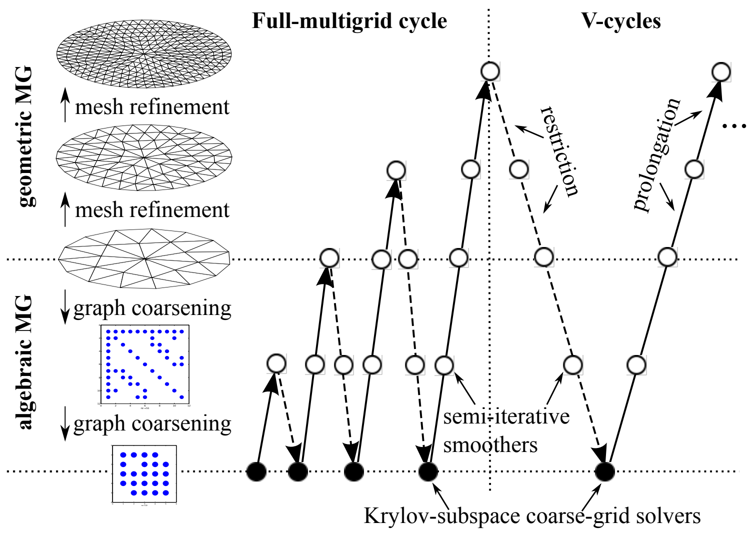


Figure 5.1: Schematic of hybrid geometric+algebraic multigrid method with semi-iterative smoothers.

method, as the smoother at both the GMG and AMG levels, for better parallel efficiency. In this way, the framework effectively couples the rigor, accuracy and runtime-and-memory efficiency of GMG at finer resolutions, with the flexibility and simplicity of AMG at coarser resolutions.

5.1 Weighted Residual Formulation

The weighted residual formulation is a general numerical technique for solving PDEs, of which both the Galerkin finite element, finite volume, and finite difference methods can be viewed as special cases. We describe a general form of multilevel weighted residuals for linear partial differential equations over a hierarchy of basis functions. We will use this form to derive a geometric multigrid over hierarchical meshes in the next section.

5.1.1 A General Weighted Residual Formulation of Linear PDEs

For generality, let us consider an abstract but general form of linear, time-independent partial differential equations

$$\mathscr{P}u(\mathbf{x}) = f(\mathbf{x}), \tag{5.1.1}$$

with Dirichlet or Neumann boundary conditions, where $\mathscr{P}$ is a linear differential operator. In a weighted residual method,[1] given a set of test functions $\Psi(\mathbf{x}) = \{\psi_j(\mathbf{x})\}$, we obtain a linear equation for each ψ_j as

$$\int_\Omega \mathscr{P}u(\mathbf{x})\,\psi_j d\mathbf{x} = \int_\Omega f(\mathbf{x})\,\psi_j d\mathbf{x}, \tag{5.1.2}$$

and then the boundary conditions may be applied by modifying the linear system. In general, a set of basis functions $\Phi(\mathbf{x}) = \{\phi_i(\mathbf{x})\}$ is used to approximate u and f, where Φ and Ψ do not need to be equal.

The above formulation with (5.1.1) and (5.1.2) is general, and it unifies large classes of PDEs and of numerical methods. Examples of (5.1.1) include the Poisson equation $-\Delta u(\mathbf{x}) = f(\mathbf{x})$ and other linear elliptic problems. An example of (5.1.2) is the finite element methods, where the basis and test functions are piecewise Lagrange polynomials. When $\Phi = \Psi$, this reduces to the Galerkin method. Another example is the classical and the generalized finite difference methods [9], where the test functions are the Dirac delta functions at each node of a mesh, and the basis functions are piecewise polynomials (i.e., the basis functions of the polynomial interpolation or approximation in computing the finite-difference formulae). The finite volume methods are also examples, where the test functions are step functions, which have value one over the control volume. This unification will allow us to derive a general formulation of the restriction and prolongation operators for geometric multigrid methods.

For the convenience of notation, suppose Φ and Ψ are both column vectors composed of the basis functions, i.e. $\Phi = [\phi_1, \phi_2, \ldots, \phi_n]^T$ and $\Psi = [\psi_1, \psi_2, \ldots, \psi_n]^T$. Let $\mathbf{u}$ denote the vector of coefficients u_i associated with ϕ_i in the approximation

[1]The term "residual" appears in two different contexts in this paper: the residual of a linear system ($\mathbf{b} - \mathbf{A}\mathbf{u}$) and the residual of a PDE ($f(\mathbf{x}) - \mathscr{P}u(\mathbf{x})$). A "residual equation" is based on the former, and "weighted residual" is based on the latter.

of u, i.e., $u \approx \mathbf{u}^T\Phi = \sum_i u_i\phi_i$, and similarly $f(\mathbf{x}) \approx \mathbf{f}^T\Phi = \sum_i f_i\phi_i$. Then $\mathscr{P}u = \mathscr{P}\left(\mathbf{u}^T\Phi\right) = \mathbf{u}^T\mathscr{P}\Phi$. From (5.1.2), we obtain a linear system

$$\mathbf{A}\mathbf{u} = \mathbf{b}, \tag{5.1.3}$$

where

$$A_{ij} = \int_\Omega \left(\mathscr{P}\phi_j(\mathbf{x})\right)\psi_i(\mathbf{x})d\mathbf{x} \quad \text{and} \quad b_i = \int_\Omega f(\mathbf{x})\psi_i(\mathbf{x})d\mathbf{x}. \tag{5.1.4}$$

For example, for the Poisson equation, we have

$$A_{ij} = \int_\Omega -\Delta\phi_j(\mathbf{x})\,\psi_i(\mathbf{x})d\mathbf{x} = \int_\Omega \nabla\phi_j(\mathbf{x})\cdot\nabla\psi_i(\mathbf{x})d\mathbf{x}. \tag{5.1.5}$$

The matrix $\mathbf{A}$ is the *stiffness matrix*, and $\mathbf{b}$ is the *force vector*. The equation (5.1.3) needs to be updated to incorporate the boundary conditions. The solution of (5.1.3) gives a vector $\mathbf{u}$ such that the residual $\mathbf{f}^T\Phi - \mathscr{P}(\mathbf{u}^T\Phi)$ is orthogonal to the test functions ψ_j. If Φ is composed of Lagrange basis functions, such as in the finite element methods, $\mathbf{u}$ and $\mathbf{f}$ are composed of the nodal values of u and f on a mesh, respectively.

Remark In finite element methods, the integral in the left-hand side of (5.1.2) is often transformed into integrals of a lower-order differential operator multiplied (in an inner-product sense) with $\nabla\psi_j(\mathbf{x})$. For example, in a finite element method where ψ_j vanishes along the boundary, $\int_\Omega -\Delta u(\mathbf{x})\,\psi_j d\mathbf{x} = \int_\Omega \nabla u(\mathbf{x})\cdot\nabla\psi_j d\mathbf{x}$. In the finite volume methods, the integral in the left-hand side of (5.1.2) is transformed into a boundary integral by utilizing Stokes's or Green's theorem. We omit these details as they do not affect the derivations.

5.1.2 Weighted Residuals with Hierarchical Basis

In a multilevel context, we assume a hierarchy of basis functions $\Phi^{(k)}(\mathbf{x})$, and similarly for the test functions $\Psi^{(k)}$. Without loss of generality, let us consider two levels first, and the construction will apply to adjacent levels in a multilevel setting.

Suppose $\Phi^{(1)}$ and $\Phi^{(2)}$ correspond to the basis functions on the fine and coarse levels, respectively, and the function space spanned by $\Phi^{(2)}$ is a subspace of $\Phi^{(1)}$. Let $\mathbf{R}_{\Phi}^{(1,2)}$ denote a *restriction matrix* of the function space such that

$$\Phi^{(2)} = \mathbf{R}_{\Phi}^{(1,2)}\Phi^{(1)}, \tag{5.1.6}$$

where $\mathbf{R}_{\Phi}^{(1,2)} \in \mathbb{R}^{n_2 \times n_1}$. Similarly, let $\Psi^{(2)} = \mathbf{R}_{\Psi}^{(1,2)}\Psi^{(1)}$ with another restriction matrix $\mathbf{R}_{\Psi}^{(1,2)}$.

At the kth level, let $\mathbf{A}^{(k)}$ denote the matrix $\mathbf{A}$ in (5.1.3). A key question is the relationships between $\mathbf{A}^{(1)}$ and $\mathbf{A}^{(2)}$. To derive this, let us re-write $\mathbf{A}^{(k)}$ in the form of an integral of an outer product of $\Psi^{(k)}$ and $\mathscr{P}\Phi^{(k)}$, i.e.,

$$\mathbf{A}^{(k)} = \int_{\Omega} \Psi^{(k)} \left(\mathscr{P}\Phi^{(k)}\right)^T d\mathbf{x}. \tag{5.1.7}$$

Substituting $\Phi^{(2)} = \mathbf{R}_{\Phi}^{(1,2)}\Phi^{(1)}$ and $\Psi^{(2)} = \mathbf{R}_{\Psi}^{(1,2)}\Psi^{(1)}$ into it, we then obtain

$$\begin{aligned}
\mathbf{A}^{(2)} &= \int_{\Omega} \left(\mathbf{R}_{\Psi}^{(1,2)}\Psi^{(1)}\right)\left(\mathscr{P}\mathbf{R}_{\Phi}^{(1,2)}\Phi^{(1)}\right)^T d\mathbf{x} \\
&= \mathbf{R}_{\Psi}^{(1,2)}\left(\int_{\Omega} \Psi^{(1)}\left(\mathscr{P}\Phi^{(1)}\right)^T d\mathbf{x}\right)\left(\mathbf{R}_{\Phi}^{(1,2)}\right)^T \\
&= \mathbf{R}_{\Psi}^{(1,2)}\mathbf{A}^{(1)}\left(\mathbf{R}_{\Phi}^{(1,2)}\right)^T
\end{aligned} \tag{5.1.8}$$

From (5.1.8), we conclude that the restriction matrix $\mathbf{R}$ and the prolongation matrix $\mathbf{P}$ in a two-level multigrid method for weighted residual methods should be

$$\mathbf{R} = \mathbf{R}_{\Psi}^{(1,2)} \text{ and } \mathbf{P} = \left(\mathbf{R}_{\Phi}^{(1,2)}\right)^T. \tag{5.1.9}$$

In particular with nested meshes, for Galerkin methods, $\mathbf{P} = \mathbf{R}^T = \left(\mathbf{R}_{\Phi}^{(1,2)}\right)^T$ is an interpolation matrix, which is a well-known result for Poisson equations [62] and [16, Chapter 10]. In addition, it is desirable for these interpolations to have the same order of accuracy as the numerical discretizations. For the classical and generalized finite difference methods, $\mathbf{P}$ is also an interpolation matrix, but $\mathbf{R}$ is not equal to $\mathbf{P}^T$ and instead is an $n_2 \times n_1$ permutation matrix (an injection operator) or scaled $\mathbf{P}^T$, because the ψ_j are Dirac delta functions. For finite volume methods, $\mathbf{R}$ is not equal to $\mathbf{P}^T$ either. Instead, it corresponds to a constant interpolation from a cell to its subcells.

We prove our result in (5.1.9) as follows. Let $u^{(1)} = \left(\mathbf{u}^{(1)}\right)^T \Phi^{(1)}$ denote the approximation of u with basis $\Phi^{(1)}$, and let $\mathbf{b}^{(1)}$ denote the right-hand vector in (5.1.3). The residual of linear system (5.1.3) with basis $\Phi^{(1)}$ is $\mathbf{r}^{(1)} = \mathbf{b}^{(1)} - \mathbf{A}^{(1)}\mathbf{u}^{(1)}$. Let $\mathbf{r}^{(2)} = \mathbf{R}_\Psi^{(1,2)}\mathbf{r}^{(1)}$. The residual equation with $\Phi^{(2)}$ is then

$$\mathbf{A}^{(2)}\mathbf{s}^{(2)} = \mathbf{R}_\Psi^{(1,2)}\mathbf{r}^{(1)}, \tag{5.1.10}$$

where $\mathbf{s}^{(2)}$ is the correction vector with $\Phi^{(2)}$. Substituting (5.1.8) into it, we then obtain

$$\mathbf{R}_\Psi^{(1,2)}\mathbf{A}^{(1)}\left(\mathbf{R}_\Phi^{(1,2)}\right)^T\mathbf{s}^{(2)} = \mathbf{R}_\Psi^{(1,2)}\mathbf{A}^{(1)}\mathbf{s}^{(1)} = \mathbf{R}_\Psi^{(1,2)}\mathbf{r}^{(1)}, \tag{5.1.11}$$

where $\mathbf{s}^{(1)} = \mathbf{P}\mathbf{s}^{(2)} = \left(\mathbf{R}_\Phi^{(1,2)}\right)^T\mathbf{s}^{(2)}$ is the prolongated correction vector with $\Phi^{(1)}$. In the functional form, (5.1.11) can be rewritten as

$$\int_\Omega \Psi^{(2)}\left(\Phi^{(1)}\right)^T\left(\mathbf{u}^{(1)} + \mathbf{s}^{(1)}\right)d\mathbf{x} = \int_\Omega \Psi^{(2)} f(\mathbf{x})\,d\mathbf{x}, \tag{5.1.12}$$

i.e., $\mathbf{s}^{(1)}$ gives a correction to $\mathbf{u}^{(1)}$ so that the updated residual of the PDE is orthogonal to all the test functions in $\Psi^{(2)}$.

From (5.1.8) and (5.1.9), we see that the matrix $\mathbf{A}^{(2)}$ is identical to the matrix $\mathbf{R}\mathbf{A}^{(1)}\mathbf{P}$ in a multilevel weighted-residual formulation with hierarchical basis functions for any linear partial differential equation in the form of (5.1.1). This allows us to discretize the PDE directly to obtain $\mathbf{A}^{(2)}$.

5.2 Hybrid Multigrid Methods

We now present our hybrid multigrid method utilizing our results on multilevel weighted residuals. Our hybrid multigrid method, called *HyGA*, combines a geometric multigrid solver with a few levels of hierarchical meshes, and an algebraic

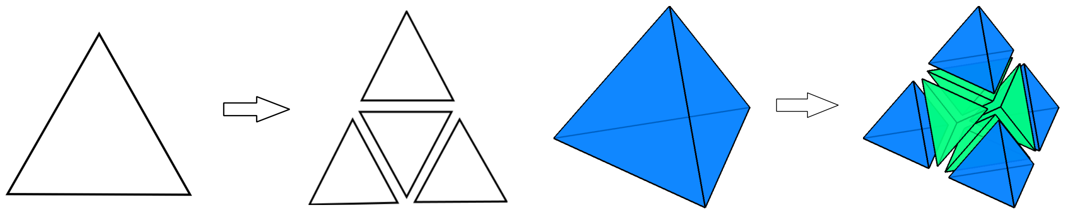


Figure 5.2: Illustration of refinement of triangle (left) and tetrahedron (right).

multigrid on the coarsest level. We will focus on finite element methods with linear simplicial elements (in particular, triangles and tetrahedra in 2-D and 3-D) as well as generalized finite differences.

5.2.1 Generation of Hierarchical Meshes

We first describe the construction of hierarchical meshes, which are needed for our geometric multigrid based on multilevel weighted residuals.

Guaranteed-Quality Mesh Refinement.

We construct hierarchical meshes through iterative mesh refinement, instead of mesh coarsening. In two dimensions, we start from a good-quality coarse triangular mesh. To generate a finer mesh, we subdivide each triangle into four equal sub-triangles that are similar to the original triangle, as illustrated in Figure 5.2(left). The element quality (in terms of angles) is preserved under mesh refinement. To generate an ℓ-level hierarchical mesh, we repeat the refinement $(\ell - 1)$ times. In three dimensions, we start from a good-quality coarse tetrahedral mesh, and subdivide each tetrahedron into eight sub-tetrahedra, as illustrated in Figure 5.2(right). There are three different choices in the subdivisions. Any of these subdivisions will produce eight sub-tetrahedra of the same volume, so it does not introduce very poor-quality elements. Among these sub-tetrahedra, the four sub-tetrahedra incident on the original corner vertices are similar to the original tetrahedra. The four interior sub-tetrahedra may vary in shapes. We choose the subdivision that minimizes the edge lengths, as it tends to minimize the aspect ratio, defined as the ratio of the sum of squared edge lengths and the two-thirds root of the volume [46].

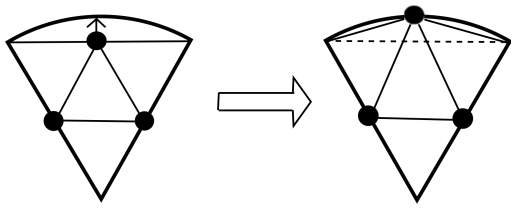


Figure 5.3: Illustration of the treatment of curved boundary by projecting inserted mid-edge points.

Treatment of Curved Boundaries.

One of the main reasons why unstructured meshes are useful in practice is its flexibility to deal with complex geometries, especially those with curved boundaries. When generating hierarchical meshes, we need to respect the curved boundaries. We achieve this by projecting the newly inserted mid-edge points onto the curved geometry, as illustrated in Figure 5.3. The projection can be done either analytically if the geometry is known, or through a high-order reconstruction of the geometry [45]. Note that if the mesh is too coarse at a concave region, some mesh smoothing may be needed to avoid inverted elements (i.e., elements with negative Jacobian). We omit this issue in this paper.

5.2.2 Prolongation and Restriction Operators

After we obtain the hierarchical meshes, the basis and test functions are determined on each level. In Galerkin finite elements, the basis functions $\Phi^{(k)}$ and test functions $\Psi^{(k)}$ are the same, and they are piecewise Lagrange polynomials. Assume the meshes are exactly nested, then the function space on a coarser mesh is strictly a subspace of that on a finer mesh. Therefore, there is an exact restriction matrix $\mathbf{R}_{\Phi}^{(k,k+1)}$ of the functional space between levels k and $k+1$. As we have shown in Section 5.1.2, for Galerkin finite elements the optimal prolongation operator is $\mathbf{P}^{(k)} = \left(\mathbf{R}_{\Phi}^{(k,k+1)}\right)^T$, and the optimal restriction operator is $\mathbf{R}^{(k)} = \mathbf{R}_{\Phi}^{(k,k+1)}$. With these definitions, the matrix $\mathbf{A}^{(k+1)}$ from discretizing the PDE over the $(k+1)$st grid is equivalent to $\mathbf{R}^{(k)}\mathbf{A}^{(k)}\mathbf{P}^{(k)}$.

In general, computing $\mathbf{R}_{\Phi}^{(k,k+1)}$ may be a daunting task. However for Lagrange basis functions over hierarchical meshes, $\mathbf{R}_{\Phi}^{(k,k+1)}$ is precisely the transpose of the interpolation matrix $\mathbf{I}^{(k+1,k)}$ of the nodal values from the $(k+1)$st level mesh to the kth level mesh. This result may not be obvious, because as noted in [67], such nodal interpolations may require proper scaling to produce the correct prolongation and restriction operators. In the following, we show that $\mathbf{R}_{\Phi}^{(k,k+1)} = \left(\mathbf{I}^{(k+1,k)}\right)^T$.

Without loss of generality, let $k = 1$, and let $\tilde{u}^{(2)} = \left(\tilde{\mathbf{u}}^{(2)}\right)^T \Phi^{(2)}$ denote the approximation of u with basis $\Phi^{(2)}$. Since $\Phi^{(2)} = \mathbf{R}_{\Phi}^{(1,2)}\Phi^{(1)}$, we have

$$\tilde{u}^{(2)} = \left(\tilde{\mathbf{u}}^{(2)}\right)^T \mathbf{R}_{\Phi}^{(1,2)}\Phi^{(1)} = \left(\tilde{\mathbf{u}}^{(1)}\right)^T \Phi^{(1)}, \tag{5.2.1}$$

where $\tilde{\mathbf{u}}^{(1)} = \left(\mathbf{R}_{\Phi}^{(1,2)}\right)^T \tilde{\mathbf{u}}^{(2)}$. At the same time, because $\Phi^{(k)}$ are Lagrange basis functions, we have

$$\tilde{u}^{(2)} = \left(\mathbf{I}^{(2,1)}\tilde{\mathbf{u}}^{(2)}\right)^T \Phi^{(1)}. \tag{5.2.2}$$

Thus, $\mathbf{I}^{(2,1)}\tilde{\mathbf{u}}^{(2)} = \left(\mathbf{R}_{\Phi}^{(1,2)}\right)^T \tilde{\mathbf{u}}^{(2)}$ for any $\tilde{\mathbf{u}}$, so $\mathbf{R}_{\Phi}^{(1,2)} - \left(\mathbf{I}^{(2,1)}\right)^T = \mathbf{0}$.

In summary, for hierarchical Lagrange basis functions, $\mathbf{P}^{(k)} = \mathbf{I}^{(k+1,k)}$, i.e., interpolation matrix of nodal values from $(k+1)$st level to the kth level. In addition for the Galerkin finite element methods, $\mathbf{R}^{(k)} = \left(\mathbf{I}^{(k+1,k)}\right)^T$. One subtle point is that after we project the points onto curved boundaries, the element is no longer nested, so the Lagrange basis functions are no longer strictly hierarchical. When constructing the prolongation and restriction matrices, we treat the elements as nested (in other words, creating the prolongation and restriction operators as if boundary projection did not occur). This is because that for linear elements, whose geometric errors are second order when approximating curved boundaries, this projection introduces second-order corrections to the positions between each pair of meshes. Therefore, the additional errors introduced by omitting the curved boundaries in the prolongation and restriction operators are in the same order as the truncation errors, so it would not affect the convergence rate.

5.2.3 Hybrid Geometric+Algebraic Multigrid

Our preceding formulation based on hierarchical basis functions is efficient and relatively easy to implement. However, it has one limitation: it requires a hierarchical mesh. In practice, it is reasonable to assume a small number of levels (such as two or three levels) in a hierarchical mesh, but we cannot expect to have too many levels. This can limit the scalability of GMG with hierarchical meshes alone.

To overcome this issue, we propose to use a hybrid geometric+algebraic multgird method, or *HyGA*. In this approach, we use geometric multigrid based on multi-level weighted residuals for the finer levels with the hierarchical mesh, and use an algebraic multigrid (AMG) method, in particular the classical AMG, for the coarser levels. Details of the classical AMG algorithm can be found in Section 3.3.

5.3 Chebyshev Smoothers

Our preceding section focused on the prolongation and restriction operators. In multigrid methods, another critical component is the smoother. The standard practice is to use stationary iterative methods, such as Gauss-Seidel iterations, as smoothers, because of their good smoothing properties. However, these stationary iterative methods tend to converge relatively slowly. The semi-iterative methods utilize polynomial acceleration to improve the convergence of the stationary iterative methods [35, 72], and thus from theoretical point of view they are appealing alternatives for smoothers. In addition, the semi-iterative methods, especially those based on polynomial accelerations of Jacobi iterations, are appealing from a practical point of view, because they are inherently parallel. In contrast, its Gauss-Seidel counterpart poses various difficulties especially for unstructured meshes [1].

In some previous studies such as [1], some variances of semi-iterative methods or polynomial accelerations have been proposed as smoothers. In this work, we propose the use of the *Chebyshev-Jacobi method*, or *CJ* for short, as a smoother. This method possesses a number of interesting properties as a standalone solver and as a smoother, for linear systems with both symmetric and asymmetric matrices under some reasonable assumptions.

5.3.1 Standalone Chebyshev-Jacobi Method

Our discussion on semi-iterative methods mostly follow [40]. For simplicity, let us first consider the case of symmetric matrices, for which all the eigenvalues are real. The semi-iterative method introduces a polynomial acceleration for a stationary iterative scheme

$$\mathbf{x}^{(n+1)} = \mathbf{G}\mathbf{x}^{(n)} + \mathbf{k}. \tag{5.3.1}$$

In particular, we are interested in accelerating the Jacobi iteration, where $\mathbf{G} = \mathbf{I} - \mathbf{D}^{-1}\mathbf{A}$ and $\mathbf{D} = \operatorname{diag}(\mathbf{A})$. To improve convergence of $\mathbf{x}^{(n)}$, we consider a new sequence $\left\{\mathbf{u}^{(n)}\right\}$ defined by the following linear combination

$$\mathbf{u}^{(n)} = \sum_{i=0}^{n} \alpha_{n,i}\mathbf{x}^{(n)}, \tag{5.3.2}$$

where the $\alpha_{n,i}$ are some constant coefficients. Under the assumption that $\mathbf{A}$ is symmetric, all the eigenvalues of $\mathbf{G}$ are real. Let $\{\lambda_i\}$ denote these eigenvalues, where λ_1 and λ_n are the largest and smallest eigenvalues of $\mathbf{G}$, respectively. In the CJ, the acceleration scheme has the following recurrence relation

$$\mathbf{u}^{(n+1)} = \rho_{n+1}\left(\gamma(\mathbf{G}\mathbf{u}^{(n)} + \mathbf{k}) + (1-\gamma)\,\mathbf{u}^{(n)}\right) + (1-\rho_{n+1})\,\mathbf{u}^{(n-1)}, \tag{5.3.3}$$

where the parameters are given by

$$\gamma = \frac{2}{2-\lambda_1-\lambda_n},\ \sigma = \frac{\gamma}{2}(\lambda_1-\lambda_n),\ \text{and}\ \rho_{n+1} = \begin{cases} 1 & n=0 \\ \left(1-\frac{1}{2}\sigma^2\right)^{-1} & n=1 \\ \left(1-\frac{1}{4}\sigma^2\rho_n\right)^{-1} & n\geqslant 2 \end{cases}. \tag{5.3.4}$$

The above formulas follow from the three-term recurrence of Chebyshev polynomials. For completeness, we summarize the derivation as follows. Let $\mathbf{u}^*$ denote the true solution of the linear system $\mathbf{A}\mathbf{x} = \mathbf{b}$, and $\mathbf{e}^{(n)} = \mathbf{u}^{(n)} - \mathbf{u}^*$ be the error at the nth iteration. It is easy to show that

$$\mathbf{e}^{(n)} = \left(\sum_{i=0}^{n} \alpha_{n,i}\mathbf{G}^i\right)\mathbf{e}^{(0)} = P_n(\mathbf{G})\,\mathbf{e}^{(0)}, \tag{5.3.5}$$

where

$$P_n(\mathbf{G}) = \alpha_{n,0}\mathbf{I} + \alpha_{n,1}\mathbf{G} + \cdots + \alpha_{n,n}\mathbf{G}^n \tag{5.3.6}$$

is an nth order matrix polynomial, and $P_n(1) = 1$. The convergence of the CJ depends on the spectral radius $\rho(P_n(\mathbf{G}))$. Note that

$$\rho(P_n(\mathbf{G})) \leqslant \max_{\lambda_n \leqslant x \leqslant \lambda_1} |P_n(x)|. \tag{5.3.7}$$

Thus we want to find a polynomial that its maximum absolute value over the spectrum of $\mathbf{G}$ is minimum. It can be shown that the unique optimal polynomial is given by [40]

$$P_n(x) = T_n\left(\frac{2x - \lambda_1 - \lambda_n}{\lambda_1 - \lambda_n}\right) \Big/ T_n\left(\frac{2 - \lambda_1 - \lambda_n}{\lambda_1 - \lambda_n}\right), \tag{5.3.8}$$

where $T_n(x)$ is the nth order Chebyshev polynomial. Using the three-term recurrence of Chebyshev polynomials T_n, we then obtain (5.3.3).

Convergence of Chebyshev-Jacobi Method for Symmetric Matrices.

An important property of the CJ is that it is guaranteed to converge if all the eigenvalues of $\mathbf{G}$ are real and less than 1 [40], which is satisfied if the matrix $\mathbf{A}$ is symmetric positive definite. Hence, the CJ significantly robustifies the Jacobi iterations, as it is guaranteed to converge when $\mathbf{A}$ is symmetric and positive definite, whereas the Jacobi iterations may diverge in this setting.

In addition, the CJ also significantly improves the convergence rate of Jacobi iterations. More precisely, it can be shown [40] that

$$\rho(P_n(\mathbf{G})) = \frac{2r^{n/2}}{(1+r^n)}, \quad \text{where } r = \frac{1-\sqrt{1-\sigma^2}}{1+\sqrt{1-\sigma^2}}, \tag{5.3.9}$$

and the asymptotic rate of convergence for CJ is given by

$$R_\infty(P_n(\mathbf{G})) = -\frac{1}{2}\log r. \tag{5.3.10}$$

In comparison, the asymptotic rate of convergence for Jacobi iterations is given by

$$R_\infty\left(\mathbf{G}_\gamma\right) = -\log\sigma, \tag{5.3.11}$$

where $\mathbf{G}_\gamma = (1-\gamma)\,\mathbf{I} + \gamma\mathbf{G}$ is the iteration matrix of weighted Jacobi iterations with a weight γ defined in (5.3.4). Combining (5.3.10) and (5.3.11), it can be shown that

$$R_\infty\left(P_n\left(\mathbf{G}\right)\right) \sim \sqrt{2R_\infty\left(\mathbf{G}_\gamma\right)} \text{ as } \sigma \to 1^-. \tag{5.3.12}$$

In words, the CJ in general can result in an order of magnitude improvement in the rate of convergence compared to Jacobi iterations.

The robustness and efficiency of the CJ make it an appealing method for solving linear systems, and they also motivate us to consider it as a smoother. However, a subtle issue of the CJ is that it requires good estimations of extreme eigenvalues for optimal convergence. Let λ_n be the smallest eigenvalue and $\lambda_n^{'}$ be an estimate of λ_n. Fortunately, as shown in [40], the estimation of the smallest eigenvalue λ_n does not affect the convergence much, as long as $\lambda_n^{'} \leq \lambda_n$. More specifically, if the following condition is satisfied

$$0 \leqslant \lambda_n - \lambda_n^{'} \leqslant 0.1\max\{|\lambda_n|, 1\}, \tag{5.3.13}$$

then a non-optimal $\lambda_n^{'}$ will cause only up to 4% increase in the number of iterations. Therefore, we do not need a very accurate estimate for λ_n. However, if $\lambda_n^{'} > \lambda_n$, divergence may occur.

On the other hand, when used as a standalone solver, the convergence of CJ is sensitive to the estimation of the largest eigenvalue λ_1. In addition, if we underestimate λ_1, the expected number of iterations would increase much more than if we overestimate λ_1 by an equal amount, so one needs to estimate an upper bound of λ_1. Fortunately, this situation will be much less an issue when CJ is used as a smoother, as we discuss in Section 5.3.2.

Generalization to Asymmetric Matrices.

In PDE discretizations, the matrix **A** is often asymmetric, due to either complex boundary conditions or the asymmetry in the stencils of the numerical discretizations (such as high-order finite differences or generalized finite differences). In this case, the matrix **A** and in turn the iteration matrix **G** may have complex eigenvalues, and the situation is somewhat more complicated. However, the matrix **A** is typically "nearly symmetric," because the symmetry is only due to the boundary effect or numerical discretization errors.

Assume the eigenvalues of **G** are bounded by the following ellipse

$$\frac{(x-d)^2}{a^2}+\frac{y^2}{b^2}=1, \tag{5.3.14}$$

where the constants a and b are real, and

$$a+d<1. \tag{5.3.15}$$

If $a \geq b$, $c=\sqrt{a^2-b^2}$ is also real, then the unique polynomial that minimizes the maximum spectrum over the ellipse is given by [20]

$$P_n(z)=T_n\left(\frac{z-d}{c}\right)\bigg/T_n\left(\frac{1-d}{c}\right), \tag{5.3.16}$$

where $T_n(z)$ is the Chebyshev polynomial. Using the three-term recurrence of T_n, we obtain the acceleration scheme same as (5.3.3), except that the parameters γ and σ are now given by

$$\gamma=\frac{1}{1-d}=\frac{2}{2-(d+a)-(d-a)}, \text{ and } \sigma=c\gamma=\gamma\sqrt{a^2-b^2}. \tag{5.3.17}$$

These parameters are consistent with (5.3.4) when the eigenvalues are all real, where $\lambda_1=d+a$, $\lambda_n=d-a$, and $c=a=(\lambda_1-\lambda_n)/2$. In addition, if the imaginary parts of all the eigenvalues of **G** are small (i.e., $b \ll a$), then $c \approx a$, and thus

$$\gamma \approx \frac{2}{2-\text{Re}(\lambda_1)-\text{Re}(\lambda_n)}, \text{ and } \sigma \approx \frac{\gamma}{2}(\text{Re}(\lambda_1)-\text{Re}(\lambda_n)). \tag{5.3.18}$$

Therefore, if the matrix is nearly symmetric (more precisely if the imaginary parts of the eigenvalues are small), we can approximate the parameters γ and σ as in the symmetric case from the extreme eigenvalues of $\mathbf{D}^{-1}\left(\mathbf{A}+\mathbf{A}^T\right)/2$.

5.3.2 Chebyshev-Jacobi Method as Smoother

We now analyze the applicability of the CJ as a smoother. The key issue is the estimation of the eigenvalues λ_1 and λ_n and in turn γ and σ. As in the case for standalone CJ, it is not necessary to obtain an accurate estimation of the smallest eigenvalue λ_n of the iteration matrix $\mathbf{G}$. In practice, we can simply apply a few Lanczos iterations to get an estimation of the largest eigenvalue of $\mathbf{D}^{-1}\mathbf{A}$, say t, and then obtain $\lambda_n \approx 1-t$. If $\mathbf{A}$ is asymmetric, instead of $\mathbf{D}^{-1}\mathbf{A}$, it suffices to estimate the largest eigenvalue of $\mathbf{D}^{-1}\left(\mathbf{A}+\mathbf{A}^T\right)/2$, whose eigenvalues are all real.

For the estimation of λ_1, while it is important for the standalone CJ, it does not need to be very accurate when CJ is used as a smoother. This is because for solving elliptic PDEs, the smoothers only need to damp high-frequency modes. Assuming $\mathbf{A}$ is symmetric, these modes typically correspond to the larger eigenvalues of $\mathbf{D}^{-1}\mathbf{A}$ and the smaller eigenvalues of $\mathbf{I}-\mathbf{D}^{-1}\mathbf{A}$. Therefore, an inaccurate estimation of λ_1 would not undermine the smoothing effect of CJ, as long as estimations satisfy $\lambda_n' < \lambda_1' < 1$, where λ_1' denotes the estimation of λ_1. In fact, we observed that an accurate λ_1 does not produce the best smoothing effect, although it gives optimal convergence of the standalone CJ. In our experiments, we have found that it works well to use $\lambda_1' \approx 2/3$ for 2-D problems and $\lambda_1' \approx 0.9$ for 3-D problems.

5.4 Numerical Experiments

In this section, we present some numerical experimentations using HyGA with CJ, and also assess the effectiveness of our method against some other alternatives. For our benchmark problem, we use some sparse linear systems from finite elements or generalized finite differences for Poisson equations with Dirichlet boundary conditions. Figure 5.4 shows the test geometries for our problem, both of which have irregular and curved boundaries and hence require unstructured meshes.

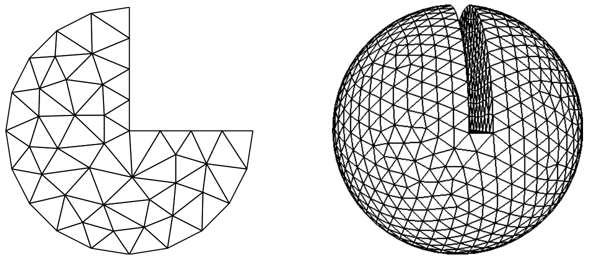


Figure 5.4: Coarsest initial meshes for 2-D (left) and 3-D tests (right) .

We solve the linear system using the following four strategies and compare their performances:

AMG: classical AMG.

GMG: GMG with multilevel weighted-residual formulation.

HyGA(2)**:** GMG on first two levels and classical AMG on coarser levels.

HyGA(3)**:** GMG on first three levels and classical AMG on coarser levels.

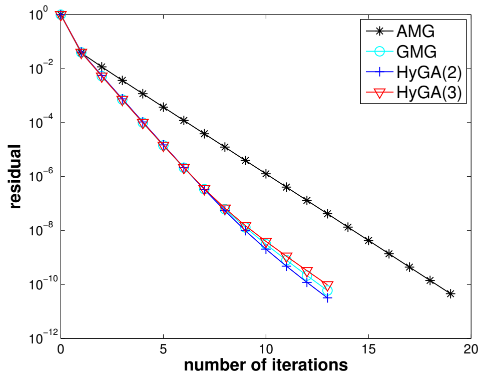
For all the tests, we use one cycle of full multigrid, followed by V-cycles. On the coarsest level, conjugate gradient or GMRES is used to obtain sufficient accuracy. In terms of smoothers, we use 2 Chebyshev-Jacobi iterations for 2-D problems and 4 for 3-D problems in GMG and HyGA. In AMG, we use Gauss-Seidel method with the same number of iterations as it is more effective. The tolerance is 10^{-10} measured by the relative 2-norm of residuals.

5.4.1 Convergence Results for 2-D Finite Elements

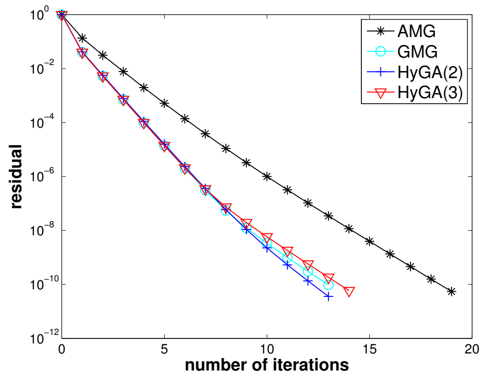
Our first test case is the 2-D finite element discretization for the Poisson equation

$$\Delta u(\mathbf{x}) = f(\mathbf{x}), \text{ where } f(\mathbf{x}) = -2\pi^2 \left(\sin(\pi x) + \sin(\pi y)\right) \tag{5.4.1}$$

with homogeneous boundary conditions on three quarters of a unit disk depicted in Figure 5.4(left). Starting from a 2-D initial mesh generated using Triangle [63], we generated three meshes with different resolutions, by refining the initial mesh



(a) 35,986 unknowns with 5 levels.



(b) 144,674 unknowns with 6 levels.

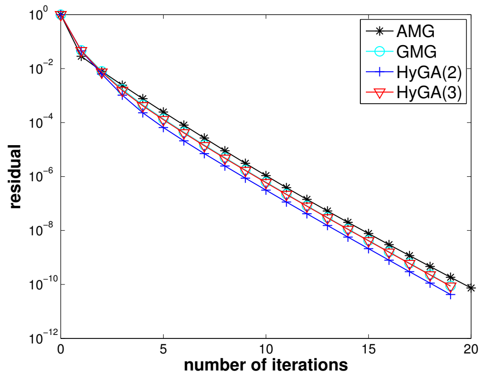
Figure 5.5: Relative residual versus numbers of iterations for 2-D test cases.

in Figure 5.4(left) for three, four, and five times, to obtain hierarchical meshes with four, five, and six levels, respectively. After each refinement, we projected the newly inserted boundary points onto the curved boundary.

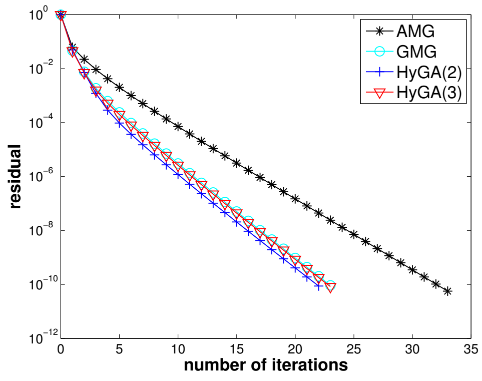
Figure 5.5 shows the convergence for the five- and six-level meshes with different strategies. For AMG we choose strength of connection θ (c.f. Section 5.2.3) to be 0.25 as it seems to give the best performance. From these results, it can be seen that GMG converges a bit faster than AMG. For HyGA with only two or three levels of GMG, its convergence rate was comparable to GMG. These results indicate that the finer levels are critical to the overall convergence of multigrid methods. Therefore, it is advantageous to use GMG to achieve the best accuracy at finer levels. In addition, as we observe in Section 5.4.3, GMG at the finer levels tend to produce coarser operators and hence better efficiency. On coarser levels, the more slowly converging but more flexible AMG suffices to ensure good overall performance of HyGA. This alleviates the complications of further mesh coarsening of a pure GMG and reduces the computational cost of a pure AMG while maintaining good convergence.

5.4.2 Convergence Results for 3-D Finite Elements

Our second test case is the 3-D finite-element discretization for the Poisson equation



(a) 291,684 unknowns with 4 levels.



(b) 2,484,807 unknowns with 5 levels.

Figure 5.6: Relative residual versus numbers of iterations for 3-D test cases.

$$\Delta u(\mathbf{x}) = f(\mathbf{x}), \text{ where } f(\mathbf{x}) = -3\pi^2 \left(\sin(\pi x) + \sin(\pi y) + \sin(\pi z)\right) \qquad (5.4.2)$$

with homogeneous boundary conditions on a slotted sphere depicted in Figure 5.4. We generated a tetrahedral mesh from the surface mesh using TetGen [64], and then build two meshes with different resolutions by refining the initial mesh in Figure 5.4 for three and four times, to obtain hierarchical meshes with four and five levels, respectively. We use similar strategies and settings as for the 2-D tests. However for AMG, the parameter θ is chosen to be 0.65 and the value decreases in lower levels to achieve good performance. Figure 5.6 shows the convergence of different strategies, which are qualitatively similar to the 2-D results. It can be seen that GMG converged faster than AMG. At the same time, HyGA performed nearly identically as GMG, while being much more flexible. Moreover, the performance of HyGA was about the same on the 2-D and 3-D mesh, while the performance of AMG was more sensitive to the choices of parameters.

5.4.3 Comparison of Computational Costs

In terms of computational times, Table 5.1 compares the runtimes of different strategies for FEMs on a Mac Pro with two 2.4 GHz quad-core Intel Xeon processor and 24 GB of memory, running Mac OS X 10.7.5 with gcc 4.2. For all strategies, we show only the solve time since the setup times were less significant. As a reference, we also show the running times of MATLAB's built-in preconditioned conjugate gradient (PCG) with incomplete Cholesky factorization (ichol) as the preconditioner. Similar to the convergence result, the performance of hybrid schemes are comparable to GMG with only two or three geometric levels. In addition, GMG and HyGA are faster than AMG even when their convergence rates are close. The reason is the higher complexity of the coarse-grid operators of the AMG, which is evident from Figure 5.7, where we compare the operator complexities of strategies on several levels for the 3-D 5-level mesh. From the charts we observe that GMG produces much smaller and more sparse matrices than AMG does. The operator complexity of HyGA is also moderate, as GMG is used on the finest levels. Since the computational time is dominated by smoothing, which is affected by matrix sparsity, the performances of GMG and HyGA overall are much better.

Table 5.1: Timing results (in seconds) for AMG, GMG and HyGA. For references, times for PCG (with incomplete Cholesky preconditioner) are shown.

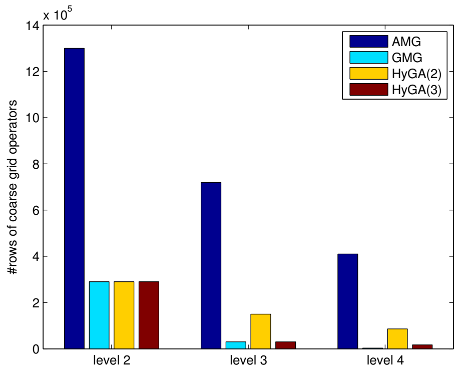
dim	#rows in the system	AMG	GMG	HyGA(2)	HyGA(3)	IC-PCG
2-D	8,906	0.12	0.04	0.08	0.04	0.21
	35,986	0.37	0.14	0.25	0.17	1.27
	144,674	1.43	0.61	1.14	0.79	11.1
3-D	291,684	30	5.26	6.67	5.22	9.32
	2,484,807	397	58.3	85.8	61.2	186

5.4.4 Assessment of Chebyshev-Jacobi Smoother

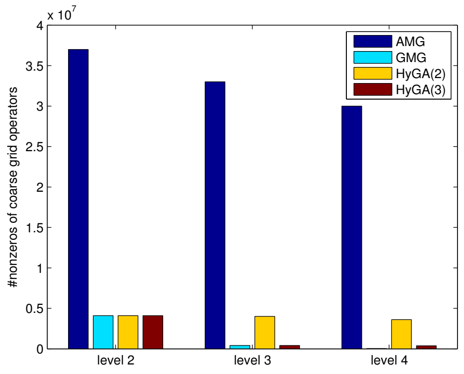
We now assess the effectiveness of the CJ as a smoother in HyGA. We first investigate the influence of parameters λ_n' and λ_1', and then compare the performances of the CJ versus lexicographic Gauss-Seidel iterations.

Influences of Parameters.

To evaluate the influence of parameters, we consider the 2-D Poisson equation with a 5-level mesh and 3-D Poisson equation with a 4-level mesh as described in the previous subsection. We computed a reference value for λ_n using MATLAB's eigs function with a relative tolerance of 10^{-5}. In these test cases, $\lambda_n < 0$. We compare



(a) number of rows in the coarse grid operators.



(b) number of nonzeros in the coarse grid operators.

Figure 5.7: Number of rows and nonzeros in the coarse grid operators of different strategies on levels 2, 3, and 4 for the 3-D 5 level mesh.

Table 5.2: Sensitivity of λ_n' for 2-D and 3-D test cases.

λ_n'	number of multigrid iterations	
	2-D	3-D
λ_n	14	19
$1.2\lambda_n$	14	21
-2	16	23
$0.8\lambda_n$	47	diverged

the numbers of iterations required by the multigrid method for the cases of λ_n' equal to the optimal value λ_n, a loose bound $1.2\lambda_n$, and the very loose bound -2. In addition, we also test the case where $\lambda_n' = 0.8\lambda_n$, which violates the requirement $\lambda_n' \leq \lambda_n$ and hence the CJ may diverge. In all these cases, we set λ_1' to $2/3$ for 2-D and 0.9 for 3-D.

Table 5.2 shows the numbers of multigrid iterations of HyGA with these parameters. It can be seen that the convergence of CJ is insensitive to λ_n as long as $\lambda_n' \leq \lambda_n$. Even a naive loose estimate of $\lambda_n \approx -2$ did not affect the convergence very much. However, when $\lambda_n' > \lambda_n$, the convergence deteriorated significantly, and it even diverged in our 3-D test.

Table 5.3: Comparison of Chebyshev-Jacobi and Gauss-Seidel as smoothers. All times are in seconds.

Tests	Chebyshev-Jacobi		Gauss-Seidel	
	iterations	time	iterations	time
2-D 5-level	14	0.17	9	0.11
2-D 6-level	14	0.79	10	0.50
3-D 4-level	19	5.24	21	6.08
3-D 5-level	23	61.3	24	61.6

Comparison with Gauss-Seidel Iterations.

To assess the effectiveness of the CJ method as smoother, we compare the convergence of HyGA(3,$\ell-3$) with CJ versus Gauss-Seidel iterations as the smoother. In both cases, we use the conjugate gradient method as the coarse grid solver. Table 5.3 shows the numbers of iterations as well as the run times of the methods. In both 2-D and 3-D, Gauss-Seidel works rather well as a smoother. The convergence of CJ is a bit worse than Gauss-Seidel in the 2-D tests, but it is better than Gauss-Seidel in 3-D when degree-4 polynomials are used. This result is remarkable, because the CJ method is much easier to parallelize and is more scalable than Gauss-Seidel, so it can be expected to out-perform Gauss-Seidel iterations on parallel computers. Similar behaviors of other semi-iterative methods have been reported in [1].

Chapter 6

OPINS: An Orthogonally Projected Implicit Null-space Method

In this chapter, we consider a different type of linear system, the saddle-point system. Our previous discussions about the multigrid methods can not be directly applied here due to the unique structure of the coefficient matrix. We will instead focus on the design of an efficient Krylov subspace method and leave the development of multigrid preconditioners as future work.

The saddle-point system has the following form:

$$\begin{bmatrix} \mathbf{A} & \mathbf{B}^T \\ \mathbf{B} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} = \begin{bmatrix} \mathbf{f} \\ \mathbf{g} \end{bmatrix}, \tag{6.0.1}$$

where $\mathbf{A} \in \mathbb{R}^{n \times n}$ is symmetric, $\mathbf{B} \in \mathbb{R}^{m \times n}$, $\mathbf{x}, \mathbf{f} \in \mathbb{R}^n$, and $\mathbf{y}, \mathbf{g} \in \mathbb{R}^m$. The coefficient matrix, denoted by $\mathbf{K}$, can be symmetric and indefinite. It may be nonsingular or singular, and often very ill-conditioned even when it is nonsingular.

This type of problems arises in many scientific applications. For example, it can be derived from solving the following constrained optimization problem,

$$\min_{\mathbf{x}} \frac{1}{2} \mathbf{x}^T \mathbf{A} \mathbf{x} - \mathbf{f}^T \mathbf{x} \quad \text{subject to} \quad \mathbf{B}\mathbf{x} = \mathbf{g}, \tag{6.0.2}$$

using the method of Lagrange multipliers, where $\mathbf{A}$ is the Hessian of the quadratic objective function, and $\mathbf{B}\mathbf{x} = \mathbf{g}$ defines a constraint hyperplane for the minimization. In this case, $\mathbf{A}$ is typically symmetric, $\mathbf{x}$ contains the *optimization variables* or *solution variables*, and $\mathbf{y}$ contains the *Lagrange multipliers* or *constraint variables*. The optimality conditions are referred as Karush-Kuhn-Tucker conditions, and the system (6.0.1) is often called the *KKT system* [55], with $\mathbf{K}$ being the *KKT matrix*. In PDE discretization, the Lagrange multipliers are often used to enforce nodal conditions, such as the sliding boundary conditions and the continuity constraints at hanging nodes [31].

Solving the saddle-point problems is particularly challenging. One may attempt to solve it using a general-purpose solver, such as a direct solver or an iterative solver [14, 23, 68], or the range-space method [73]. For large and sparse saddle-point systems, a more powerful method is the null-space method [10], which solves for $\mathbf{x}$ first and then $\mathbf{y}$, with a Krylov subspace method as its core solver. The disadvantage of the null-space method is that computing an orthonormal basis can be very expensive. Another class of methods, which we refer to as *implicit null-space methods* [61, 36, 37], are equivalent to the null-space method with an orthonormal basis of $\operatorname{null}(\mathbf{B})$, but they do not require computing the basis explicitly. However, with rounding errors, these methods may suffer from numerical instability. The instability may be mitigated by iterative refinements [36, 37], but with increased computational costs.

We propose an implicit orthogonal projection method, called *OPINS*, which is a more stable variant of implicit null-space methods. Specifically, we compute an orthonormal basis for $\operatorname{range}\left(\mathbf{B}^T\right)$, which can be constructed efficiently when $m \ll n$ using a stable algorithm such as *QR* factorization with column pivoting (QRCP). Let $\mathbf{U}$ denote the matrix composed of such an orthonormal basis. Instead of using $\mathbf{Z}^T\mathbf{A}\mathbf{Z}$ in the null-space method, we use the orthogonal projector $\Pi_{\mathbf{U}}^{\perp} \equiv \mathbf{I} - \mathbf{U}\mathbf{U}^T$ to construct a singular but compatible linear system, with a symmetric coefficient matrix $\Pi_{\mathbf{U}}^{\perp}\mathbf{A}\Pi_{\mathbf{U}}^{\perp}$. We solve this system using a solver for singular systems, such as MINRES [56, 19] and SYMMLQ [56] for symmetric systems. We also propose preconditioners for saddle-point systems, based on the work in [37]. The resulting OPINS method is highly efficient when $m \ll n$, and it is stable, robust, and easy to implement using existing Krylov-subspace method for singular systems, without the issue of stagnation suffered by other implicit null-space methods.

Besides being more stable, another advantage of OPINS is that it can be applied to singular saddle-point systems, which arise in various applications. In the literature, most methods assume the saddle-point system is nonsingular. When this assumption is violated, for example when there are redundant constraints or insufficient constraints, these methods may force the user to add artificial boundary conditions or soft constraints to make the system nonsingular, which unfortunately may undermine the physical accuracy of the solution. OPINS not only solves the singular system but also finds the minimum norm solution in terms of $\mathbf{x}$.

6.1 Background and Related Works

We briefly review some important concepts and related methods for solving saddle-point problems. There is a large body of literature on saddle-point problems; see [10] for a comprehensive survey up to early 2000s and the references in [37] for more recent works. We focus our discussions on the null-space methods, since they are the most relevant to our proposed method. For completeness, we will also briefly discuss some other methods for saddle-point systems and singular linear systems.

6.1.1 Explicit Null-Space Methods

One of the most powerful methods for solving saddle-point systems is the null-space method. Geometrically, for a nonsingular saddle-point problem arising from constrained minimization, this method finds a critical point of an objective function within a constraint hyperplane $\mathbf{Bx} = \mathbf{g}$. Algorithm 6 outlines the algorithm, where $\mathbf{x}_p \in \mathbb{R}^n$ denotes a particular solution within the constraint hyperplane. Let q denote the rank of the constraint matrix $\mathbf{B}$, and $q \le m$. The column vectors of the $\mathbf{Z} \in \mathbb{R}^{n\times(n-q)}$ form a basis of null$(\mathbf{B})$, and $\mathbf{BZ} = \mathbf{0}$. Step 3 finds a component $\mathbf{x}_n$ in null$(\mathbf{B})$, i.e. a vector tangent to the constraint hyperplane, so that $\mathbf{x}_n + \mathbf{x}_p$ is at a critical point. After determining $\mathbf{x}$, the final step finds the Lagrange multipliers $\mathbf{y}$ in range$(\mathbf{B})$.

Algorithm 6 Null-Space Method

```
input: $\mathbf{A}$, $\mathbf{B}$, $\mathbf{f}$, $\mathbf{g}$, tolerance for the iterative solver (if used)
output: $\mathbf{x}$, $\mathbf{y}$ (optional)
1: solve $\mathbf{B}\mathbf{x}_p = \mathbf{g}$
2: compute $\mathbf{Z}$, composed of basis vectors of null$(\mathbf{B})$
3: solve $\mathbf{Z}^T\mathbf{A}\mathbf{Z}\mathbf{v} = \mathbf{Z}^T(\mathbf{f} - \mathbf{A}\mathbf{x}_p)$
4: $\mathbf{x}_n \leftarrow \mathbf{Z}\mathbf{v}$ and $\mathbf{x} \leftarrow \mathbf{x}_p + \mathbf{x}_n$
5: $\mathbf{y} \leftarrow \mathbf{B}^{+T}(\mathbf{f} - \mathbf{A}\mathbf{x})$
```

In the algorithm, step 3 is the most critical, which solves the equation

$$\mathbf{Z}^T\mathbf{A}\mathbf{Z}\mathbf{v} = \mathbf{Z}^T(\mathbf{f} - \mathbf{A}\mathbf{x}_p), \tag{6.1.1}$$

within null$(\mathbf{B})$. We refer to (6.1.1) as the *null-space equation*, and denote its coefficient matrix by $\hat{\mathbf{N}}$. This matrix is $(n-q)\times(n-q)$, which is smaller than the original matrix $\mathbf{K}$. When $\mathbf{A}$ is symmetric and positive semidefinite and null$(\mathbf{A}) \cap$ null$(\mathbf{B}) = \{\mathbf{0}\}$, $\hat{\mathbf{N}}$ is SPD [10], and (6.1.1) can be solved efficiently using preconditioned conjugate gradient (CG). However, if $\hat{\mathbf{N}}$ is symmetric but indefinite or is nonsymmetric, then an alternative iterative solver, such MINRES, SYMMLQ [56], or GMRES [60], can be used; see textbooks such as [59] for details of these iterative solvers.

In the traditional null-space method, the matrix $\mathbf{Z}$ is constructed explicitly. We refer to such an approach as the *explicit null-space method*. In exact arithmetic, it is not necessary for $\mathbf{Z}$ to be orthonormal. If $\mathbf{B}$ has full rank, then there exists an $n\times n$ permutation matrix $\mathbf{P}$ such as $\mathbf{B}\mathbf{P} = [\mathbf{B}_1 \mid \mathbf{B}_2]$, where $\mathbf{B}_1$ is an $m\times m$ nonsingular matrix, and $\mathbf{B}_2$ is $m\times(n-m)$. Then, one could simply choose $\mathbf{Z}$ to be [33]

$$\mathbf{Z} = \mathbf{P}\begin{bmatrix} -\mathbf{B}_1^{-1}\mathbf{B}_2 \\ \mathbf{I} \end{bmatrix}, \tag{6.1.2}$$

where $\mathbf{I}$ is the $(n-m)\times(n-m)$ identity matrix. However, if the columns of $\mathbf{Z}$ are too far from being orthonormal, $\mathbf{Z}^T\mathbf{A}\mathbf{Z}$ may be ill-conditioned, which in turn can cause slow convergence for iterative solvers or large errors in the resulting solution. Therefore, it is desirable for $\mathbf{Z}$ to be orthonormal or nearly orthonormal, so that $\mathbf{Z}^T\mathbf{A}\mathbf{Z}$ approximately preserves the condition number of $\mathbf{A}$. If the dimension of null$(\mathbf{B})$ is high, i.e., when $m \ll n$, $\mathbf{Z}$ is typically quite dense. Determining an orthonormal $\mathbf{Z}$ requires the full QR factorization of $\mathbf{B}^T$, which takes $\mathscr{O}(n^3)$ operations. Therefore, explicit null-space methods are impractical for large-scale applications.

6.1.2 Implicit Null-Space Methods

For saddle-point systems where $m \ll n$, it is desirable to avoid constructing an orthonormal basis of null$(\mathbf{B})$ explicitly. One such approach is to use a Krylov subspace method with a *constraint preconditioner* [36, 47], which has the form

$$\mathbf{M} = \begin{bmatrix} \mathbf{G} & \mathbf{B}^T \\ \mathbf{B} & \mathbf{0} \end{bmatrix}, \tag{6.1.3}$$

where $\mathbf{G}$ is an approximation of $\mathbf{A}$. This preconditioner is indefinite, and it was shown in [61] that it provides optimal bounds for the maximum eigenvalues among similar indefinite preconditioners. Since the preconditioner is indefinite, it is not obvious whether we can use it as a preconditioner for methods such as CG or MINRES, which typically require symmetric positive-definite preconditioners. As shown in [36, 37], one can apply the preconditioner to the modified saddle-point system

$$\begin{bmatrix} \mathbf{A} & \mathbf{B}^T \\ \mathbf{B} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{x}_n \\ \mathbf{y} \end{bmatrix} = \begin{bmatrix} \mathbf{f} - \mathbf{A}\mathbf{x}_p \\ \mathbf{0} \end{bmatrix}, \tag{6.1.4}$$

with preconditioned CG and MINRES. The vector $\mathbf{x}_n$ in (6.1.4) is the same as that from step 4 in Algorithm 6, but the vector $\mathbf{y}$ computed from (6.1.4) may be inaccurate, so one may need to solve for $\mathbf{y}$ separately once $\mathbf{x}$ is obtained. In exact arithmetic, the Krylov subspace method with this constraint preconditioner is equivalent to the *projected Krylov method* in [37]. Because of this equivalence, we will use the names *projected Krylov methods* and *Krylov subspace methods with constraint preconditioning* interchangeably in this paper, although these methods somewhat differ in their implementation details.

The projected Krylov method is closely related to the preconditioned null-space method on the null-space equation (6.1.1) with an orthonormal $\mathbf{Z}$. The stability of the method requires the computed $\mathbf{x}_n$ to be exactly in null$(\mathbf{B})$ at each step. However, the rounding errors can quickly introduce a nonnegligible component in range$(\mathbf{B}^T)$, which can cause the method to break down [36, 37]. To mitigate this issue, the constraint preconditioner must be applied "exactly," by solving the preconditioned system with a direct method followed by one or more steps of iterative refinement per iteration [36, 37]. The iterative refinement introduces extra cost, and there is no guarantee that it would recover orthogonality between $\mathbf{x}_n$ and range$(\mathbf{B})$ to machine precision, so the projected Krylov method may still stagnate. Besides its potential instability, the projected Krylov methods typically assume the KKT system is nonsingular [37]. It is desirable to develop a more stable version of the implicit null-space method that can also be applied to singular KKT systems.

6.1.3 Other Methods for Saddle-Point Systems

Besides the null-space methods, another class of methods for saddle-point problems is the range-space method [73]. It first obtains $\mathbf{y}$ by solving the system

$$\left(\mathbf{B}\mathbf{A}^{-1}\mathbf{B}^T\right)\mathbf{y} = \mathbf{B}\mathbf{A}^{-1}\mathbf{f} - \mathbf{g}, \tag{6.1.5}$$

where the coefficient matrix is the Schur complement, and then computes $\mathbf{x}$ by solving

$$\mathbf{A}\mathbf{x} = \mathbf{f} - \mathbf{B}^T\mathbf{y}. \tag{6.1.6}$$

The range-space method can be attractive if a factorization of $\mathbf{A}$ is available. However, computing the Schur complement is expensive if $\mathbf{A}$ is large and sparse, and the method is not directly applicable if $\mathbf{A}$ is singular.

The null-space and range-space methods both leverage the special structures of the saddle-point systems. In some cases, one may attempt to solve the whole system (6.0.1) directly using a factorization-based method, such as LDL^T decomposition for symmetric systems [68]. These methods are prohibitively expensive for large-scale problems. In addition, since the solution variables $\mathbf{x}$ and constraint variables $\mathbf{y}$ have different physical meanings, the entries in $\mathbf{A}$ and $\mathbf{B}$ may have very different scales. As a result, $\mathbf{K}$ may be arbitrarily ill-conditioned, and these methods may break down or produce inaccurate solutions due to poor scaling. Another class of method is iterative solvers with preconditioners [11]. The constraint preconditioner is a special case, which is equivalent to an implicit null-space method, with some optimality properties among similar indefinite preconditioners [4, 61]. Some other preconditioners include the block diagonal [23], block triangular [14], and multigrid [2]. These methods do not distinguish between $\mathbf{x}$ and $\mathbf{y}$, so they may have more difficulties when $\mathbf{K}$ is singular or ill-conditioned.

6.1.4 Methods for Singular Saddle-Point Systems

The methods we discussed above typically assume nonsingular saddle-point systems. For general singular saddle-point problems, one may resort to truncated SVD [34] or rank-revealing QR, which are computationally expensive. One may also apply an iterative solver for singular systems, such as MINRES and SYMMLQ [56] for compatible symmetric systems, and MINRES-QLP [19] for incompatible symmetric systems. Without preconditioners, these methods can find the minimum-norm solution of compatible singular systems when they are applicable. However, they do not distinguish between $\mathbf{x}$ and $\mathbf{y}$ in the solver. As a result, these methods minimize $\|\mathbf{x}\|^2 + \|\mathbf{y}\|^2$, which may substantially differ from the minimum-norm solution of $\mathbf{x}$, especially when $\mathbf{K}$ is ill-conditioned. OPINS will overcome this issue by leveraging the iterative solvers in the implicit null-space method in a stable and efficient fashion, and in turn offer an effective method for solving singular saddle-point systems.

6.2 Orthogonally Projected Null-space Method

In this section, we introduce the *Orthogonally Projected Implicit Null-Space Method*, or *OPINS*, for solving saddle-point systems (6.0.1). Similar to the other implicit null-space methods, OPINS does not require the explicit construction of the basis of null($\mathbf{B}$), and it is particularly effective when $m \ll n$. However, unlike previous null-space methods, OPINS enforces orthogonality explicitly and hence enjoys better stability. It is also applicable to singular saddle-point systems. In the following subsections, we present the OPINS method, its derivation, and the analysis of its cost and stability.

6.2.1 Algorithm Description

A core idea of OPINS is to use the orthogonal projector onto null$(\mathbf{B})$, constructed from an orthonormal basis of range$(\mathbf{B}^T)$. Let $\mathbf{U}$ denote the matrix composed of an orthonormal basis of range$(\mathbf{B}^T)$, which can be computed using truncated SVD or QR with column pivoting (QRCP) [17, 34]. Since $\mathbf{U}$ is orthonormal, $\Pi_{\mathbf{U}} \equiv \mathbf{U}\mathbf{U}^T$ is the unique orthogonal projector onto range$(\mathbf{B}^T)$, and

$$\Pi_{\mathbf{U}}^{\perp} \equiv \mathbf{I} - \Pi_{\mathbf{U}} = \mathbf{I} - \mathbf{U}\mathbf{U}^T \tag{6.2.1}$$

is its complementary orthogonal projector onto null$(\mathbf{B})$. Note that $\Pi_{\mathbf{U}}^{\perp} = \Pi_{\mathbf{Z}}$, where $\mathbf{Z}$ is composed of an orthonormal basis of null$(\mathbf{B})$. Let $q = \text{rank}(\mathbf{B})$. Using this projector, OPINS replaces the $(n-q)\times(n-q)$ null-space equation ((6.1.1)) in the null space method with the $n \times n$ singular system

$$\Pi_{\mathbf{U}}^{\perp}\mathbf{A}\Pi_{\mathbf{U}}^{\perp}\mathbf{w} = \Pi_{\mathbf{U}}^{\perp}\left(\mathbf{f} - \mathbf{A}\mathbf{x}_p\right), \tag{6.2.2}$$

and then solves it using a solver for singular systems. We refer to the above equation as the *projected null-space (PNS) equation*, and denote its coefficient matrix as $\mathbf{N}$.

Algorithm 7 outlines the complete OPINS algorithm, which applies to nonsingular or compatible singular systems. The first two steps find an orthonormal basis of $\mathbf{B}^T$ using QRCP, where $\mathbf{P}$ is an $m \times m$ permutation matrix, so that the diagonal values of $\mathbf{R}$ are sorted in descending order. $\mathbf{Q} \in \mathbb{R}^{n\times m}$ is orthonormal, and $\mathbf{R} \in \mathbb{R}^{m\times m}$ is upper triangular. For stability, QRCP should be computed based on Householder QR factorization [34]. If $\mathbf{B}$ is rank deficient, its rank can be estimated from the magnitude of the diagonal entries in $\mathbf{R}$, or more robustly using a condition-number estimator [34]. The first q columns of $\mathbf{Q}$ form an orthonormal basis of range$\left(\mathbf{B}^T\right)$. With QRCP, both $\mathbf{x}_p$ in step 3 and $\mathbf{y}$ in step 6 can also be solved efficiently. When $\mathbf{B}$ is rank deficient, so is $\mathbf{R}$. We use $\mathbf{R}_{1:q,1:q}^{-T}$ and $\mathbf{R}_{1:q,1:q}^{-1}$ to denote the forward and back substitutions on $\mathbf{R}_{1:q,1:q}$.

The key step of the algorithm is the solution of the PNS equation (6.2.2) in step 4, which is singular. As we will show in Section 6.2.2, (6.2.2) is compatible under the assumption that (6.0.1) is compatible in terms of $\mathbf{f}$, i.e., $\mathbf{f} \in \text{range}(\mathbf{A}) + \text{range}(\mathbf{B}^T)$. Therefore, we can solve it using a Krylov subspace method for compatible singu-

Algorithm 7 OPINS: Orthogonally Projected Implicit Null-Space Method

```
input: $\mathbf{A}$, $\mathbf{B}$, $\mathbf{f}$, $\mathbf{g}$, tolerances for rank estimation and iterative solver
output: $\mathbf{x}$, $\mathbf{y}$ (optional)
1: $\mathbf{B}^T\mathbf{P} = \mathbf{QR}$ {QR factorization with column pivoting}
2: $\mathbf{U} \leftarrow \mathbf{Q}_{1:q}$, where $q$ is rank$(\mathbf{B})$ estimated from QRCP
3: $\mathbf{x}_p \leftarrow \mathbf{B}^+\mathbf{g} = \mathbf{U}\mathbf{R}_{1:q,1:q}^{-T}\left(\mathbf{P}^T\mathbf{g}\right)_{1:q}$
4: solve $\Pi_{\mathbf{U}}^{\perp}\mathbf{A}\Pi_{\mathbf{U}}^{\perp}\mathbf{w} = \Pi_{\mathbf{U}}^{\perp}(\mathbf{f} - \mathbf{A}\mathbf{x}_p)$ using iterative singular solver
5: $\mathbf{x} \leftarrow \mathbf{x}_p + \mathbf{x}_n$, where $\mathbf{x}_n = \Pi_{\mathbf{U}}^{\perp}\mathbf{w}$
6: $\mathbf{y} \leftarrow \mathbf{B}^{+T}(\mathbf{f} - \mathbf{A}\mathbf{x}) = \mathbf{P}_{:,1:q}\mathbf{R}_{1:q,1:q}^{-1}\mathbf{U}^T(\mathbf{f} - \mathbf{A}\mathbf{x})$
```

lar systems, as we discuss in more detail in Subsection 6.2.2. A key operation in these methods is the multiplication of the coefficient matrix with a vector. For the multiplication with $\Pi_{\mathbf{U}}^{\perp}$ with any vector $\mathbf{v} \in \mathbb{R}^n$,

$$\Pi_{\mathbf{U}}^{\perp}\mathbf{v} = \mathbf{v} - \mathbf{U}\left(\mathbf{U}^T\mathbf{v}\right), \qquad (6.2.3)$$

which can be computed stably and efficiently. Note that $\mathbf{U}$ is not stored explicitly either, but as a collection of Householder reflection vectors in QRCP.

6.2.2 Detailed Derivation of OPINS

The derivation of OPINS is similar to that of the null-space method. For completeness, we will start with the derivation of the explicit null-space method, and then extend it to derive OPINS. We will also discuss the solution techniques of the PNS equation.

Null-Space Method for Singular and Nonsingular Systems

Let $\mathbf{x}_* = \mathbf{x}_n + \mathbf{x}_p$, where $\mathbf{x}_p \in \text{range}(\mathbf{B}^T)$ is a particular solution and $\mathbf{x}_n = \mathbf{Z}\mathbf{v}$ is a component in null$(\mathbf{B})$. Our goal is to find $\mathbf{x}_n$. Substituting the expression into the equation yields

$$\mathbf{A}(\mathbf{Z}\mathbf{v} + \mathbf{x}_p) + \mathbf{B}^T = \mathbf{f}. \qquad (6.2.4)$$

We can multiply both sides with $\mathbf{Z}^T$ and obtain

$$\mathbf{Z}^T\mathbf{A}(\mathbf{x}_n+\mathbf{x}_p)=\mathbf{Z}^T\mathbf{f}, \tag{6.2.5}$$

or equivalently,

$$\mathbf{Z}^T\mathbf{A}\mathbf{Z}\mathbf{v}=\mathbf{Z}^T\left(\mathbf{f}-\mathbf{A}\mathbf{x_p}\right). \tag{6.2.6}$$

This is the null-space equation (6.1.1). If $\mathbf{f}\in \text{range}(\mathbf{A})+\text{range}(\mathbf{B}^T)$, then the null-space equation (6.1.1) is compatible.

In the above derivation, $\mathbf{Z}$ does not need to be orthonormal. However, if $\mathbf{Z}$ is far from being orthonormal, the system (6.1.1) may have a large condition number, which may affect the convergence of the iterative solvers and the accuracy of the numerical solution.

OPINS for Singular and Nonsingular Systems

To derive OPINS, now assume $\mathbf{Z}$ is orthonormal. We further multiply $\mathbf{Z}$ to both sides of the null-space equation and then obtain

$$\mathbf{Z}\mathbf{Z}^T\mathbf{A}\mathbf{Z}\mathbf{v}=\mathbf{Z}\mathbf{Z}^T\left(\mathbf{f}-\mathbf{A}\mathbf{x}_p\right). \tag{6.2.7}$$

In addition, we rewrite $\mathbf{x}_n=\mathbf{Z}\mathbf{v}$ as the orthogonal projection of a vector $\mathbf{w}\in\mathbb{R}^n$ onto $\text{null}(\mathbf{B})$, i.e.,

$$\mathbf{Z}\mathbf{v}=\mathbf{Z}\mathbf{Z}^T\mathbf{w}. \tag{6.2.8}$$

Substituting it into (6.2.7), we have

$$\mathbf{Z}\mathbf{Z}^T\mathbf{A}\mathbf{Z}\mathbf{Z}^T\mathbf{w}=\mathbf{Z}\mathbf{Z}^T\left(\mathbf{f}-\mathbf{A}\mathbf{x}_p\right), \tag{6.2.9}$$

which is equivalent to (6.2.2) in step 4 of Algorithm 7.

The above transformation may seem counter-intuitive, as we intentionally constructed a singular system (6.2.2), which is larger than the null-space equation (6.1.1). However, (6.2.2) has three key properties: First, it uses an orthogonal projector to ensure that $\mathbf{x}_n$ is exactly in $\text{null}(\mathbf{B})$, and hence it overcomes the instability associated with the projected Krylov methods. Second, since $\Pi_{\mathbf{Z}}=\mathbf{Z}\mathbf{Z}^T=\mathbf{I}-\mathbf{U}\mathbf{U}^T=\Pi_{\mathbf{U}}^{\perp}$, we can compute the projection by finding $\mathbf{U}$, which is much more efficient than finding $\mathbf{Z}$ when $q \leq m \ll n$. Third, since (6.2.2) is always singular, and QRCP in step 1 supports rank-deficient $\mathbf{B}$, we can apply OPINS to a saddle-point system regardless of whether $\mathbf{A}$, $\mathbf{B}$, or $\mathbf{Z}^T\mathbf{AZ}$ is singular.

Solution of Orthogonally Projected Null-Space Equation

Since the PNS system (6.1.1) has an infinite number of solutions, a natural question is which solution of the system suffices in producing the minimum-norm solution in terms of $\mathbf{x}$. In the following, we will address the question first for systems with a nonsingular null-space equation, followed by more general singular systems.

Theorem 2. *Given a saddle-point system (6.0.1) where the null-space equation (6.1.1) is nonsingular and the solution to (6.0.1) is unique in terms of* $\mathbf{x}$*, then the solution of (6.2.2) recovers this unique* $\mathbf{x}$.

Proof. Consider the alternative form of the PNS equation in (6.2.9). The system is compatible, so we can always find a solution $\mathbf{w} \in \mathbb{R}^n$ for the equality to hold. Since $\mathbf{Z}^T\mathbf{Z} = \mathbf{I}$, left-multiplying $\mathbf{Z}^T$ on both sides of (6.2.9), we obtain

$$\mathbf{Z}^T\mathbf{AZZ}^T\mathbf{w} = \mathbf{Z}^T\left(\mathbf{f} - \mathbf{A}\mathbf{x}_p\right). \tag{6.2.10}$$

Since system (6.1.1) is nonsingular, $\mathbf{Z}^T\mathbf{w}$ recovers the unique solution for $\mathbf{v}$, and in turn recovers the unique $\mathbf{x}_n$ and $\mathbf{x}$. □

Note that nonsingular (6.1.1) includes the cases where the saddle-point system (6.0.1) is nonsingular. However, it does not necessarily imply that (6.0.1) is nonsingular, because $\mathbf{B}$ may be rank-deficient. An implication of Theorem 2 is that we have the flexibility of solving (6.1.1) with any solver even if (6.0.1) is singular.

For a general singular saddle-point system, the situation is more complicated. Assume (6.0.1) is compatible in terms of $\mathbf{f}$, the following theorem indicates that OPINS finds the minimum-norm solution of $\mathbf{x}$.

Theorem 3. *Given a saddle-point system (6.0.1) compatible in terms of* $\mathbf{f}$*, i.e.,* $\mathbf{f} \in \mathit{range}(\mathbf{A}) + \mathit{range}(\mathbf{B}^T)$*; if* $\mathbf{x}$ *is the minimum-norm solution of the PNS equation (6.2.2), then* $\|\mathbf{x}\|$ *is minimized among all the solutions* $\mathbf{x} = \mathbf{x}_n + \mathbf{x}_p$ *that satisfy the constraint* $\mathbf{Bx} = \mathbf{g}$.

Proof. Note that $\mathbf{x} = \mathbf{x}_p + \mathbf{x}_n$, where $\mathbf{x}_p \in \text{range}(\mathbf{B}^T)$ and $\mathbf{x}_n \in \text{null}(\mathbf{B})$, so $\|\mathbf{x}\|^2 = \|\mathbf{x}_p\|^2 + \|\mathbf{x}_n\|^2$. In step 3 of QRCP, $\mathbf{x}_p$ is the minimum-norm solution in $\text{range}(\mathbf{B}^T)$. Therefore, we only need to show that $\|\mathbf{x}_n\|$ is minimized in $\text{null}(\mathbf{B})$, where $\mathbf{x}_n = \mathbf{Z}\mathbf{v}$. This is satisfied if $\|\mathbf{v}\|$ is minimized among the exact solutions to the null-space equation (6.1.1), i.e.,

$$\mathbf{Z}^T\mathbf{A}\mathbf{Z}\mathbf{v} = \mathbf{Z}^T(\mathbf{f} - \mathbf{A}\mathbf{x}_p). \tag{6.2.11}$$

Since $\mathbf{Z}$ is orthonormal, $\|\mathbf{x}_n\| = \|\mathbf{v}\|$. In OPINS, if $\mathbf{w}$ is an exact solution to (6.2.2),

$$\mathbf{Z}\mathbf{Z}^T\mathbf{A}\mathbf{Z}\mathbf{Z}^T\mathbf{w} = \mathbf{Z}\mathbf{Z}^T(\mathbf{f} - \mathbf{A}\mathbf{x}_p). \tag{6.2.12}$$

Since $\mathbf{Z}^T\mathbf{Z} = \mathbf{I}$, by left-multiplying $\mathbf{Z}^T$ on both sides, we have

$$\mathbf{Z}^T\mathbf{A}\mathbf{Z}\mathbf{Z}^T\mathbf{w} = \mathbf{Z}^T(\mathbf{f} - \mathbf{A}\mathbf{x}_p), \tag{6.2.13}$$

so $\mathbf{v} = \mathbf{Z}^T\mathbf{w}$ is an exact solution of the null-space equation (6.1.1). Note that $\|\mathbf{v}\| = \|\mathbf{Z}^T\mathbf{w}\| \leq \|\mathbf{Z}^T\|\|\mathbf{w}\| = \|\mathbf{w}\|$, and it is an equality if $\mathbf{w} \in \text{range}(\mathbf{Z}) = \text{null}(\mathbf{B})$, i.e., $\mathbf{w} = \mathbf{Z}\mathbf{Z}^T\mathbf{w} = \mathbf{x}_n$. Therefore, the minimum-norm solution of $\mathbf{w}$ in (6.2.2) minimizes $\|\mathbf{x}_n\|$. □

Note that Theorem 3 is more general than Theorem 2, as it includes Theorem 3 as a special case. Finally, regarding the $\mathbf{y}$ component in (6.1.1), the values of $\mathbf{y}$ are not important for many applications. However, if desired, we can obtain $\mathbf{y}$ by solving $\mathbf{B}^T\mathbf{y} = (\mathbf{f} - \mathbf{A}\mathbf{x})$ using QRCP. In general, $\|\mathbf{y}\|$ may not be minimized if $\mathbf{B}$ is rank deficient, but the norm is typically small.

6.2.3 Efficiency of OPINS

We now analyze the computational cost of OPINS. There are two components that are relatively more expensive. The first is the QRCP in step 1, which takes $\mathcal{O}(\frac{4}{3}m^2n)$ operations when using Householder transformation. When $m \ll n$, this operation is far more efficient than finding an orthonormal basis of $\text{null}(\mathbf{B})$. After obtaining the QR factorization, steps 3, 5, and 6 all take $\mathcal{O}(mn)$ operations. In addition, if $\mathbf{B}$ is sparse, $\mathbf{Q}$ and $\mathbf{R}$ are in general also sparse, leading to even more cost savings. The other one is the solution of the singular system (6.2.2). When $\mathbf{A}$ is large and sparse, it is not advisable to use truncated SVD or rank-revealing QR factorization for this system. Instead, we apply a Krylov-subspace method for singular systems. Within each iteration of these methods, the dominating operation is matrix-vector multiplications, which cost $\mathscr{O}(N+nq)$, where N denote the total number of nonzeros in $\mathbf{A}$. The convergence of these methods depend on the nonzero eigenvalues [39]. The following proposition correlates the eigenvalues of the coefficient matrices $\hat{\mathbf{N}}$ and $\mathbf{N}$ in (6.1.1) and (6.2.2), respectively.

Proposition 4. *Given a saddle-point system (6.0.1), the PNS matrix in (6.2.2) has the same nonzero eigenvalues as the null-space system (6.1.1) with an orthonormal* $\mathbf{Z}$.

Proof. Let $\hat{\mathbf{N}} = \mathbf{Z}^T\mathbf{A}\mathbf{Z}$ and $\mathbf{N} = \mathbf{Z}\left(\mathbf{Z}^T\mathbf{A}\mathbf{Z}\right)\mathbf{Z}^T = \mathbf{Z}\hat{\mathbf{N}}\mathbf{Z}^T$. If $\hat{\lambda}$ is a nonzero eigenvalue of $\hat{\mathbf{N}}$, and $\hat{\mathbf{x}}$ is a corresponding eigenvalue, then

$$\mathbf{N}(\mathbf{Z}\hat{\mathbf{x}}) = \left(\mathbf{Z}\hat{\mathbf{N}}\mathbf{Z}^T\right)(\mathbf{Z}\hat{\mathbf{x}}) = \mathbf{Z}\hat{\mathbf{N}}\hat{\mathbf{x}} = \hat{\lambda}\mathbf{Z}\hat{\mathbf{x}}, \tag{6.2.14}$$

so $\hat{\lambda}$ and $\mathbf{Z}\hat{\mathbf{x}}$ form a pair of eigenvalue and eigenvector of $\mathbf{N}$. Conversely, if λ a nonzero eigenvalue of $\mathbf{N}$ and $\mathbf{x}$ is its corresponding eigenvalue, then

$$\hat{\mathbf{N}}(\mathbf{Z}^T\mathbf{x}) = \left(\mathbf{Z}^T\mathbf{Z}\right)\left(\mathbf{Z}^T\mathbf{A}\mathbf{Z}\right)\mathbf{Z}^T\mathbf{x} = \mathbf{Z}^T\mathbf{N}\mathbf{x} = \lambda\mathbf{Z}^T\mathbf{x}. \tag{6.2.15}$$

Therefore, $\mathbf{N}$ and $\hat{\mathbf{N}}$ have the same nonzero eigenvalues. □

Based on the above proposition, the convergence rate of the Krylov subspace method on the PNS equation (6.2.2) is asymptotically the same as that on the null-space equation (6.1.1) with an orthonormal $\mathbf{Z}$. To accelerate the convergence of these methods, it is desirable to use preconditioners, which we discuss next.

6.3 Preconditioners

In OPINS, the most time consuming step is typically the iterative solver for the PNS equation (6.2.2) in step 4. To speed up its computation, it is critical to use preconditioners. In this section, we present some principles for constructing effective preconditioners for PNS equations, based on the recent work on the projected Krylov methods [37].

Let $\mathbf{N} = \Pi_{\mathbf{U}}^{\perp}\mathbf{A}\Pi_{\mathbf{U}}^{\perp}$. The general idea of preconditioning is to find a matrix $\mathbf{M}$ that approximates the coefficient matrix $\mathbf{N}$, or $\mathbf{M}^{+}$ that approximates the pseudoinverse $\mathbf{N}^{+}$. The latter form is more convenient for solving the PNS systems. Algorithm 8 outlines the pseudocode of the preconditioned OPINS with a left preconditioner. The preconditioning routine takes an operator $\mathbf{M}$ to evaluate $\mathbf{M}^{+}\mathbf{b}$ for any $\mathbf{b} \in \mathbb{R}^{n}$. Note that for symmetric systems, most preconditioned Krylov-subspace methods would apply the preconditioners symmetrically. Specifically, suppose $\mathbf{M}^{+}$ has a symmetric factorization $\mathbf{M}^{+} = \mathbf{L}\mathbf{L}^{T}$, then these methods solve the equation

$$\mathbf{L}^{T}\Pi_{\mathbf{U}}^{\perp}\mathbf{A}\Pi_{\mathbf{U}}^{\perp}\mathbf{L}\tilde{\mathbf{w}} = \mathbf{L}^{T}\Pi_{\mathbf{U}}^{\perp}\left(\mathbf{f} - \mathbf{A}\mathbf{x}_{p}\right) \tag{6.3.1}$$

in the preconditioned method, and then computes $\mathbf{w} = \mathbf{L}\tilde{\mathbf{w}}$. Typically, the algorithm is constructed such that the explicit factorization $\mathbf{M}^{+} = \mathbf{L}\mathbf{L}^{T}$ is not needed. We omit the details of such preconditioned Krylov-subspace methods; interested readers may refer to [5, 59].

Algorithm 8 Preconditioned OPINS

```
input: $\mathbf{A}$, $\mathbf{B}$, $\mathbf{f}$, $\mathbf{g}$, $\mathbf{G}$, tolerances for rank estimation and iterative solver
output: $\mathbf{x}$, $\mathbf{y}$ (optional)
1: do first three steps of Algorithm 7
2: solve $\mathbf{M}^{+}\Pi_{\mathbf{U}}^{\perp}\mathbf{A}\Pi_{\mathbf{U}}^{\perp}\mathbf{w} = \mathbf{M}^{+}\Pi_{\mathbf{U}}^{\perp}\left(\mathbf{f} - \mathbf{A}\mathbf{x}_{p}\right)$ using a preconditioned Krylov-subspace method
3: do the last two steps of Algorithm 7.
```

In this section, we will focus on the construction of $\mathbf{M}$ or $\mathbf{M}^{+}$. A straightforward choice of $\mathbf{M}$ is the approximation of $\mathbf{A}$. Possible candidates include SSOR-type preconditioners, incomplete factorization, and multigrid methods. We propose to approximate $\mathbf{N}^{+}$ with $\mathbf{M}^{+} = \mathbf{P}_{G} = \mathbf{Z}\left(\mathbf{Z}^{T}\mathbf{G}\mathbf{Z}\right)^{-1}\mathbf{Z}^{T}$, where $\mathbf{G}$ is an approximation of $\mathbf{A}$ and $\mathbf{Z}^{T}\mathbf{G}\mathbf{Z}$ is nonsingular. We refer to this preconditioner as the *projected preconditioner*. As we shall show later, for nonsingular systems it is equivalent to the constraint preconditioner in the projected Krylov methods [37, 47].

For symmetric systems, most preconditioned Krylov subspace methods require the preconditioner to be symmetric. Hence, we require that $\mathbf{G}$ is symmetric and $\mathbf{Z}^T\mathbf{G}\mathbf{Z}$ is SPD. Therefore, $\mathbf{P}_G$ is symmetric and positive semi-definite. The following proposition states that OPINS with the projected preconditioner is equivalent to applying $\mathbf{Z}^T\mathbf{G}\mathbf{Z}$ as a preconditioner in solving the corresponding null-space equation.

Proposition 5. *Assume $\mathbf{Z}^T\mathbf{G}\mathbf{Z}$ is SPD and the saddle-point system is symmetric. OPINS with the projected preconditioner is equivalent to applying $\mathbf{Z}^T\mathbf{G}\mathbf{Z}$ as the preconditioner for solving the null-space equation in the null-space method.*

Proof. If $\mathbf{Z}^T\mathbf{G}\mathbf{Z}$ is SPD, then so is $\left(\mathbf{Z}^T\mathbf{G}\mathbf{Z}\right)^{-1}$, which has a Cholesky factorization

$$\left(\mathbf{Z}^T\mathbf{G}\mathbf{Z}\right)^{-1} = \mathbf{L}_G\mathbf{L}_G^T. \tag{6.3.2}$$

Then $\mathbf{P}_G = \mathbf{Z}\left(\mathbf{Z}^T\mathbf{G}\mathbf{Z}\right)^{-1}\mathbf{Z}^T$ has a symmetric factorization

$$\mathbf{P}_G = \mathbf{L}\mathbf{L}^T = \mathbf{Z}\mathbf{L}_G(\mathbf{Z}\mathbf{L}_G)^T = \mathbf{Z}\mathbf{L}_G\mathbf{L}_G^T\mathbf{Z}^T, \tag{6.3.3}$$

where $\mathbf{L} = \mathbf{Z}\mathbf{L}_G$. If $\mathbf{P}_G$ is applied symmetrically, the preconditioned OPINS would solve the equation

$$\underbrace{\mathbf{L}_G^T\mathbf{Z}^T}_{\mathbf{L}^T}\underbrace{\mathbf{Z}\mathbf{Z}^T\mathbf{A}\mathbf{Z}\mathbf{Z}^T}_{\mathbf{N}}\underbrace{\mathbf{Z}\mathbf{L}_G}_{\mathbf{L}}\tilde{\mathbf{w}} = \underbrace{\mathbf{L}_G^T\mathbf{Z}^T}_{\mathbf{L}^T}\mathbf{Z}\mathbf{Z}^T\left(\mathbf{f}-\mathbf{A}\mathbf{x}_p\right), \tag{6.3.4}$$

where $\mathbf{Z}^T\mathbf{Z} = \mathbf{I}$ and $\mathbf{w} = \mathbf{L}_G\tilde{\mathbf{w}}$. Therefore, it is equivalent to solving

$$\mathbf{L}_G^T\underbrace{\mathbf{Z}^T\mathbf{A}\mathbf{Z}}_{\hat{\mathbf{N}}}\mathbf{L}_G\tilde{\mathbf{w}} = \mathbf{L}_G^T\mathbf{Z}^T\left(\mathbf{f}-\mathbf{A}\mathbf{x}_p\right), \tag{6.3.5}$$

which is equivalent to applying $\mathbf{Z}^T\mathbf{G}\mathbf{Z}$ as a symmetric preconditioner to (6.1.1). □

As shown in [47], the eigenvalues of the constraint-preconditioned null-space equation are well clustered if $\mathbf{G}$ is a good approximation of $\mathbf{A}$. Due to the above equivalence, the projected preconditioner is a good choice for OPINS. Note that if the system is singular, applying the preconditioner symmetrically may alter the null space of the coefficient matrix. Therefore, additional care must be taken to find the minimum-norm solution.

The remaining task is to find a way to provide $\mathbf{P}_G$ as an operator for efficient computation of $\mathbf{s} = \mathbf{P}_G\mathbf{b}$ for any $\mathbf{b} \in \mathbb{R}^n$. Note that $\mathbf{s}$ is the solution to the following modified but simpler saddle-point system

$$\begin{bmatrix} \mathbf{G} & \mathbf{U} \\ \mathbf{U}^T & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{s} \\ \mathbf{t} \end{bmatrix} = \begin{bmatrix} \mathbf{b} \\ \mathbf{0} \end{bmatrix}, \tag{6.3.6}$$

where $\mathbf{U}$ is composed of an orthonormal basis of range$(\mathbf{B}^T)$. Because from the null-space method, we have

$$\mathbf{s} = \mathbf{Z}\left(\mathbf{Z}^T\mathbf{G}\mathbf{Z}\right)^{-1}\mathbf{Z}^T\mathbf{b} = \mathbf{P}_G\mathbf{b}. \tag{6.3.7}$$

This implies that $\mathbf{P}_G$ can be given as an operator through the solution of (6.3.6). (6.1.3) is similar to the procedure for evaluating the constraint preconditioner in the projected Krylov method [37], for which the off-diagonal entries are $\mathbf{B}^T$ and $\mathbf{B}$ instead of $\mathbf{U}$ and $\mathbf{U}^T$. By replacing $\mathbf{B}^T$ by $\mathbf{U}$, (6.1.3) can be solved more efficiently, and it is also applicable if $\mathbf{B}$ is rank deficient.

If $\mathbf{G}$ is a simple matrix or operator, such as the preconditioner based on SSOR or incomplete factorization, we can solve (6.1.3) efficiently by using the range-space method, given by (6.1.5) and (6.1.6), especially when $m \ll n$. Algorithm 9 outlines the procedure for computing $\mathbf{P}_G\mathbf{b}$ using the range-space method, where $\mathbf{U}$ is composed of an orthonormal basis of $\mathbf{B}^T$.

Algorithm 9 Operator $\mathbf{P}_G$ for Projected Preconditioner

```
input: $\mathbf{G}$, $\mathbf{U}$, $\mathbf{b}$
output: $\mathbf{s} = \mathbf{P}_G\mathbf{b}$
1: solve $\mathbf{G}\mathbf{r} = \mathbf{b}$
2: solve $(\mathbf{U}^T\mathbf{G}^{-1}\mathbf{U})\mathbf{t} = \mathbf{U}^T\mathbf{r}$
3: solve $\mathbf{G}\mathbf{s} = \mathbf{b} - \mathbf{U}\mathbf{t}$
```

Table 6.1: Summary of test problems.

problem	len($\mathbf{x}$)	len($\mathbf{y}$)	rank($\mathbf{K}$)	nnz($\mathbf{K}$)	source
3d-var	3240	3	3243	18360	climate modeling [43]
sherman5	3312	20	3332	153273	nonsymmetric problem from [13]
mosarqp1	2500	700	3200	9434	quadratic programming [52]
fracture	780	92	786	10464	2-D nonlinear elasticity [7]
can_61	61	20	81	2997	symmetric problem from [13]
random	100	20	120	14000	random nonsingular matrix
random-s	100	20	90	14000	random singular matrix

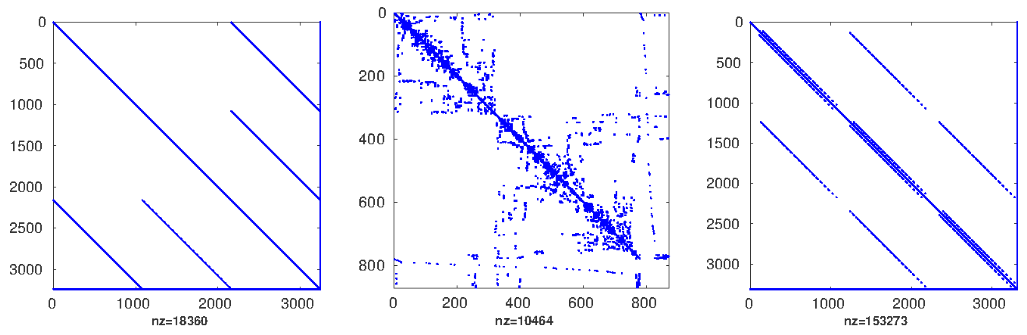
6.4 Numerical Experiments

In this section, we evaluate OPINS with various test problems. The experiments are mainly focused on symmetric nonsingular systems. For singular systems, please refer to Section 7.1. Some results of nonsymmetric systems are also included in Section 6.4.1. We start by evaluating the performance of OPINS with and without preconditioners, to demonstrate the importance of preconditioners and the effectiveness of the projected preconditioner. We then compare OPINS against some present state-of-the-art methods for symmetric nonsingular and singular systems. We use a few test problems, as summarized in Table 6.1 arising from constrained minimization, finite element analysis, climate modeling, and random matrices. Among these problems, the first six are sparse, and Figure 6.1 shows the sparsity patterns for some of their KKT matrices. If the right-hand sides were unavailable, we generate them by multiplying the matrix with a random vector. For all problems, we set the convergence tolerance to 10^{-10} for the residual in iterative methods, and set the tolerance for QRCP to 10^{-12}.

In terms of error measures, different methods use different convergence criteria internally. For a direct comparison, we present the convergence results in two difference measures. The first is the residual of $\mathbf{x}$ within the null space of $\mathbf{B}$, i.e.,

$$\mathbf{r} := \Pi_{\mathbf{Z}}(\mathbf{f} - \mathbf{A}\mathbf{x}_p) - \Pi_{\mathbf{Z}}\mathbf{A}\mathbf{x}_n, \tag{6.4.1}$$

where $\mathbf{x}_p$ is the particular solution in range$(\mathbf{B}^T)$ and $\mathbf{x}_n$ is the corresponding solution in null$(\mathbf{B})$. To make the metric scale independent, we measure the residual relative



(a) 3d-var (b) fracture (c) sherman5

Figure 6.1: Sparsity patterns of KKT matrices in 3d-var, fracture and sherman5.

to the right-hand side of (6.1.1), i.e.,

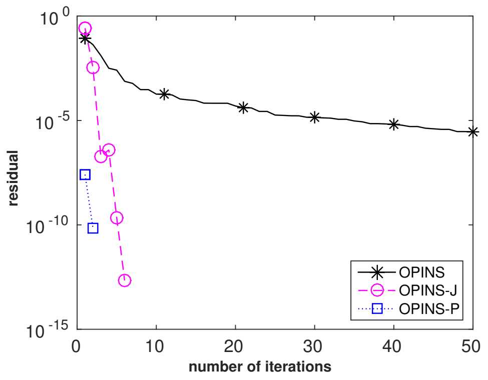
$$\text{relative residual in } \mathbf{x} := \|\mathbf{r}\| \,/\, \left\|\Pi_{\mathbf{Z}}\left(\mathbf{f} - \mathbf{A}\mathbf{x}_p\right)\right\|. \tag{6.4.2}$$

When comparing OPINS with methods that solve for **x** and **y** simultaneously, such as preconditioned Krylov methods, we calculate the residual in **x** by computing **U** using QR factorization. For a more complete comparison, in addition to the above error metric, we also compute the residual for the whole system (6.0.1) relative to the right-hand side in terms of both **x** and **y**.

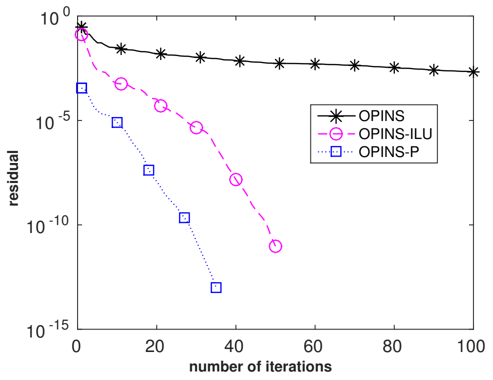
6.4.1 Effectiveness of Preconditioners

For Krylov subspace methods, the preconditioners have significant impact on the convergence rate. First let us consider symmetric systems. The core solver in OPINS is based on MINRES, so we assess the effectiveness of OPINS with a straightforward preconditioner as well as the projected preconditioner as described in Section 2.3. For simplicity, we choose **G** as the Jacobi preconditioner for both preconditioners, and denote them as OPINS-J and OPINS-P, respectively.

We solve the test case 3d-var with unpreconditioned OPINS, OPINS-J and OPINS-P. Figure 6.2(a) shows the convergence results measured in terms of the **x** residual. For 3d-var, the block **A** is nearly diagonal and is strongly diagonal dominant, so both preconditioners worked well and performed significantly better than unpreconditioned OPINS. Between the preconditioners, the projected preconditioner



(a) 3d-var



(b) sherman5

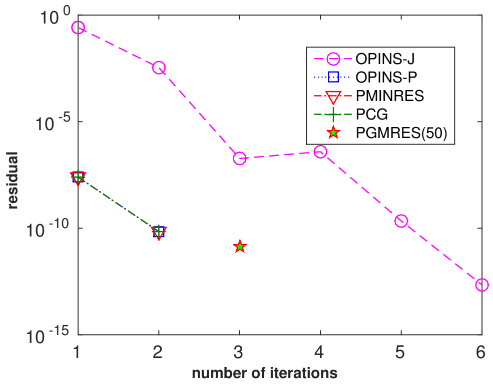
Figure 6.2: Convergence history of relative residual in **x** of OPINS versus preconditioned OPINS for 3d-var and sherman5.

performed better than the Jacobi preconditioner for MINRES alone, because the projected preconditioner provides a better approximation to the whole matrix. This indicates that the projected preconditioner is effective for accelerating OPINS, if **G** is a good approximation of **A**.

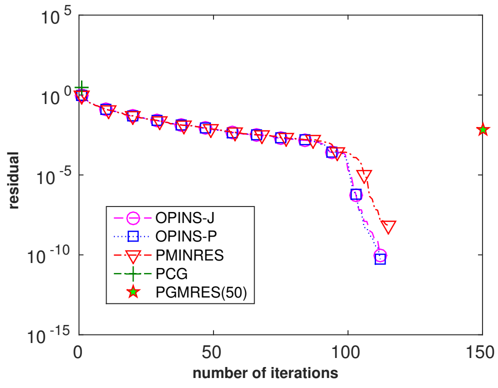
To demonstrate the applicability to nonsymmetric systems, we solve the problem sherman5 from the matrix market database [13]. **A** comes from an oil reservoir simulation and **B** is a random matrix. Since the system is nonsymmetric, we choose **G** as the ILU factorization of **A** for both preconditioners, and denote the two strategies as OPINS-ILU and OPINS-P, respectively. The inner solver is GMRES with the number of restarts set to 50. Figure 6.2(b) shows the convergence results for the three approaches. The results show that OPINS indeed works for nonsymmetric systems. Similar to the symmetric case, OPINS-P is faster than OPINS-ILU while both are much faster than unpreconditioned OPINS.

6.4.2 Assessment for Symmetric Nonsingular Systems

We now perform a more in-depth assessment of OPINS for symmetric nonsingular systems. Since unpreconditioned OPINS is not effective for nonsingular systems, we only consider OPINS-J and OPINS-P. We compare them against other Krylov subspace methods, including PMINRES and PCG with constraint preconditioners, which are equivalent to implicit null-space methods and are effective choices for the



(a) 3d-var.



(b) random.

Figure 6.3: Convergence history of relative residuals in **x** for OPINS and projected Krylov methods for 3d-var and random.

Table 6.2: Relative residual in $[\mathbf{x}, \mathbf{y}]$ upon convergence for OPINS and after convergence, stagnation, or excessive numbers of iterations of projected Krylov methods.

Problem	OPINS-J	OPINS-P	PMINRES	PCG	PGMRES(50)
3d-var	7.8×10^{-13}	2.5×10^{-10}	2.5×10^{-10}	2.5×10^{-10}	1.3×10^{-11}
mosarqp1	2.1×10^{-11}	3.9×10^{-11}	1.3×10^{-10}	1.3×10^{-10}	5.4×10^{-11}
random	1.2×10^{-12}	1.2×10^{-12}	3.5×10^{-4}	breakdown	1.5×10^{-4}

saddle-point systems that we are considering. As a reference, we also consider the preconditioned GMRES with the constraint preconditioner applied to the original system (6.0.1). The number of restarts is set to 50. We calculated the error for GMRES only after the solver has converged, so only one data point is available for it.

We solve three test problems, mosarqp1, 3d-var and random. Figure 6.3(a) and (b) show the convergence of the residual in terms of **x** for 3d-var and random. In 3d-var, all methods perform well with OPINS-P being similar to PMINRES. In terms of solution accuracy, all strategies give accurate $[\mathbf{x}, \mathbf{y}]$ as shown in Table 6.2.

In the random test, the **A** block is a 100×100 symmetric, nonsingular and indefinite matrix. **B** is a randomly generated 20×100 dense matrix with full rank. Figure 6.3(b) shows the **x** errors of different strategies. In this example, **A** is no longer positive definite, and PCG breaks at the initial iterations. Compared to PGMRES, OPINS and PMINRES converged much faster both in terms of **x** and in terms of

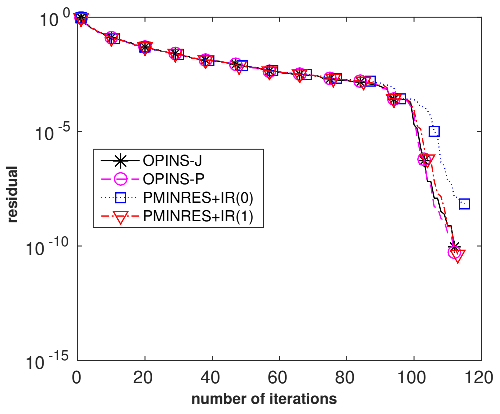
$[\mathbf{x}, \mathbf{y}]$. One reason is that OPINS and PMINRES are symmetric methods that can take advantage of a complete Krylov subspace basis. On the other hand, the $\mathbf{x}$ residual of PMINRES will stagnate around 10^{-8}. This is due to the instability of the algorithm as mentioned in [37]; see Subsection 6.4.3 for a more detailed analysis and comparison. Due to its stability and faster convergence, OPINS delivered the most accurate solution among all the methods both in terms of $\mathbf{x}$ and in terms of $[\mathbf{x}, \mathbf{y}]$.

6.4.3 Stability of OPINS versus PMINRES

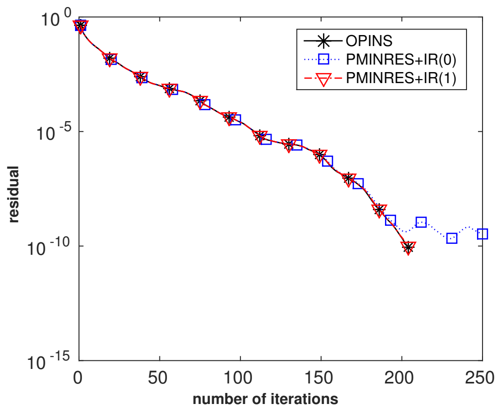
In this section, we study the stability of OPINS and PMINRES. The difference between PCG and PMINRES with a constraint preconditioner mainly lies in the underlying solver. Since MINRES is more robust than CG, we restrict our attention to PMINRES. In our previous discussion, PMINRES and OPINS with the projected preconditioner are both equivalent to a preconditioned null-space method. However PMINRES may stagnate for some problems, as shown in Figure 6.3 (b).

To further illustrate this, we consider the `random` test used in Subsection 6.4.2. OPINS with the two preconditioners are applied here. PMINRES uses the constraint preconditioner with $\mathbf{G}$ chosen as the diagonal part of $\mathbf{A}$. To examine the stability of PMINRES, we consider PMINRES+IR(0) and PMINRES+IR(1), which denote no iterative refinement and one step of iterative refinement, respectively. The constraint preconditioner is solved by factorization. Figure 6.4(a) shows the convergence of various strategies. It can be observed that OPINS-J, OPINS-P and PMINRES+IR(1) have similar convergence behaviors. On the other hand, PMINRES failed to converge to the desired tolerance if no iterative refinement is applied. Another example is the fracture problem. This system is singular, but the projected MINRES is also applicable. We set $\mathbf{G}$ to be the identity matrix in the constraint preconditioner. No preconditioner is used in OPINS. From Figure 6.4(b), we can see that OPINS is similar to the more stable PMINRES+IR(1), while the residual of PMINRES+IR(0) oscillates.

In addition, OPINS is more stable compared to PMINRES with iterative refinement, because the latter may still stagnate. We consider the example `can_61` from the matrix market database [13]. The matrix is set as the $\mathbf{A}$ part of the saddle-point



(a) Relative residuals in $\mathbf{x}$ for random.

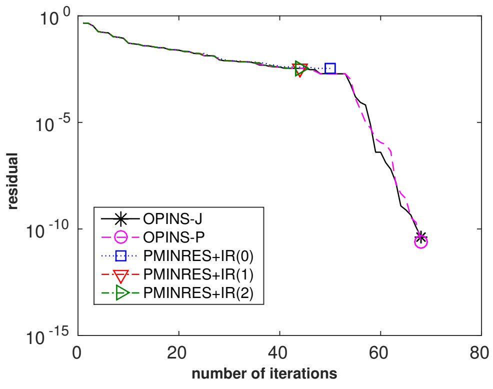


(b) Relative residuals in $\mathbf{x}$ for fracture.

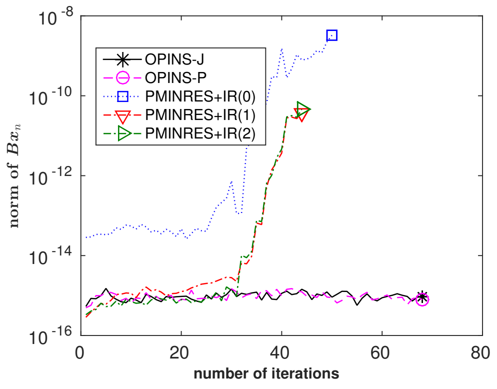
Figure 6.4: Comparison of OPINS and projected MINRES with/without iterative refinement for random and fracture.

system. $\mathbf{B}$ is a randomly generated 20×61 dense matrix with full rank. In this example, we include an additional strategy, PMINRES+IR(2) which uses two steps of iterative refinement per iteration. Figure 6.5(a) displays the residuals of the five strategies. Both OPINS-J and OPINS-P converged to the specified accuracy. However, PMINRES+IR(0) and PMINRES+IR(1) both stagnated early. Even additional iterative refinement could not increase the accuracy further as PMINRES+IR(2) still stagnated.

One reason for the stagnation of PMINRES is that $\mathbf{x}_n$ in the constraint preconditioner may deviate from null$(\mathbf{B})$ during the iteration. Eventually the large residual in $\mathbf{B}\mathbf{x}_n$ may cause MINRES to stagnate [37]. This is evident in Figure 6.5(b), where we plot $\|\mathbf{B}\mathbf{x}_n\|$ and $\left\|\mathbf{B}\Pi_{\mathbf{U}}^{\perp}\mathbf{w}\right\|$ for PMINRES and OPINS. For PMINRES with and without iterative refinement, $\|\mathbf{B}\mathbf{x}_n\|$ grew, and the growth of non null-space component was significant even with two steps of iterative refinement. In contrast, this norm stayed at close to machine precision for OPINS throughout the computation, because OPINS explicitly projects the solution onto null$(\mathbf{B})$. For this reason, OPINS is more stable than PMINRES.



(a) Convergence history of relative residual in $\mathbf{x}$.



(b) History of $\|\mathbf{Bx}_n\|$ in OPINS and PMINRES.

Figure 6.5: Comparison of OPINS and projected MINRES with/without iterative refinement for can_61, for which iterative refinement could not improve accuracy for projected MINRES due to loss of orthogonality.

Chapter 7

Applications

In this chapter, we apply the proposed solvers to various computational models such as elasticity with brittle fracture [50] and climate modeling. We also extend the HyGA framework to support weighted least squares based discretization techniques, such as the *generalised finite difference method (GFD)* [9] and the *adapted extended-stencil finite element method (AES-FEM)* [21].

7.1 Elasticity with Brittle Fracture

As a continuation of Chapter 6, we consider the application of OPINS to a finite element based brittle fracture model [50]. The model uses a quasi-static updated Lagrangian description of the elastic process [6]. For any solid body of some material, denoted at time τ as ${}^{\tau}V$, the material is in energetic equilibrium when the principle of virtual work is satisfied, namely that the body is in a minimum energy configuration. The principle of virtual work for a body at some future time $t+\Delta t$ can be written as

$$\int_{{}^{t+\Delta t}V} {}^{t+\Delta t}\tau_{ij}\delta_{t+\Delta t}e_{ij}{}^{t+\Delta t}dv = {}^{t+\Delta t}R, \tag{7.1.1}$$

where ${}^{t+\Delta t}\tau_{ij}$ is the Cauchy stress tensor of the material, ${}^{t+\Delta t}e_{ij}$ is the linear variation of linear strain tensor at time $t+\Delta t$, and ${}^{t+\Delta t}R$ is the total sum external virtual work due to surface tractions and body forces.

Since the configuration of the body at time $t+\Delta t$ is unknown, 7.1.1 cannot be solved directly and instead must be rewritten in term of some previously known configuration. In the updated Lagrangian formulation, 7.1.1 may be rewritten in terms of the material body in its current configuration at time t,

$$\int_{^tV} {}_tC_{ijrs}\, {}_t\varepsilon_{rs} \delta_t \varepsilon_{ij}\, {}^t dv + \int_{^tV} {}^t\tau_{ij} \delta_t \eta_{ij}\, {}^t dv = {}^{t+\Delta t}R - \int_{^tV} {}^t\tau_{ij} \delta_t e_{ij}\, {}^t dv, \qquad (7.1.2)$$

which is a nonlinear equation in the incremental nodal displacements u_i.

Equation 7.1.1 is nonlinear in general, and one needs to perform an iterative scheme to determine a solution at each step. We choose to use the modified Newton-Raphson scheme of [6] to iteratively determine the solution of 7.1.1. When some part of the brittle material exceeds its critical strain, we split overstrained nodes and generate new edges to introduce cracks. Lagrange multipliers are used to accommodate several types of Dirichlet boundary conditions and constraints used to avoid element penetration. The resulting linear system is a KKT system of the form 6.0.1.

In this system, $\mathbf{A}$ represents the global stiffness matrix. The constraint matrix $\mathbf{B}$ is used to enforce various boundary conditions, such as sliding boundary conditions and contact constraints. Due to the cracks opening along edges, some pieces are isolated completely from the main body. This introduces additional singularity to $\mathbf{A}$ such that $\operatorname{null}(\mathbf{A}) \cap \operatorname{null}(\mathbf{B}) \neq \{0\}$. In addition, $\mathbf{B}$ contains redundant constraints, and it is rank-deficient. We apply OPINS without preconditioners to solve the problem. PMINRES, PCG and PGMRES with the constraint preconditioner are also applied as comparisons. For PMINRES, PCG and PGMRES, $\mathbf{G}$ is set to be the identity in the constraint preconditioner. Since $\mathbf{B}$ is rank-deficient, an extra step of QR factorization is used to remove linearly dependent rows.

Convergence Comparison

Figure 7.1 shows the convergence of strategies. It can be seen that OPINS converges monotonically to the desired accuracy while the errors of PMINRES and PCG oscillate. In particular, the error of PCG starts to grow after a certain number of iterations. This shows that CG is not as robust as MINRES for solving singular systems. In terms of the overall residual, OPINS has around 10^{-9} while the residuals of PCG and PMINRES stay around 1. The reason why PCG and PMINRES

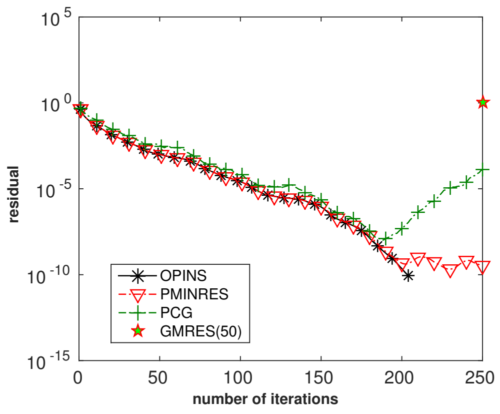


Figure 7.1: Convergence history of relative residual in x for OPINS and Krylov methods for singular system from fracture.

have large overall errors is that they do not necessarily minimize the error of $[\mathbf{x}, \mathbf{y}]$. To get a more accurate $\mathbf{y}$, we need to solve the least-squares problem $\mathbf{B}^T\mathbf{y} = \mathbf{f} - \mathbf{A}\mathbf{x}$ once convergence in $\mathbf{x}$ is reached. On the other hand, the residual of GMRES is about 10^{-6}. However the converged solution is incorrect, since $\|\mathbf{g} - \mathbf{B}\mathbf{x}\|$ is around 1. Overall OPINS is more stable and accurate than the other approaches.

Solution Norm

To illustrate the minimum norm property of OPINS, we compare OPINS with truncated SVD. The $\mathbf{x}$ and $\mathbf{y}$ components are measured by 2-norm. The relative 2-norm of the residuals are provided as references. In the fracture system, large Lamè parameters are used in calculating element stresses. As a result, there is a 10^{10} gap between the scaling of $\mathbf{A}$ and $\mathbf{B}$ which makes the system very ill-conditioned. The conditioning will be improved if we scale $\mathbf{A}$ and $\mathbf{f}$ by 10^{-10}. After the operation, $\mathbf{x}$ should remain unchanged while $\mathbf{y}$ will be scaled by 10^{-10} accordingly. We apply OPINS and truncated SVD (TSVD) to both scaled and unscaled systems, where the tolerance for TSVD was 10^{-12}. As shown in Table 7.1, OPINS is stable with and without scaling. Its $\mathbf{x}$ component stays unchanged, and it has the same norm as the TSVD solution on the scaled system. Without scaling, TSVD produced a wrong solution that is far away from the constraint hyperplane. This is evident from Table 7.2, where $\|\mathbf{g} - \mathbf{B}\mathbf{x}\|$ is very large for TSVD. In contrast, OPINS finds

Table 7.1: Norms of solutions of OPINS versus truncated SVD.

Test Problems	OPINS		TSVD	
	$\|\|\mathbf{x}\|\|$	$\|\|\mathbf{y}\|\|$	$\|\|\mathbf{x}\|\|$	$\|\|\mathbf{y}\|\|$
fracture	1.53×10^{-4}	1.79×10^{7}	2.86×10^{-5}	4.25×10^{-11}
scaled fracture	1.53×10^{-4}	1.79×10^{-3}	1.53×10^{-4}	1.54×10^{-3}
random-s	5.56	2.41	5.56	2.41

Table 7.2: Errors of solutions of OPINS versus truncated SVD.

Test Problems	OPINS		TSVD	
	$\|\|\mathbf{g}-\mathbf{Bx}\|\|$	$\|\|\mathbf{r}\|\|$	$\|\|\mathbf{g}-\mathbf{Bx}\|\|$	$\|\|\mathbf{r}\|\|$
fracture	1.0×10^{-17}	1.2×10^{-9}	1.1	1.9×10^{-10}
scaled fracture	8.6×10^{-18}	5.6×10^{-10}	1.1×10^{-14}	1.3×10^{-13}
random-s	3.7×10^{-16}	2.1×10^{-13}	2.3×10^{-14}	4.6×10^{-15}

the minimum-norm **x** independently of the scaling. Therefore, it is advantageous to solve **x** and **y** separately over solving them together, and it is not advisable to use any linear solver, including TSVD or GMRES, as black box solvers for saddle-point problems.

In terms of the **y** component, the solution of OPINS has a slightly larger norm than the TSVD solution for the scaled system. This is due to the rank deficiency of **B**. For the case where **B** has full rank, we consider a random system. **A** is a 100×100 dense and semi-definite matrix with rank $= 50$. **B** is a randomly generated 20×100 dense matrix with full rank. By construction, the system is singular with rank equal to 90. Since **A** is semi-definite and **B** has full rank, we can expect both **x** and **y** components of the OPINS solution to have minimum norms, which is evident in Table 7.1.

7.2 Climate Modeling

Our second model problem is a 2-D anisotropic Helmholtz equation in cloud modeling. The original 3-D equation can be expressed as

$$\frac{\partial^2 P}{\partial x^2} + \mu \frac{\partial}{\partial y} \mu \frac{\partial P}{\partial y} + \mu^2 \frac{\partial^2 P}{\partial z^2} = \mu^2 F, \tag{7.2.1}$$

with homogeneous Neumann boundary conditions. Due to the large problem size, it can be time consuming to solve the PDE directly. Noticing that the step size in x is usually uniform, we can apply FFT in the x direction and decompose the 3-D problem into a collection of 2-D equations of the following form

$$\mu \frac{\partial}{\partial y} \mu \frac{\partial P}{\partial y} + \mu^2 \frac{\partial^2 P}{\partial z^2} - \alpha P = \tilde{F}. \tag{7.2.2}$$

We refer to the equation as *Anisotropic Helmholtz equation*, or AHE.

In the above equation, each slice corresponds to a different x value. The x direction corresponds to longitude, y corresponds to latitude, and z corresponds to altitude. The parameter μ depends only on y, so it does not change with respect to x. The parameter α depends only on x, so it is a constant in the Helmholtz equation corresponding to different x. The right-hand side $\tilde{F}$ may vary depending on x. In general, $\alpha \geq 0$. When $\alpha = 0$, the Helmholtz equation reduces to a Poisson equation with pure Neumann boundary conditions, which does not have a unique solution. For simplicity, we only consider the case where $\alpha \neq 0$.

We discretize the PDE using a cell-centered finite-difference scheme on a structured grid. The 3-D grid is structured, and the spacing between the slices, i.e., the edge length in the x direction, is uniform. In addition, the yz grid is the same across all slices. Within each slice, the number of cells in the y and z directions are N_y and N_z, respectively. The cell sizes may vary along y and z directions. The edge lengths can be provided by two vectors $dy \in \mathbb{R}^{N_y}$ and $dz \in \mathbb{R}^{N_z}$, respectively. In general, the problem is anisotropic, with $dz \leqslant dy$, and sometimes $dz \ll dy$.

GMG Solver

To solve the problem, we employ the GMG solver in Chapter 4. Taking advantage of the fact that the yz grid is the same across all slices, we can setup the GMG data structure only once for $\alpha = 0$ and reuse it for all other α values. This can greatly reduce the total computational time, particularly if GMG converges quickly for each α. Note that there is no simple way to reuse the data structure in AMG. Even if we store the structure for $\alpha = 0$, each different α modifies the coefficient matrix which in turn changes the multigrid operators.

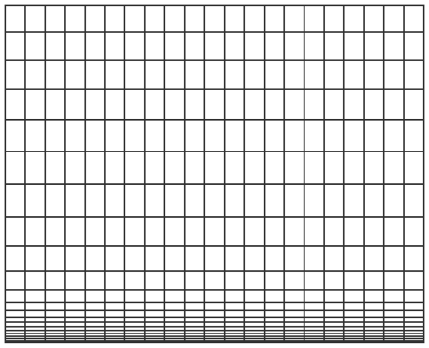


Figure 7.2: Example nonuniform structured mesh for the climate model.

To demonstrate the effectiveness of the solver, we compare it against the preconditioned GMRES in PETSc. Hypre is used as the preconditioner as it delivers better performance than the other alternatives. The tests are performed in serial on a laptop with a dual-core 2.53GHz intel i5 CPU and 4gb memory. PETSc is built in optimized mode with gcc 4.6.3 and option -O3. We consider three test cases. Figure 7.2 illustrates an example of the computational mesh. The first two problems are manufactured using $f = \sin\left(\frac{2\pi y}{N_y \cdot dy}\right)\sin\left(\frac{2\pi z}{N_z \cdot dz}\right)$ as the analytical solution. The third test is constructed from the real data set. Table 7.3 shows the timing results measured in seconds. We can see that GMG is much faster than Hypre in all the test cases and its rate of convergence stays consistent. If one considers solving multiple αs on the same grid, the advantage of GMG is even greater.

Table 7.3: Timing comparison of AMG and GMG. The results are measured in seconds.

(N_y, N_z, α)	GMG		Hypre	
	iter	solve (secs)	iter	solve (secs)
$(2071, 41, 1e-8)$	3	0.07	4	0.24
$(3300, 100, 1e-14)$	4	0.40	29	4.32
$(1024, 34, 2e-14)$	5	0.03	9	0.12

7.3 HyGA for WLS-based Discretization Methods

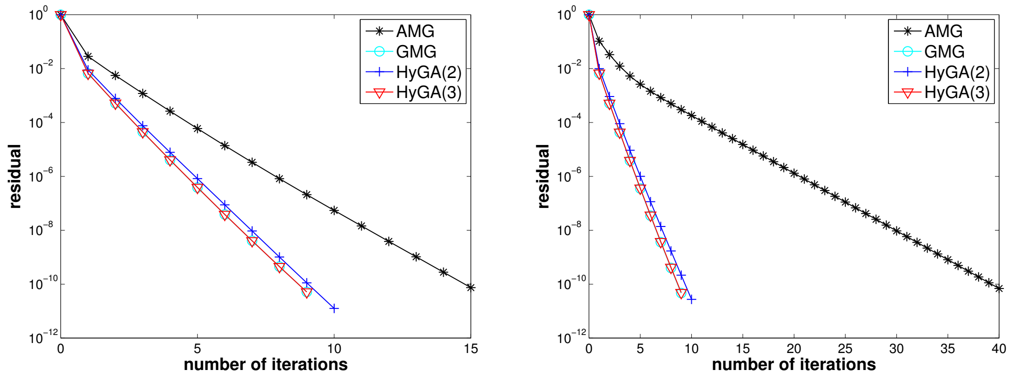
Finally, we adapt HyGA to support two weighted least squares based discretization methods: GFD and AES-FEM. GFD extends the classical finite differences to unstructured meshes with high-order accuracy [9]. The GFD methods can be derived based on a weighted least squares (WLS) formulation locally over a weighted stencil at each point, which generalizes the classical interpolation-based finite differences. It has been shown that the WLS can deliver the same order of accuracy as the interpolation-based approximations, while delivering better stability, flexibility, and robustness for multivariate approximations over irregular unstructured meshes.

Similar to GFD, the adaptive extended stencil finite element method (AES-FEM) [21] is proposed to overcome the dependence of the element shape quality. Its better stability comes from the fact that it uses local WLS approximations as the basis function. Although AES-FEM has different basis functions, it preserves the theoretical framework of FEM and thus shares the same multigrid algorithm. As such, we will mainly focus on GFD in the following section.

HyGA for GFD

The multi-level weighted-residual formulation in Section 5.1 is applicable to GFD. As described in Section 5.1.2, the prolongation operator $\mathbf{P}$ for GFD is an interpolation matrix, but the restriction operator $\mathbf{R}$ is its scaled transpose with unit row sums, because the test functions ψ_j are Dirac delta functions. Applying these operators to adjacent grids in a multilevel method, we then obtain a geometric multigrid for GFD, where the coefficient matrices at all levels are obtained from re-discretizations with GFD. Coupling with algebraic multigrid at the coarser levels, we then obtain HyGA for GFD.

We applied HyGA to GFD using the same FEM meshes for the 2-D geometry in Figure 5.4(left). One thing to note here is that the matrix is nonsymmetric. As in the previous test case, we use 2 iterations of presmoothing and post smoothing iterations. For the parameters in CJ, we chose $\lambda_1^{'}$ to be $2/3$ as in the FEM case. We obtained $\lambda_n^{'}$ by estimating the largest eigenvalue of $\mathbf{D}^{-1}\left(\mathbf{A}+\mathbf{A}^{T}\right)/2$. The resulting estimation turned out to be very accurate. For example, for the 2-D 5-level mesh, the smallest real part of λ_n is ≈ -0.746, while the estimated λ_n' is $1 - 1.747 = -0.747$, whose accuracy is more than sufficient. Figure 7.3 shows the convergence results for GFD. We can see that HyGA converges nicely.



(a) 35,986 unknowns with 5 levels.

(b) 144,674 unknowns with 6 levels.

Figure 7.3: Relative residual versus numbers of iterations for 2-D GFD tests.

Chapter 8

Conclusions and Future Work

In this dissertation, we considered efficient iterative and multigrid solvers for linear systems. Multigrid methods, in particular the geometric multigrid methods, are the optimal choices for many application problems. We demonstrated that geometric multigrid methods, when applicable, can be much faster than the alternatives, which include AMG standalone solvers and AMG preconditioners. The AMG preconditioners, in turn, are better than SSOR and ILU. One limitation of GMG is that the algorithm is very problem dependent and it is hard to generalize to unstructured meshes.

For 2-D anisotropic Helmholtz equation, we presented an effective GMG solver with line-smoother and bi-linear interpolation. Line-smoothers are essential for smoothing out errors geometrically while bi-linear interpolation with ghost cells is a natural approach to construct the transfer operators. To generalize GMG further, we introduced a hybrid geometric+algebraic multigrid framework, *HyGA*, with semi-iterative smoothers. This approach is motivated by a new trend of large-scale parallel mesh generation through mesh refinements. The HyGA multigrid solver utilizes a few levels of geometric multigrid on these hierarchical meshes for accuracy and efficiency, along with algebraic multigrid on the coarse-levels for simplicity and robustness. We presented a unified derivation of the prolongation and restriction operators on the GMG levels for weighted-residual methods with hierarchical basis functions, including finite elements and general WLS-based methods with hierarchical unstructured meshes. The semi-iterative smoother, Chebyshev-

Jacobi method exhibits nice convergence properties and it can be parallelized easily. Our numerical experiments demonstrated the advantages of our proposed hybrid technique compared to the classical GMG and AMG methods.

Our third contribution is the orthogonally projected implicit null-space method (OPINS) for symmetric saddle point systems. The systems can be nonsingular or singular but compatible. Instead of finding the null space of the constraint matrix explicitly, OPINS uses an orthonormal basis of its orthogonal complementary subspace to reduce the saddle-point system to a singular but compatible system. We showed that OPINS is equivalent to a null-space method with an orthonormal basis for nonsingular systems. In addition, it can solve singular systems and produce the minimum-norm solution, which is desirable for many applications. Because of its use of orthogonal projections, OPINS is more stable than other implicit null-space methods, such as the projected Krylov methods.

Future Work

This work opens up several directions for future work. For OPINS, an immediate question is how to apply the multigrid preconditioner. Since OPINS uses the Krylov subspace method as the core solver, its performance can be accelerated significantly if we can find a way to apply multigrid. One possible approach is to apply the multigrid method on the $(1,1)$ block and use it directly as a preconditioner to the projected null-space equation. However this approach may not be very effective as it does not directly approximate the coefficient matrix. A more effective preconditioner is the projected Krylov preconditioner. Yet combing multigrid with the projected Krylov preconditioner is challenging since multigrid is an implicit operator. More studies on this area are needed.

On the multigrid side, we can work on the parallelization of the algorithm. One of our long term goals is to develop a parallel multigrid package which includes a framework for geometric multgrid method as well as HyGA. We can continue our efforts in the integration with PETSc, but similar work has been done in [49]. Alternatively we can consider shared-memory implementation with OpenMP, CUDA or OpenACC. At the moment, there is a lack of good multi-threaded linear solvers. We believe a multi-threaded GMG and HyGA would be a good contribution to the scientific computing community.

Finally there are some open questions about efficient solvers for singular systems. For symmetric systems, MINRES and MINRES-QLP with a multigrid preconditioner have the potential to deliver good performance. However, the preconditioner has to be applied symmetrically, which alters both the residual and the norm of the solution. Nonsymmetric methods such as GMRES, when applicable, may work with a one-sided preconditioner. The downside is that they are often based on the Krylov subspace $\mathscr{K}_k(\mathbf{A}, \mathbf{b})$, which is the wrong space for the minimum-norm solution. With a right preconditioner, the pseudo-inverse solution can be recovered by projecting off the null-space component if the null-space is known. Unfortunately, finding the null-space is hard in general. We hope to design a method that is based on $\mathscr{K}_k(\mathbf{A}^T, \mathbf{b})$ with similar efficiency as GMRES.

Bibliography

[1] M. Adams, M. Brezina, J. Hu, and R. Tuminaro. Parallel multigrid smoothing: polynomial versus Gauss-Seidel. *J. Comput. Phys.*, 188(2):593–610, 2003.

[2] M. F. Adams. Algebraic multigrid methods for constrained linear systems with applications to contact problems in solid mechanics. *Numer. Linear Algebra Appl.*, 11(2-3):141–153, 2004.

[3] W. E. Arnoldi. The principle of minimized iterations in the solution of the matrix eigenvalue problem. *Quarterly of Applied Mathematics*, 9:17–29, 1951.

[4] R. E. Bank, B. D. Welfert, and H. Yserentant. A class of iterative methods for solving saddle point problems. *Numer. Math.*, 56(7):645–666, 1989.

[5] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. SIAM, Philadelphia, PA, 2nd edition, 1994.

[6] K. A. Bathe, E. Ramm, and E. L. Wilson. Finite element formulations for large deformation dynamic analysis. *Int. J. Num. Method. Engrg.*, 9(2):353–386, 1975.

[7] K. J. Bathe, E. Ramm, and E. L. Wilson. Finite element formulations for large deformation dynamic analysis. *Int. J. Numer. Meth. Engrg.*, 9(2):353–386, 1975.

[8] W. Bell, L. Olson, and J. Schroder. Pyamg: Algebraic multigrid solvers in python v2. 0. *h ttp://www. pyamg. org*, 2011.

[9] J. J. Benito, F. Ureña, and L. Gavete. The generalized finite difference method. In M. P. Álvarez, editor, *Leading-Edge Applied Mathematical Modeling Research*, chapter 7. Nova Science Publishers, Inc., 2008.

[10] M. Benzi, G. H. Golub, and J. Liesen. Numerical solution of saddle point problems. *Acta Numer.*, 14:1–137, 5 2005.

[11] M. Benzi and A. Wathen. Some preconditioning techniques for saddle point problems. In W. H. A. Schilders, H. A. van der Vorst, and J. Rommes, editors, *Model Order Reduction: Theory, Research Aspects and Applications*, volume 13 of *Mathematics in Industry*, pages 195–211. Springer, 2008.

[12] B. K. Bergen and F. Hülsemann. Hierarchical hybrid grids: data structures and core algorithms for multigrid. *Numerical linear algebra with applications*, 11(2-3):279–291, 2004.

[13] R. F. Boisvert, R. Pozo, K. Remington, R. F. Barrett, and J. J. Dongarra. Matrix market: A web resource for test matrix collections. In *Proceedings of the IFIP TC2/WG2.5 Working Conference on Quality of Numerical Software: Assessment and Enhancement*, pages 125–137, London, UK, 1997. Chapman & Hall, Ltd.

[14] J. H. Bramble and J. E. Pasciak. A preconditioning technique for indefinite systems resulting from mixed approximations of elliptic problems. *Math. Comput.*, 50(181):1–17, 1988.

[15] A. Brandt, S. McCormick, and J. Ruge. Algebraic multigrid (amg) for sparse matrix equations. Cambridge University Press, 1984.

[16] W. L. Briggs, V. E. Henson, and S. F. McCormick. *A Multigrid Tutorial*. SIAM, second edition edition, 2000.

[17] T. F. Chan. Rank revealing QR factorizations. *Linear Algebra Appl.*, 88:67–82, 1987.

[18] T. F. Chan, E. Gallopoulos, V. Simoncini, T. Szeto, and C. H. Tong. A quasi-minimal residual variant of the Bi-CGSTAB algorithm for nonsymmetric systems. *SIAM J. Sci. Comput.*, 15(2):338–347, 1994.

[19] S. Choi, C. C. Paige, and M. A. Saunders. MINRES-QLP: A Krylov subspace method for indefinite or singular symmetric systems. *SIAM J. Sci. Comput.*, 33(4):1810–1836, 2011.

[20] A. J. Clayton. Further results on polynomials having least maximum modulus pver an ellipse in the complex plane. Technical Report AEEW-M348, United Kingdom Atomic Energy Authority, Winfrith, Dorchester, 1963.

[21] R. Conley, T. J. Delaney, and X. Jiao. Overcoming element quality dependence of finite elements with adaptive extended stencil FEM (AES-FEM). *Int. J. Num. Method. Engrg.*, 2016.

[22] H. De Sterck, U. M. Yang, and J. J. Heys. Reducing complexity in parallel algebraic multigrid preconditioners. *SIAM Journal on Matrix Analysis and Applications*, 27(4):1019–1039, 2006.

[23] E. de Sturler and J. Liesen. Block-diagonal and constraint preconditioners for nonsymmetric indefinite linear systems. Part I: Theory. *SIAM J. Sci. Comput.*, 26(5):1598–1619, 2005.

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

[25] R. D. Falgout and U. M. Yang. Hypre: A library of high performance preconditioners. In *International Conference on Computational Science*, pages 632–641. Springer, 2002.

[26] R. Fletcher. Conjugate gradient methods for indefinite systems. In *Numerical analysis*, pages 73–89. Springer, 1976.

[27] D. Fong and M. M. Saunders. LSMR: An iterative algorithm for sparse least-squares problems. *SIAM J. Sci. Comput.*, 33(5):2950–2971, 2011.

[28] D. C.-L. Fong and M. A. Saunders. CG versus MINRES: An empirical comparison. *SQU J. Sci.*, 17(1):44–62, 2012.

[29] R. W. Freund. A transpose-free quasi-minimal residual algorithm for non-Hermitian linear systems. *SIAM J. Sci. Comput.*, 14(2):470–482, 1993.

[30] R. W. Freund and N. M. Nachtigal. QMR: a quasi-minimal residual method for non-Hermitian linear systems. *SIAM J. Numer. Math.*, 60:315–339, 1991.

[31] T.-P. Fries, A. Byfut, A. Alizada, K. W. Cheng, and A. Schröder. Hanging nodes and XFEM. *Int. J. Numer. Meth. Engrg.*, 86(4-5):404–430, 2011.

[32] M. Gee, C. Siefert, J. Hu, R. Tuminaro, and M. Sala. ML 5.0 smoothed aggregation user's guide. Technical Report SAND2006-2649, Sandia National Laboratories, Albuquerque, NM, 2006.

[33] J. R. Gilbert and M. T. Heath. Computing a sparse basis for the null space. *SIAM J. Alg. Disc. Meth.*, 8(3):446–459, 1987.

[34] G. H. Golub and C. F. V. Loan. *Matrix Computations*. Johns Hopkins, 4th edition, 2013.

[35] G. H. Golub and R. S. Varga. Chebyshev semi-iterative methods, successive overrelaxation iterative methods, and second-order Richardson iterative methods. *Numer. Math. Parts I and II*, 3:147–168, 1961.

[36] N. I. M. Gould, M. E. Hribar, and J. Nocedal. On the solution of equality constrained quadratic programming problems arising in optimization. *SIAM J. Sci. Comput.*, 23(4):1376–1395, 2001.

[37] N. I. M. Gould, D. Orban, and T. Rees. Projected krylov methods for saddle-point systems. *SIAM J. Matrix Anal. & Appl.*, 35(4):1329–1343, 2014.

[38] A. Greenbaum. *Iterative Methods for Solving Linear Systems*. SIAM, 1997.

[39] A. Greenbaum and L. Gurvits. Max-min properties of matrix factor norms. *SIAM J. Sci. Comput.*, 15(2):348–358, 1994.

[40] L. A. Hageman and D. M. Young. *Applied Iterative Methods*. Dover Books on Mathematics. Dover Publications, 2004.

[41] V. E. Henson and U. M. Yang. Boomeramg: a parallel algebraic multigrid solver and preconditioner. *Applied Numerical Mathematics*, 41:155–177, 2002.

[42] M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *J. Res. Nat. Bur. Stand.*, 49(6), 1952.

[43] J. Huang. *Constrained Variational Analysis Integrating Vertical and Temporal Correlations*. PhD thesis, Applied Math and Statistics, Stony Brook University, 2012.

[44] F. Hulsemann, M. Kowarschik, M. Mohr, and U. Rude. Parallel geometric multigrid. In *Numerical Solution of Partial Differential Equations on Parallel Computers, volume 51 of LNCSE, chapter 5*, pages 165–208, 2005.

[45] X. Jiao and D. Wang. Reconstructing high-order surfaces for meshing. *Engineering with Computers*, 28:361–373, 2012.

[46] X. Jiao, D. Wang, and H. Zha. Simple and effective variational optimization of surface and volume triangulations. *Engineering with Computers*, 27:81–94, 2011.

[47] C. Keller, N. I. M. Gould, and A. J. Wathen. Constraint preconditioning for indefinite linear systems. *SIAM J. Matrix Anal. & Appl.*, 21(4):1300–1317, 2000.

[48] C. Lanczos. An iterative method for the solution of the eigenvalue problem of linear differential and integral operaators. *J. Res. Nat. Bur. Staandards, Sec. B*, 45:225–280, 1950.

[49] M. Lange, L. Mitchell, M. G. Knepley, and G. J. Gorman. Efficient mesh management in firedrake using petsc-dmplex. *arXiv preprint arXiv:1506.07749*, 2015.

[50] W. Li, T. Delaney, X. Jiao, R. Samulyak, and C. Lu. Finite element model for brittle fracture and fragmentation. *Procedia Computer Science*, 80:245–256, 2016.

[51] C. Lu, X. Jiao, and N. Missirlis. A hybrid geometric + algebraic multigrid method with semi-iterative smoothers. *Numer. Linear Algebra Appl.*, 21(2):221–238, 2014.

[52] I. Maros and C. Mészáros. A repository of convex quadratic programming problems. *Optim. Method. Softw.*, 11(1-4):671–681, 1999.

[53] D. Mavriplis and A. Jameson. Multigrid solution of the euler equations on unstructured and adaptive meshes. In *3rd Copper Mountain Conference on Multigrid Methods*, Copper Mountain, CO, 1987. Paper 23.

[54] N. M. Nachtigal, S. C. Reddy, and L. N. Trefethen. How fast are nonsymmetric matrix iterations? *SIAM J. Matrix Anal. Appl.*, 13(3):778–795, 1992.

[55] J. Nocedal and S. Wright. *Numerical Optimization*. Springer Science & Business Media, 2006.

[56] C. C. Paige and M. A. Saunders. Solution of sparse indefinite systems of linear equations. *SIAM J. Numer. Anal.*, 12(4):617–629, 1975.

[57] C. C. Paige and M. A. Saunders. LSQR: An algorithm for sparse linear equations and sparse least squares. *ACM Trans. Math. Softw.*, 8(1):43–71, Mar. 1982.

[58] L. Reichel and Q. Ye. Breakdown-free GMRES for singular systems. *SIAM J. Matrix Anal. Appl.*, 26(4):1001–1021, 2005.

[59] Y. Saad. *Iterative methods for sparse linear systems*. SIAM, 2nd edition, 2003.

[60] Y. Saad and M. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 7(3):856–869, 1986.

[61] J. Schöberl and W. Zulehner. Symmetric indefinite preconditioners for saddle point problems with applications to PDE-constrained optimization problems. *SIAM J. Matrix Anal. & Appl.*, 29(3):752–773, 2007.

[62] V. Shaidurov. *Multigrid methods for finite elements.* Springer, 1995.

[63] J. R. Shewchuk. Triangle: Engineering a 2d quality mesh generator and Delaunay triangulator. In *Lecture Notes in Computer Science*, volume 1148, pages 203–222, 1996.

[64] H. Si. TetGen, a quality tetrahedral mesh generator and three-dimensional Delaunay triangulator v1.4, 2006.

[65] P. Sonneveld. CGS, a fast Lanczos-type solver for nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 10(1):36–52, 1989.

[66] L. N. Trefethen and D. Bau. *Numerical Linear Algebra.* SIAM, 1997.

[67] U. Trottenberg, C. W. Oosterlee, and A. Schuller. *Multigrid.* Academic Press, 2000.

[68] M. Tuma. A note on the LDL^T decomposition of matrices from saddle-point problems. *SIAM J. Matrix Anal. & Appl.*, 23(4):903–915, 2002.

[69] H. A. van der Vorst. Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 13(2):631–644, 1992.

[70] H. A. Van der Vorst. *Iterative Krylov Methods for Large Linear Systems*, volume 13. Cambridge University Press, 2003.

[71] P. Vanek, M. Brezina, and J. Mandel. Convergence of algebraic multigrid based on smoothed aggregation. *Numerische Mathematik*, 88:559–579, 2001.

[72] R. Varga. *Matrix iterative analysis.* Prentice-Hall, Englewood Cliffs, NJ, 1962.

[73] S. A. Vavasis. Stable numerical algorithms for equilibrium systems. *SIAM J. Matrix Anal. & Appl.*, 15(4):1108–1131, 1994.

[74] C. Xu and J. L. Prince. Snakes, shapes, and gradient vector flow. *IEEE Transactions on Image Processing*, 7(3):359–369, 1998.