A Parallel Aggregation Algorithm for Inter-Grid Transfer Operators in Algebraic Multigrid

Nilton Alan Garcia Hilares

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Mathematics

Mark P. Embree, Chair
Eric De Sturler
Timothy Warburton

September 13, 2019
Blacksburg, Virginia

Keywords: Algebraic multigrid, Aggegation, MIS(2), Poisson's Equation

A Parallel Aggregation Algorithm for Inter-Grid Transfer Operators in Algebraic Multigrid

Nilton Alan Garcia Hilares

(ABSTRACT)

As finite element discretizations ever grow in size to address real-world problems, there is an increasing need for fast algorithms. Nowadays there are many GPU/CPU parallel approaches to solve such problems.

Multigrid methods can be used to solve large-scale problems, or even better they can be used to precondition the conjugate gradient method, yielding better results in general. Capabilities of multigrid algorithms rely on the effectiveness of the inter-grid transfer operators. In this thesis we focus on the aggregation approach, discussing how different aggregation strategies affect the convergence rate. Based on these discussions, we propose an alternative parallel aggregation algorithm to improve convergence. We also provide numerous experimental results that compare different aggregation approaches, multigrid methods, and conjugate gradient iteration counts, showing that our proposed algorithm performs better in serial and parallel.

A Parallel Aggregation Algorithm for Inter-Grid Transfer Operators in Algebraic Multigrid

Nilton Alan Garcia Hilares

(GENERAL AUDIENCE ABSTRACT)

Modeling real-world problems incurs a high computational cost because these mathematical models involve large-scale data manipulation. Thus we need fast and efficient algorithms. Nowadays there are many high-performance approaches for these problems.

One such method is called the Multigrid algorithm. This approach models a physical domain using a hierarchy of grids, and so the effectiveness of these approaches relies on how well data can be transferred from grid to grid. In this thesis, we focus on the aggregation approach, which clusters a grid's vertices according to its connections. We also provide an alternative parallel aggregation algorithm to give a faster solution. We show numerous experimental results that compare different aggregation approaches and multigrid methods, showing that our proposed algorithm performs better in serial and parallel than other popular implementations.

Table of Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Large-scale computational simulations demand efficient methods to solve scientific and engineering problems in real time. Algebraic Multigrid (AMG) methods are a class of solvers intended originally to solve linear systems, but they can also be used to precondition the conjugate gradient method, achieving more efficient results as a solver. Since the linear systems that arise from the finite element discretization are sparse, and for real applications we expect large-scale problems, we seek algorithms that performs well on high performance hardware including parallel machines and GPUs.

There are many different AMG approaches. In this work we will focus on the aggregation approach, taking as reference the classical smooth/rough aggregation and the maximal independent set of distance two, MIS(2), as references for our proposed parallel aggregation algorithm. Let us start by presenting our case of study; the *Poisson's equation*, and the continuous Galerkin finite element method, before discussing multigrid methods. This and more detailed information can be found in [2], [6], [7], [8], [13], and [12].

1.1 Finite Elements for the Poisson's Equation

Given a function $u = u(x_1, x_2)$, the function

$$\Delta u = \frac{\partial^2 u}{\partial x_1^2} + \frac{\partial^2 u}{\partial x_2^2}$$

is called the two-dimensional *Laplacian* of u.

Perhaps the most important partial differential equation is *Laplace's equation*

$$\Delta u = 0, \tag{1.1}$$

and its inhomogeneous version, *Poisson's equation*

$$-\Delta u = f. \tag{1.2}$$

Both (1.1) and (1.2) are posed on a bounded open domain $\Omega \subset \mathbb{R}^2$, with boundary Γ. The unknown is $u : \overline{\Omega} \to \mathbb{R}, u = u(\mathbf{x})$, where $\mathbf{x} \in \Omega$. In (1.2), the function $f : \Omega \to \mathbb{R}$ is also given.

Now consider the boundary value problem for Poisson's equation

$$\begin{array}{rcll} -\Delta u & = & f & \text{in } \Omega, \\ u & = & 0 & \text{on } \Gamma. \end{array} \tag{1.3}$$

Many different problems in physics and mechanics are modeled by (1.3). For instance, in particular settings u can represent: a temperature, a steady fluid flow, an electro-magnetic potential, Brownian motion, or the displacement of an elastic membrane, etc.

Now, recall Green's formula

$$\int_\Omega \nabla v \cdot \nabla w \, \mathrm{d}\mathbf{x} = \int_\Gamma v \frac{\partial w}{\partial \mathbf{n}} \, \mathrm{d}\mathbf{s} - \int_\Omega v \Delta w \, \mathrm{d}\mathbf{x},$$

where $\mathbf{n} = (n_1, n_2)$ is the outward unit normal to Γ. Here $\mathrm{d}\mathbf{x}$ denotes the element of area in $\mathbb{R}^2$, and $\mathrm{d}s$ the element of arc length along Γ, ∇v is the gradient of v, and $\partial w/\partial \mathbf{n}$ is the normal derivative, i.e.,

$$\nabla v = \left(\frac{\partial v}{\partial x_1}, \frac{\partial v}{\partial x_2} \right) \quad \text{and} \quad \frac{\partial w}{\partial \mathbf{n}} = \frac{\partial w}{\partial x_1} n_1 + \frac{\partial w}{\partial x_2} n_2.$$

We shall now give a variational formulation of problem (1.3). Let us start by showing that if u satisfies (1.3), then u is a solution of the following variational problem.

Find $u \in \mathcal{V}$ such that

$$a(u, v) = \langle f, v \rangle \quad \text{for all } v \in \mathcal{V}, \tag{1.4}$$

where $a(u, v) = \displaystyle\int_\Omega \nabla u \cdot \nabla v \, \mathrm{d}\mathbf{x}$, $\langle f, v \rangle = \displaystyle\int_\Omega f v \, \mathrm{d}\mathbf{x}$, and

$$\mathcal{V} = \left\{ v \in \mathcal{C}(\Omega) \ : \ \frac{\partial v}{\partial x_1}, \frac{\partial v}{\partial x_2} \in \mathcal{PC}(\Omega) \text{ and } v = 0 \text{ on } \Gamma \right\},$$

where $\mathcal{C}(\Omega)$ and $\mathcal{PC}(\Omega)$ are the space of continuous and piecewise continuous functions on Ω.

To see that (1.4) follows from (1.3), we multiply (1.3) by an arbitrary test function $v \in \mathcal{V}$ and integrate over Ω. Applying Green's formula, we get

$$\langle f, v \rangle = -\int_\Omega \Delta u \; v \, \mathrm{d}\mathbf{x} = -\int_\Gamma \frac{\partial u}{\partial \mathbf{n}} \; v \, \mathrm{d}s + \int_\Omega \nabla u \cdot \nabla v \, \mathrm{d}\mathbf{x} = a(u, v).$$

Since $v = 0$ on Γ, the boundary integral vanishes. Additionally, if $u \in \mathcal{V}$ satisfies (1.4) and u is sufficiently regular, one can show that u satisfies (1.3).

Now construct a finite-dimentional subspace $\mathcal{V}_h$ of $\mathcal{V}$. For simplicity consider that Γ is a polygonal curve; in this case we say that Ω is a polygonal domain. (If Γ is curved, we can first approximate Γ with a polygonal curve.) A *triangulation* of Ω is made by subdividing Ω into a set $\mathcal{T}_h = \{T_1, T_2, \ldots, T_m\}$ of non-overlapping triangles T_i, such that

$$\Omega = \bigcup_{T \in \mathcal{T}_h} T = T_1 \cup T_2 \cup \cdots \cup T_m,$$

and no vertex of a triangle lies on the edge of another triangle.

Define the mesh parameter h as

$$h = \max_{T \in \mathcal{T}_h} \{\text{diam}(T)\}$$

where $\text{diam}(T)$ is the largest edge of T. Then we define

$$\mathcal{V}_h = \{\ v \in \mathcal{C}(\Omega)\ :\ v|_T \text{ is linear for } T \in \mathcal{T}_h,\ v = 0 \ \text{ on } \ \Gamma\ \}.$$

Here $v|_T$ denotes the restriction of v to T, i.e., the function defined on T that agrees with v on T. The space $\mathcal{V}_h$ consists of all continuous linear functions on each triangle T that vanish on Γ. Notice that $\mathcal{V}_h \subset \mathcal{V}$.

The functions $v \in \mathcal{V}_h$ are described by the values $v(N_i)$ of v at the *nodes* N_i, $i = 1, 2, \ldots, M$ of $\mathcal{T}_h$, but since $v = 0$ on Γ, the nodes in the boundary are excluded. Therefore the basis functions $\varphi_j \in \mathcal{V}_h$, $j = 1, 2, \ldots, M$ are given for $i, j = 1, 2, \ldots, M$ by

$$\varphi_j(N_i) = \begin{cases} 1, & \text{if } i = j; \\ 0, & \text{if } i \neq j. \end{cases}$$

We see that the *support* of φ_j (the set of points $\mathbf{x}$ for which $\varphi_j(\mathbf{x}) \neq 0$) consists of all the triangles with common node N_j (shaded area in Figure 1.1). A function $v \in \mathcal{V}_h$ now has the representation

$$v(\mathbf{x}) = \sum_{j=1}^{M} \eta_j \varphi_j(\mathbf{x}), \quad \eta_j = v(N_j), \qquad \text{for } \mathbf{x} \in \Omega \cup \Gamma.$$

The finite element method for (1.3) is given by the variational formulation (1.4): find $u_h \in \mathcal{V}_h$ such that

$$a(u_h, v) = \langle f, v \rangle \quad \text{for all } v \in \mathcal{V}_h. \tag{1.5}$$

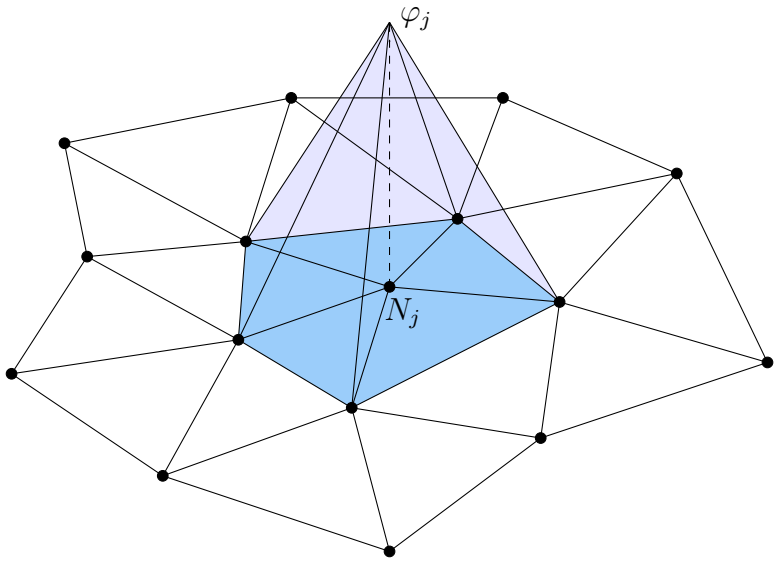


Figure 1.1: Linear basis function φ_j in purple, and support of φ_j in blue.

By applying (1.5) with v being all the basis functions φ_j for $\mathcal{V}_h$, we obtain the linear system of equations

$$\mathbf{A}\boldsymbol{\xi} = \mathbf{f}, \tag{1.6}$$

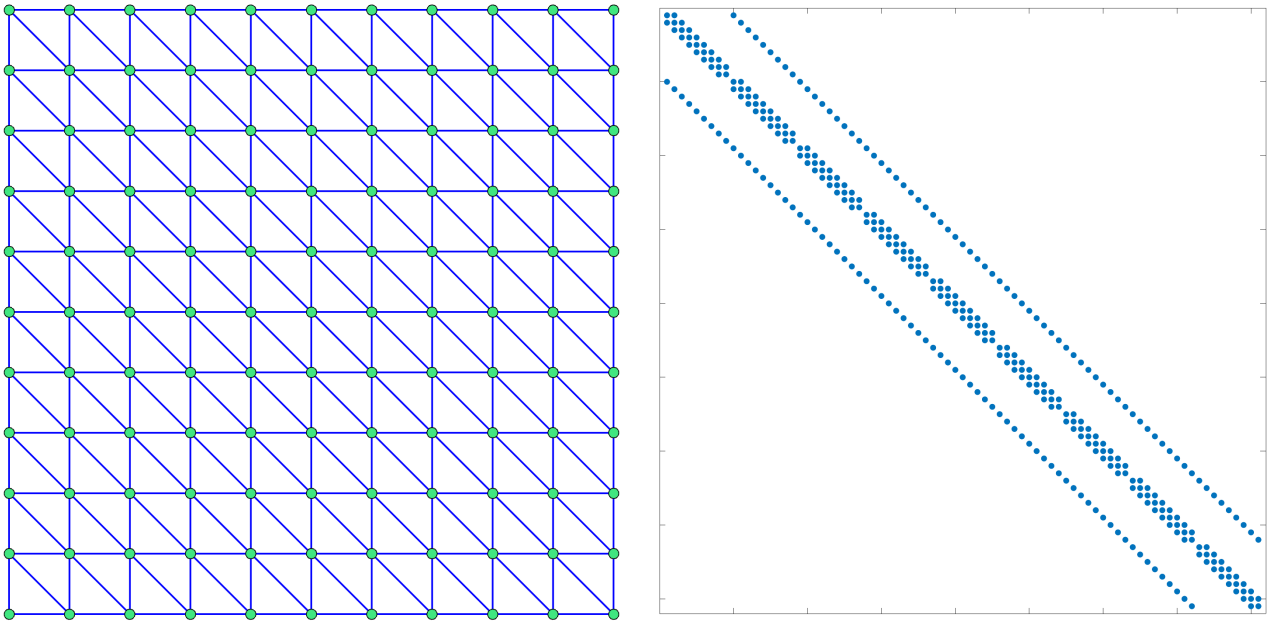
where $\mathbf{A} = (\mathbf{A}_{i,j})$, the stiffness matrix, is an $M \times M$ matrix with elements $\mathbf{A}_{i,j} = a(\varphi_i, \varphi_j)$, $\boldsymbol{\xi} = (\xi_i)$ and $\mathbf{f} = (f_i)$ are vectors in $\mathbb{R}^M$ with elements $\xi_i = u_h(N_i)$ and $f_i = \langle f, \varphi_i \rangle$, respectively.

Clearly, $\mathbf{A}$ is a symmetric matrix. It is also positive definite, and the non-singular. Thus (1.6) admits a unique solution $\boldsymbol{\xi}$. Moreover, $\mathbf{A}$ is sparse; we have $\mathbf{A}_{i,j} = 0$ unless N_i and N_j are nodes of the same triangle. The next example illustrates this structure.

Example 1.1. *Consider the Poisson's equation in two dimensions with homogeneous boundary conditions*

$$\begin{aligned} -\Delta u &= 0 \quad \text{in } \Omega = [0,1] \times [0,1]; \\ u &= 0 \quad \text{on } \Gamma. \end{aligned}$$

To apply the linear finite element method, the grid should be decomposed into triangles. Using a decomposition where the edges are either parallel to the axes or parallel to the line $y = -x$, one obtains a banded matrix with 4 in the diagonal and -1 elsewhere, depending on the way the nodes are labeled.



(a) (b)

Figure 1.2: Example 1.1: (a) sample structured grid; (b) sparsity pattern of $\mathbf{A}$.

For the grid shown in Figure 1.2(a), the nodes were labeled row-by-row, starting in the bottom-left until the top-right, and the elements were labeled in the same way. The stiffness matrix has the pattern shown in Figure 1.2(b), where we see the sparsity pattern described before.

Figure 1.2(a) shows the triangular elements K_j (blue triangles) and the corresponding nodes N_i (green dots). Here we should notice that regular or structured meshes produce nice patterns in the stiffness matrix $\mathbf{A}$, so there are many efficient algorithms for solving the system (1.6). What happens if, rather than linear finite elements, we use higher order finite elements? Or if we use unstructured meshes? Do we still get nice sparsity patterns in the stiffness matrix?

First, we should note that the nice sparsity pattern shown in Figure 1.2(b) comes from a specific labeling of the elements and nodes in the sample structured mesh; for different labeling of the elements, we expect different sparsity patterns, but the number of nonzero entries should be the same.

Example 1.2. *Again consider the problem in Example 1.1, but with an unstructured mesh for* $\Omega = [0,1] \times [0,1]$ *and also let us increase the order of the finite elements.*

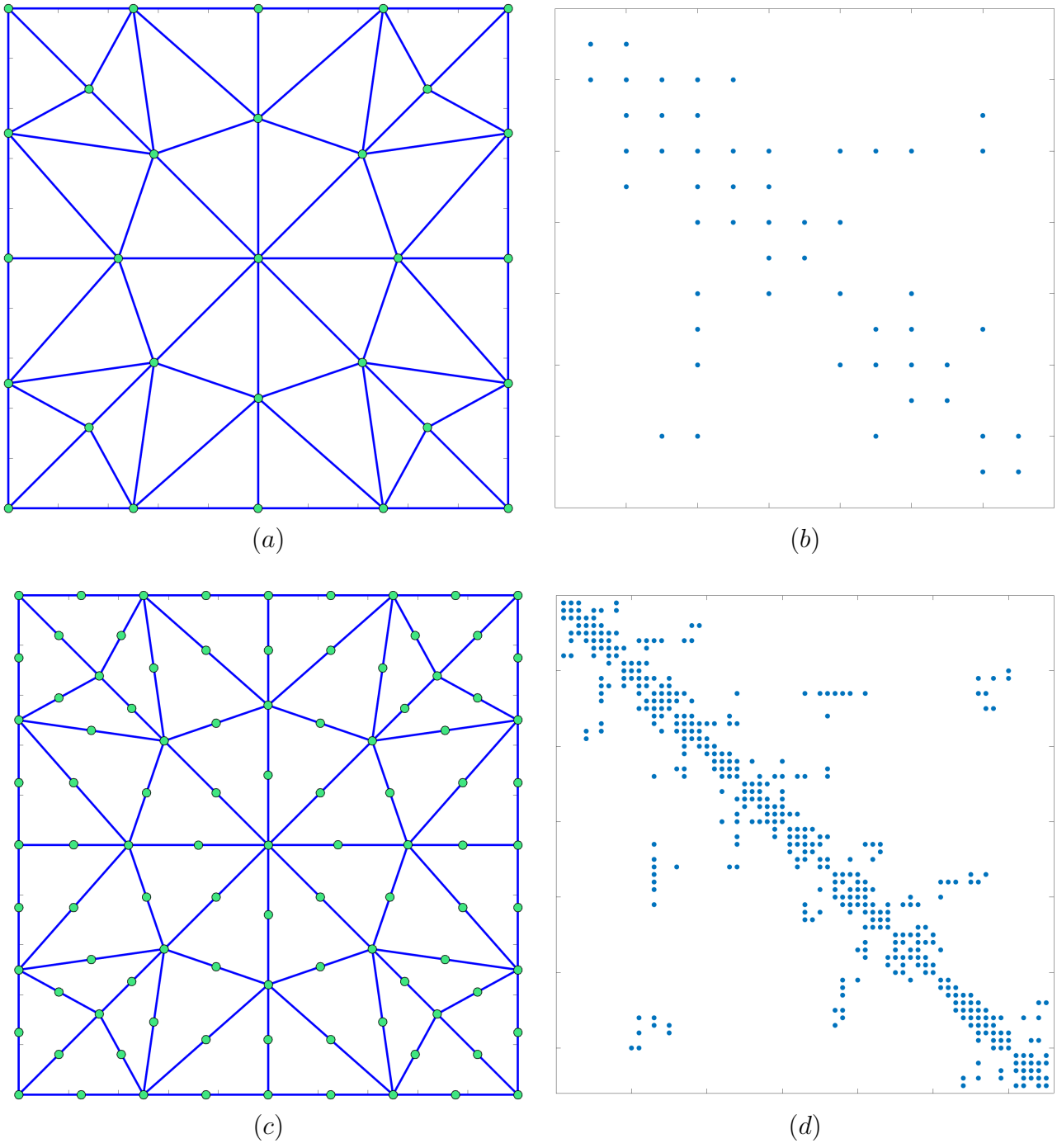
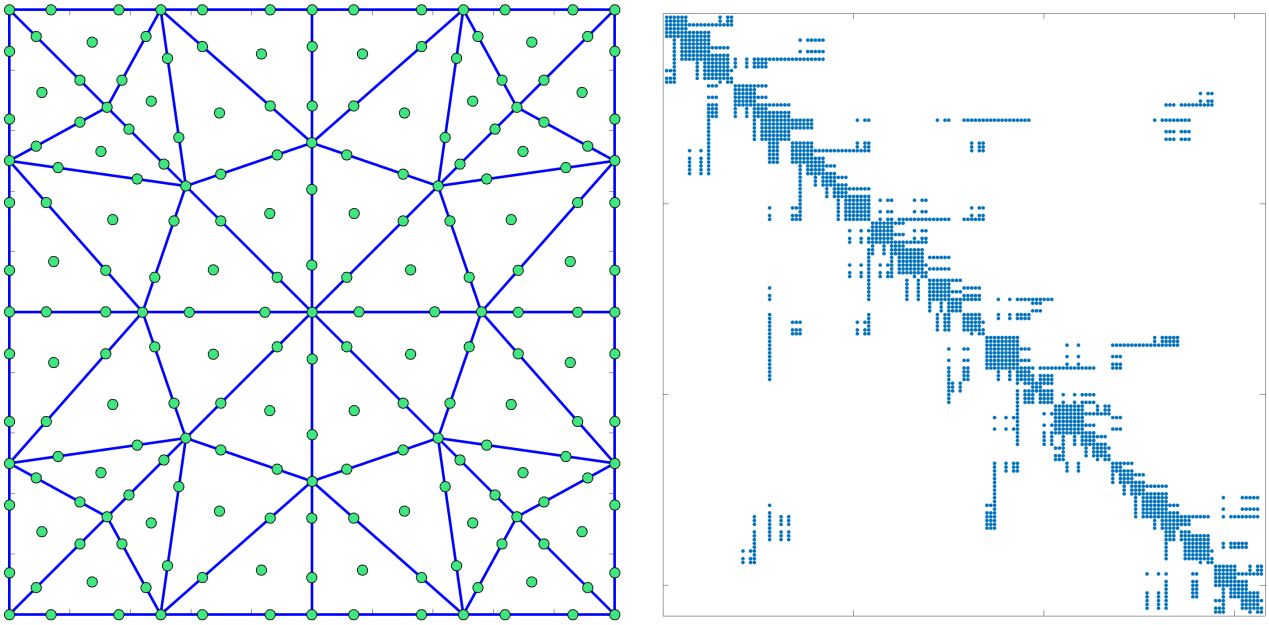


Figure 1.3: Example 1.2: unstructured small mesh (a) linear elements, (c) quadratic elements; sparsity pattern of $\mathbf{A}$ for (b) linear elements, (d) quadratic elements.

Consider the unstructured mesh with linear finite elements shown in Figure 1.3(a), which is a small mesh with 40 elements. After removing the nodes in the boundary, we only get 13 degrees of freedom (DOF), yielding the sparsity pattern shown in Figure 1.3(b). On the other hand, if we use quadratic finite elements, as shown in Figure 1.3(c), the size of the problem increases to 65 DOF after applying the boundary conditions. The sparsity pattern is shown in Figure 1.3(d), where we can see some similarity with Figure 1.3(b).

Finally, using cubic finite elements, as shown in Figure 1.4(a), the size of the problem increases again to 165 DOF after applying the boundary conditions. As we increase the order of the finite elements, we expect better accuracy, but the price is paid in computational effort and storage space, since higher order elements lead to larger and less sparse problems. As we can see in Figure 1.4(b), this large problem will still be quite sparse, so we can use efficient algorithms to overcome the size of the problem.



(a) (b)

Figure 1.4: Example 1.2: (a) unstructured small mesh with cubic finite elements; (b) sparsity pattern of $\mathbf{A}$.

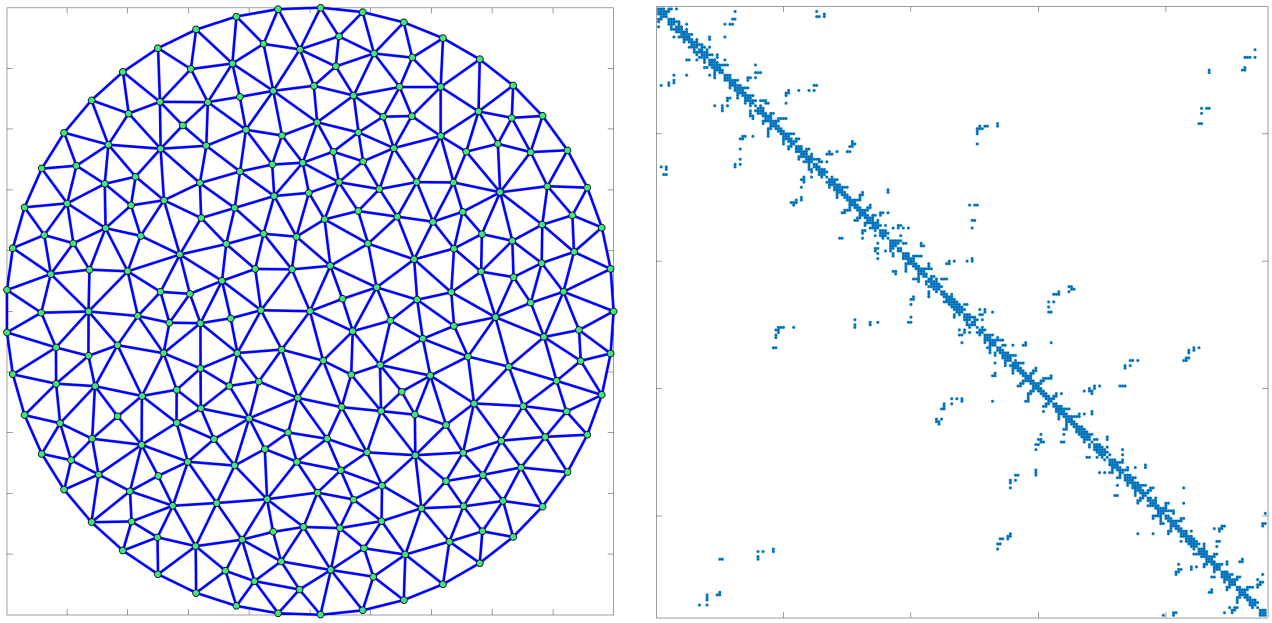
Example 1.2 also shows that unstructured meshes with high order finite elements generate sparse stiffness matrices, but there is not regular pattern to exploit, as in Example 1.1, where techniques like *geometric multigrid* produce good results. The underlying idea in the geometric multigrid approach is to define fine meshes based on the geometric properties of the initial mesh, defining inter-grid transfer operations to go along the different levels.

Another source of difficulties could be the complexity of the domain Ω. To appreciate this, consider Example 1.3, where the domain is the closed disc of radius one centered at the origin.

Example 1.3. *Consider the Poisson's equation in two dimensions with homogeneous boundary conditions*

$$\begin{aligned} -\Delta u &= 0 \quad \text{in } \Omega = \{\, \mathbf{x} \in \mathbb{R}^2 \;:\; \|\mathbf{x}\| \leq 1 \,\}; \\ u &= 0 \quad \text{on } \Gamma. \end{aligned}$$

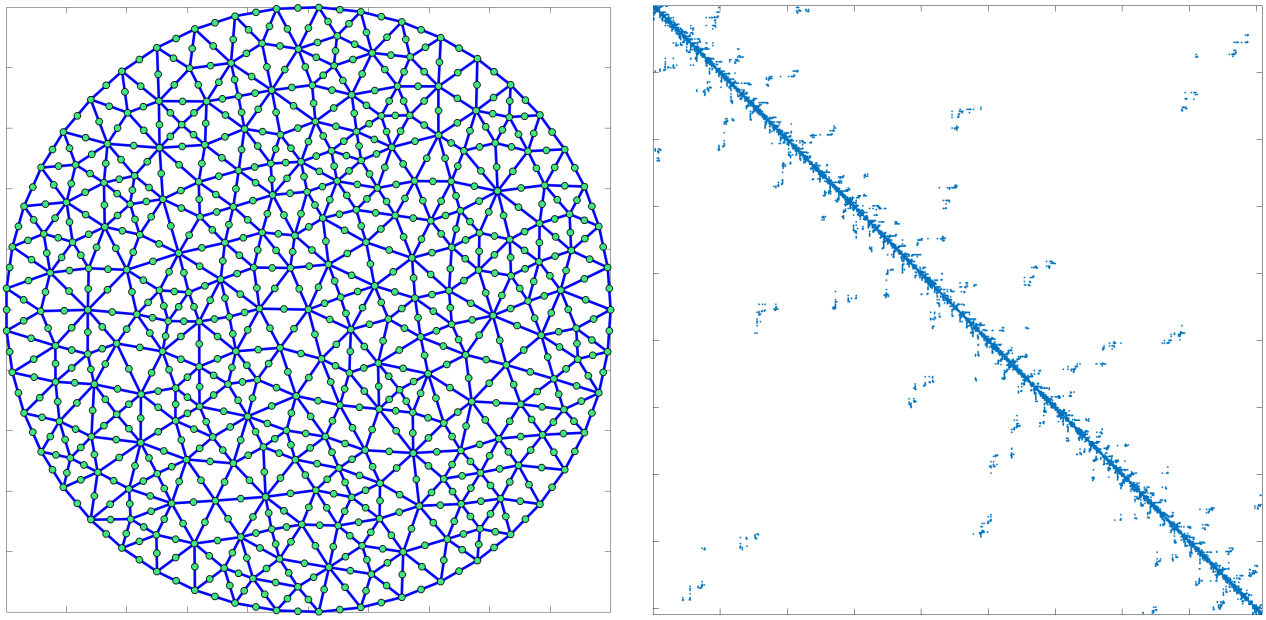
Since the domain Ω is not a polygonal curve, we start by approximating Ω with a triangulation, as shown in Figure 1.5(a). Here, we use 431 elements; after applying the boundary conditions, we get 239 DOF. Figure 1.5(b) shows the sparsity pattern of the corresponding stiffness matrix.



(a) (b)

Figure 1.5: Example 1.3: (a) unstructured circular mesh with linear finite elements; (b) sparsity pattern of $\mathbf{A}$.

Now, considering quadratic finite elements, rather than linear finite elements, as shown in Figure 1.6(a). We get 908 DOF and the sparsity pattern in the stiffness matrix is shown in Figure 1.6(b). Here we should note the similarity with Figure 1.5(b). This fact was also observed in Example 1.2; as we increase the order of the finite element, the pattern observed with linear finite elements spreads out along the new stiffness matrices for higher order finite elements.



(*a*) (*b*)

Figure 1.6: Example 1.3: (*a*) unstructured circular mesh with quadratic finite elements; (*b*) sparsity pattern of $\mathbf{A}$.

As we expect, the stiffness matrices arising from finite element discretization are sparse matrices even if the domain is not a regular shape like a square, but the difficulty arises from the lack of general sparsity patterns for unstructured meshes, as we see in the last examples.

As we said before, for well structured stiffness matrices arising from structured meshes, there are many efficient algorithms like geometric multigrid, which relies on the structured geometric qualities of the mesh. However, for general meshes or for complicated domains we cannot easily apply such methods; thus we use an alternative approach called *algebraic multigrid*. This technique can be seen as a generalization of the geometric multigrid approach, but since the regularity of the mesh is lost it uses alternative approaches to define and transfer between fine and coarse grids.

Note that the effectiveness of algebraic multigrid algorithms depends on the inter-grid transfer operators. Thus we will focus our attention on defining "good" inter-grid transfer operators using aggregation techniques.

1.2 Thesis Overview and Organization

As technology improves, we keep seeking increasingly efficient ways to large-scale problems. Problems arising from natural sciences continue to grow in size and complexity; thus we need faster algorithms to find solutions to these problems. As we saw in Example 1.1, linear systems arising from finite element discretization are sparse in general, and thus iterative methods (like the conjugate gradient algorithm) seem more plausible than direct methods (like Gaussian elimination). Multigrid (MG) methods can solve these large sparse linear systems, but rather than a solver we will show that MG can be used as preconditioner, increasing the effectiveness of the conjugate gradient method.

Chapter 2 introduces the classical iterative methods: Jacobi, Gauss-Seidel, damped Jacobi, conjugate gradients, and preconditioned conjugate gradient. Then we use some of these methods as smoothers in multigrid methods, assuming the existence of the inter-grid prolongation and restriction transfer operators. Here we also introduce the classical multigrid cycles, like the `Two-Grid` cycle, `V`-cycle, `W`-cycle and the `MG`γ cycle. We end this chapter with a discussion about convergence analysis.

Chapter 3 explains how the inter-grid transfer operators are assembled using aggregation techniques. Here we discuss the classical smooth/rough aggregation and the maximal independent set aggregation provided by the `libparanumal`[1] codes. After a comparison between the different aggregation approaches mentioned, we propose an alternative aggregation approach that combines the best features of the previous methods.

Chapter 4 shows the computational results of our experiments. Here we start comparing algebraic multigrid as a solver and preconditioner for the conjugate gradient method. We also show comparisons between the different aggregation algorithms mentioned in the previous chapter. We will use smooth aggregation in a serial implementation for small problems, and rough aggregation in serial/parallel implementations for large-scale problems.

We conclude this work with a discussion about the main results obtained in our experiments and future work.

[1] An experimental set of finite element flow solvers for heterogeneous (GPU/CPU) systems. The initial development of libParanumal was performed by the Parallel Numerical Algorithms Group at Virginia Tech.

1.3 Contributions of this Thesis

The novel contributions of this thesis are provided in Chapter 3, and illustrated in the numerical experiments in Chapter 4. As summary we have:

- A new metric for classifying the strong connections between nodes (Definition 3.5), yielding symmetric strong connectivity matrices;

- A parallel aggregation algorithm (Algorithm 15), based on MIS(2), but rather than aggregating nodes by the relative strong connection between nodes, we aggregate based on strong neighborhoods and weak neighborhoods;

- An improvement in the quality of the aggregates obtained by the MIS(2) coarsener provided by `libparanumal`; our algorithm does not create aggregates consisting of a single node, and it keeps as homogeneous as possible the number of nodes in each aggregate;

- Improvement to the PCG solver provided by `libparanumal`. The MIS(2) coarsener often creates aggregates consisting of a single node. This generates high-energy modes in the inter-grid transfer operators, yielding slow convergence; since our algorithm fixes this issue, we obtain a speedup of about 30% to 50%.

Chapter 2

Multigrid Methods

This chapter will begin with the foundations needed to solve linear systems by iterative methods, then show how these methods can be sped up by preconditioning, and finally describe use of these iterative methods as smoothers in multigrid algorithms. There exist different methods for solving linear systems: direct methods like Gaussian elimination (GEM), LU factorization (LU), Cholesky factorization (CHOL), etc., which provide the exact solution up to machine precision in a finite number of steps; and iterative methods like Richardson (RI), Conjugate Gradient (CG), and Generalized Minimum RESidual (GMRES), which under the right conditions will converge to the solution. See [4], [17], [18], and [19]. We also discuss the multigrid algorithm from a general perspective. For more information, see [3], [6], and [11].

2.1 Iterative methods

Let $\mathbf{A} \in \mathbb{R}^{n\times n}$ and $\mathbf{x}, \mathbf{f} \in \mathbb{R}^n$. Then

$$\mathbf{A}\mathbf{x} = \mathbf{f} \tag{2.1}$$

is a linear system whose exact solution is denoted by $\mathbf{x}$; we denote an approximate solution by $\mathbf{x}_k$. The exact error (or simply `error`) is defined by

$$\mathbf{e} = \mathbf{x} - \mathbf{x}_k. \tag{2.2}$$

Note that $\mathbf{e} \in \mathbb{R}^n$ will be measured by the Euclidean vector norm, but the error is not a convenient measure, since the exact solution is needed to compute it. Thus we use the `residual`, which also quantifies how well $\mathbf{A}\mathbf{x}_k$ approximates $\mathbf{f} = \mathbf{A}\mathbf{x}$, and is defined as

$$\mathbf{r} = \mathbf{f} - \mathbf{A}\mathbf{x}_k. \tag{2.3}$$

We begin by writing $\mathbf{A}$ as the difference of two matrices,

$$\mathbf{A} = \mathbf{M} - \mathbf{N},$$

with $\mathbf{M}$ nonsingular. This is called a *splitting* of $\mathbf{A}$. Obviously there are many ways one could do this, and we will have to distinguish useful splittings for a given $\mathbf{A}$.

From (2.1), we get $\mathbf{Ax} = (\mathbf{M} - \mathbf{N})\mathbf{x} = \mathbf{f}$; therefore, $\mathbf{Mx} = \mathbf{f} + \mathbf{Nx}$, or, equivalently

$$\mathbf{x} = \mathbf{M}^{-1}(\mathbf{f} + \mathbf{Nx}) = \mathbf{M}^{-1}\mathbf{Nx} + \mathbf{M}^{-1}\mathbf{f}.$$

This equation will only be approximately satisfied for some $\mathbf{x}_k \approx \mathbf{x}$:

$$\mathbf{x}_k \approx \mathbf{M}^{-1}\mathbf{N}\mathbf{x}_k + \mathbf{M}^{-1}\mathbf{f}. \tag{2.4}$$

When the initial approximation $\mathbf{x}_0$ is near the exact solution $\mathbf{x}$, we can hope the right hand side of (2.4) provides a more accurate approximation, leading to the iteration:

$$\mathbf{x}_{k+1} = \mathbf{M}^{-1}\mathbf{N}\mathbf{x}_k + \mathbf{M}^{-1}\mathbf{f}. \tag{2.5}$$

Under what conditions will this iteration converge? The error can be introduced by subtracting (2.5) from (2.4):

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

Since $\mathbf{A} = \mathbf{M} - \mathbf{N}$, then $\mathbf{M}^{-1}\mathbf{N} = \mathbf{M}^{-1}(\mathbf{M} - \mathbf{A}) = \mathbf{I} - \mathbf{M}^{-1}\mathbf{A}$, and thus

$$\mathbf{e}_{k+1} = (\mathbf{I} - \mathbf{M}^{-1}\mathbf{A})\mathbf{e}_k.$$

Using this equality recursively, we get

$$\mathbf{e}_k = (\mathbf{I} - \mathbf{M}^{-1}\mathbf{A})^k\mathbf{e}_0. \tag{2.6}$$

To illustrate the basic iterative methods, consider the splitting of $\mathbf{A}$ in the form

$$\mathbf{A} = \mathbf{D} - \mathbf{L} - \mathbf{U},$$

where $\mathbf{D}$ is the diagonal part of $\mathbf{A}$, and $-\mathbf{L}$ and $-\mathbf{U}$ are the strictly lower and upper triangular parts of $\mathbf{A}$, respectively.

For the *Jacobi* method we let $\mathbf{M} = \mathbf{D}$ and $\mathbf{N} = \mathbf{L} + \mathbf{U}$, so (2.5) becomes

$$\mathbf{x}_{k+1} = \mathbf{D}^{-1}\left(\mathbf{L} + \mathbf{U}\right)\mathbf{x}_k + \mathbf{D}^{-1}\mathbf{f}.$$

Thus the Jacobi iteration matrix is given by $\mathbf{T_J} = \mathbf{D}^{-1}(\mathbf{L} + \mathbf{U})$.

The *Gauss-Seidel* method incorporates an important update: the computed components of the new approximation are used immediately. In other words, components of the approximation vector $\mathbf{x}_k$ are updated as soon as they are computed. To express this idea in matrix form, we let $\mathbf{M} = \mathbf{D} - \mathbf{L}$ and $\mathbf{N} = \mathbf{U}$. Thus (2.5) becomes

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

Therefore the Gauss-Seidel iteration matrix is $\mathbf{T_{GS}} = (\mathbf{D} - \mathbf{L})^{-1}\mathbf{U}$.

While the previous iterations may converge under particular conditions, they often are slow. Moreover, they may not be effective smoothing algorithms for multigrid methods. Relaxation techniques can lead to more effective smoothers.

The Jacobi iteration can be enhanced by a scaling factor to yield the *weighted* (or *damped*) Jacobi method. Let $\omega \in \mathbb{R}$, $\omega > 0$, and take $\mathbf{M} = \frac{1}{\omega}\mathbf{D}$ and $\mathbf{N} = \frac{1}{\omega}\mathbf{D} - \mathbf{A}$. Thus (2.5) becomes

$$\mathbf{x}_{k+1} = ((1 - \omega)\mathbf{I} - \omega\mathbf{D}^{-1}(\mathbf{L} + \mathbf{U}))\mathbf{x}_k + \omega\mathbf{D}^{-1}\mathbf{f}.$$

Here ω is called the weighting factor. Note that $\omega = 1$ gives the standard Jacobi method. The weighted Jacobi iteration matrix is $\mathbf{T}_{\omega\mathbf{J}} = (1 - \omega)\mathbf{I} + \omega\mathbf{T_J}$.

Another important method is the Successive Over-Relaxation (SOR) method. This method seeks to reduce rapidly the norm of the residual using a fixed relaxation parameter $0 < \omega < 2$. Let $\mathbf{M} = \frac{1}{\omega}\mathbf{D} - \mathbf{L}$ and $\mathbf{N} = \frac{1}{\omega}\mathbf{D} - (\mathbf{D} - \mathbf{U})$. Thus (2.5) becomes

$$\mathbf{x}_{k+1} = (\mathbf{D} - \omega\mathbf{L})^{-1}(\mathbf{D} - \omega\mathbf{D} + \omega\mathbf{U})\mathbf{x}_k + \omega(\mathbf{D} - \omega\mathbf{L})^{-1}\mathbf{f},$$

and so the SOR iteration matrix is $\mathbf{T_{SOR}} = (\mathbf{D} - \omega\mathbf{L})^{-1}((1 - \omega)\mathbf{D} + \omega\mathbf{U})$.

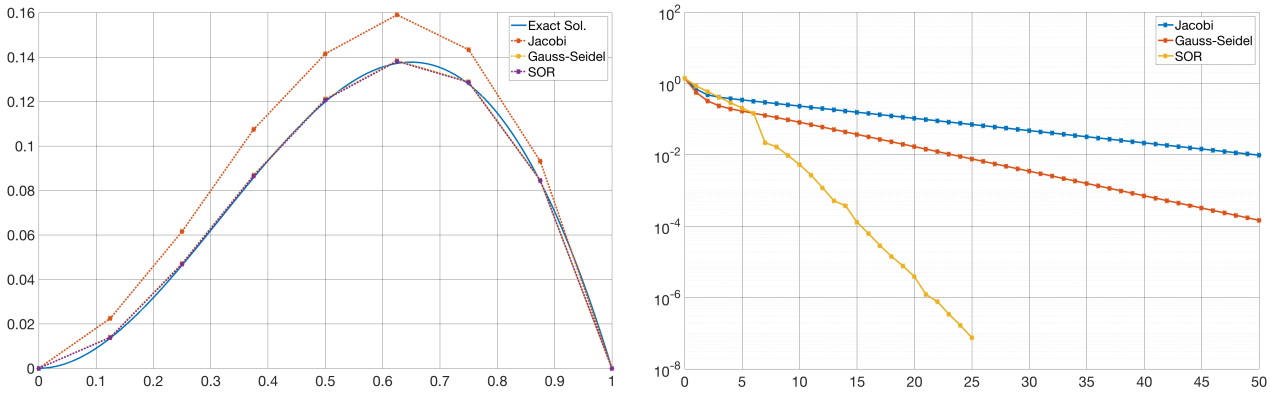
Example 2.1. *Consider the Poisson's equation in 1D with*

$$f(x) = 2(2x - 1)\cos(x) + (2 + x - x^2)\sin(x).$$

For this particular choice of f, the exact solution is given by $u(x) = x(x - 1)\sin(x)$, so we can compare the approximations obtained by Jacobi, Gauss-Seidel and SOR iterations up to $\|\mathbf{r}_k\| \leq Tol = 10^{-7}$ or $n_{max} = 50$.

In this example $\mathbf{A}$ is positive definite and tridiagonal, thus $\omega = 1$ is the optimal damping factor for the damped Jacobi method, and $\omega = 1.4464626922$ is the optimal relaxation factor for the SOR method. For more details about the optimal choice of ω, see [18] and [4].

Figure 2.1(a) shows the approximations obtained after 50 Jacobi, Gauss-Seidel and SOR iterations. Here the SOR iteration converged in 22 iterations, while Jacobi and Gauss-Seidel converged more slowly; after 50 iterations, the Jacobi iteration has a residual of order 10^{-2} and the Gauss-Seidel residual is of order 10^{-3}. Figure 2.1(b) shows the norm of the residual vs. iterations, where it is easy to see which iterative method converges faster up to the given Tol.



(a) (b)

Figure 2.1: Example 2.1: (a) comparison between the exact solution and approximations; (b) comparison between the norms of residuals.

2.1.1 Convergence Analysis

All the methods in the previous section have the form (2.5), i.e.,

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

with $\mathbf{T} = \mathbf{M}^{-1}\mathbf{N}$ called the *iteration matrix*.

Then we can ask the following questions:

1. If the iteration (2.5) converges, does it converge to a solution of the original system?
2. What conditions does iteration (2.5) need for convergence?

If the iteration (2.5) converges to $\mathbf{z}$, then

$$\begin{aligned} \mathbf{z} &= \mathbf{M}^{-1}\mathbf{N}\mathbf{z} + \mathbf{M}^{-1}\mathbf{f} \\ \mathbf{M}\mathbf{z} &= \mathbf{N}\mathbf{z} + \mathbf{f} \\ \mathbf{M}\mathbf{z} - \mathbf{N}\mathbf{z} &= \mathbf{f} \\ \mathbf{A}\mathbf{z} &= \mathbf{f} \end{aligned}$$

Therefore the limit of the iteration (2.5) is a solution of the linear system $\mathbf{Ax} = \mathbf{f}$. Additionally, the algebraic error satisfies (2.6), i.e., $\mathbf{e}_k = (\mathbf{I} - \mathbf{M}^{-1}\mathbf{A})^k\mathbf{e}_0$. Therefore

$$\|\mathbf{e}_k\| = \|(\mathbf{I} - \mathbf{M}^{-1}\mathbf{A})^k\mathbf{e}_0\| \leq \|\mathbf{I} - \mathbf{M}^{-1}\mathbf{A}\|^k\|\mathbf{e}_0\|.$$

Thus a sufficient condition for the iteration (2.5) to be convergent is

$$\|\mathbf{I} - \mathbf{M}^{-1}\mathbf{A}\| = \|\mathbf{M}^{-1}\mathbf{N}\| < 1.$$

Note $\|\mathbf{A}\| < 1$ is sufficient but not necessary for $\mathbf{A}^k \to 0$ as $k \to \infty$. However, one can say something a bit sharper. Recall that the spectral radius $\rho(\mathbf{A})$ satisfies

$$\lim_{k\to\infty} \|\mathbf{A}^k\| = 0 \quad \Longleftrightarrow \quad \rho(\mathbf{A}) < 1,$$

as stated on page 189 in [19]. Thus, we state the next theorem.

Theorem 2.1. *Suppose* $\mathbf{M}^{-1}\mathbf{N}$ *is a square matrix with* $\rho(\mathbf{M}^{-1}\mathbf{N}) < 1$. *Then* $\mathbf{I} - \mathbf{M}^{-1}\mathbf{N}$ *is nonsingular, and the iteration (2.5) converges for any* $\mathbf{f}$ *and* $\mathbf{x}_0$.

It is not always convenient or even possible to calculate the eigenvalues of $\mathbf{M}^{-1}\mathbf{N}$ in our efforts to get a handle on $\rho(\mathbf{M}^{-1}\mathbf{N})$. Sometimes, it may be easier to calculate $\|\mathbf{M}^{-1}\mathbf{N}\|$.

For any eigenvalue λ of $\mathbf{M}^{-1}\mathbf{N}$, there exists some $\mathbf{v} \neq \mathbf{0}$ such that $\mathbf{M}^{-1}\mathbf{Nv} = \lambda\mathbf{v}$. Thus

$$|\lambda|\|\mathbf{v}\| = \|\lambda\mathbf{v}\| = \|\mathbf{M}^{-1}\mathbf{Nv}\| \leq \|\mathbf{M}^{-1}\mathbf{N}\|\|\mathbf{v}\|.$$

This means $|\lambda| \leq \|\mathbf{M}^{-1}\mathbf{N}\|$ for all eigenvalues λ of $\mathbf{M}^{-1}\mathbf{N}$, so maximizing over λ we get

$$\rho(\mathbf{M}^{-1}\mathbf{N}) \leq \|\mathbf{M}^{-1}\mathbf{N}\|.$$

Thus we impose the condition $\|\mathbf{M}^{-1}\mathbf{N}\| < 1$ to estimate the rate at which the iteration converges.

Remark 2.1. *Although the condition* $\rho(\mathbf{M}^{-1}\mathbf{N}) < 1$ *is both necessary and sufficient for eventual convergence of the error to zero, it is not sufficient to guarantee monotone convergence of the error. In particular, the error could actually increase for a time in the course of the iteration before eventually diminishing to zero.*

The *numerical range* of $\mathbf{A} \in \mathbb{R}^{n\times n}$ is a set defined by

$$W(\mathbf{A}) = \{\ \mathbf{x}^T\mathbf{Ax}\ :\ \|\mathbf{x}\| = 1\ \}.$$

This is an important set that provides information about $\mathbf{A}$ as a transformation, its eigenvalues, and its departure from normality. Thus it can help us to get a better understanding about convergence of the previous methods.

Example 2.2. *Consider Example 2.1, where we saw that the SOR iteration converges within 50 iterations, while Jacobi and Gauss-Seidel iterations did not.*

Figure 2.2 shows the numerical range of each one of the iteration matrices with its respective eigenvalues. Here, we can see why the SOR iteration converged faster, since

$$\rho(\mathbf{T_{SOR}}) < \rho(\mathbf{T_{GS}}) < \rho(\mathbf{T_J}) < 1.$$

The SOR iteration converged within 50 iterations, while the Jacobi iteration and Gauss-Seidel took much longer, as was shown in Figure 2.1(b).

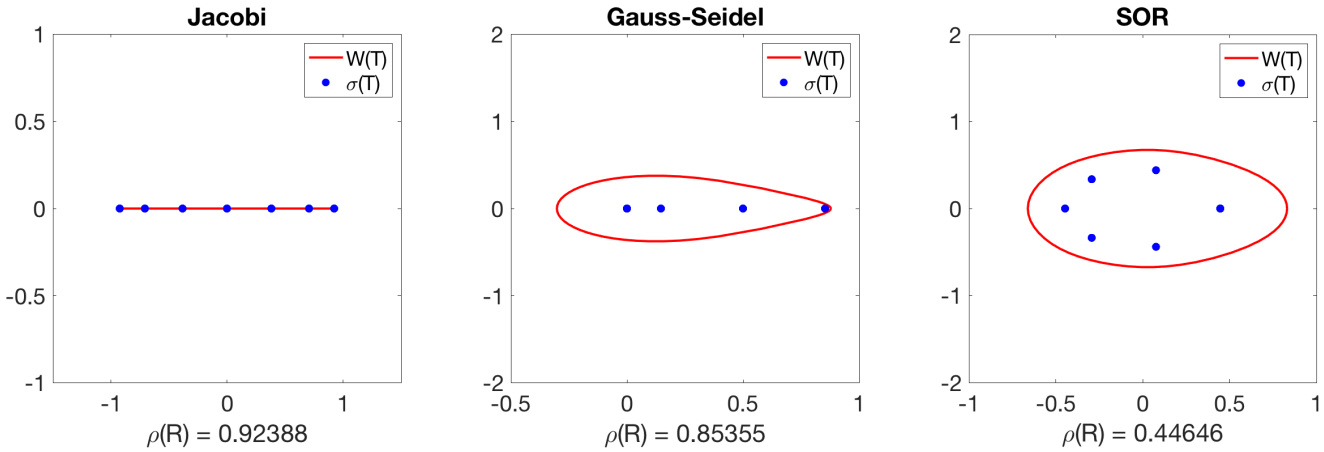


Figure 2.2: Example 2.1: numerical range $W(\mathbf{R})$ and eigenvalues $\sigma(\mathbf{R})$ for each iteration matrix $\mathbf{R}$.

2.1.2 The Conjugate Gradient method

The *Conjugate Gradient* (CG) method is another iterative algorithm for solving $\mathbf{Ax} = \mathbf{f}$, provided $\mathbf{A}$ is a symmetric positive definite matrix. In this case, the linear system's solution also satisfies the optimization problem

$$\min_{\mathbf{x}} \phi(\mathbf{x}), \qquad \text{where} \qquad \phi(\mathbf{x}) = \mathbf{f}^T\mathbf{x} - \frac{1}{2}\mathbf{x}^T\mathbf{Ax}. \tag{2.7}$$

Note the residual of the linear system equals the gradient of ϕ, that is

$$\nabla\phi(\mathbf{x}) = \mathbf{f} - \mathbf{Ax} = \mathbf{r};$$

therefore, when $\mathbf{x} = \mathbf{x}_k$ we have $\mathbf{r}_k = \mathbf{f} - \mathbf{Ax}_k$.

Definition 2.1 (Conjugate vectors)**.** *A set of nonzero vectors* $\{\mathbf{p}_0, \mathbf{p}_1, \ldots, \mathbf{p}_k\}$ *is* ***conjugate*** *with respect to the symmetric positive matrix* $\mathbf{A}$ *if* $\mathbf{p}_i^T \mathbf{A} \mathbf{p}_j = 0$ *for all* $i \neq j$*.*

Conjugacy is important, since we can minimize ϕ defined by (2.7) in at most n steps by minimizing it along each direction in the conjugate set.

Given $\mathbf{x}_0 \in \mathbb{R}^n$ and the set of conjugate directions $\{\mathbf{p}_0, \mathbf{p}_1, \ldots, \mathbf{p}_k\}$, a sequence $\{x_k\}$ is generated by

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k, \tag{2.8}$$

where α_k minimizes the one-dimensional quadratic function ϕ over iterates of the form $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha \mathbf{p}_k$, given explicitly by

$$\alpha_k = \frac{\mathbf{r}_k^T \mathbf{p}_k}{\mathbf{p}_k^T \mathbf{A} \mathbf{p}_k}.$$

In the CG method, each direction $\mathbf{p}_k$ is chosen to be a linear combination of the residual $\mathbf{r}_k$ and the previous direction $\mathbf{p}_{k-1}$. Therefore

$$\mathbf{p}_k = \mathbf{r}_k + \beta_k \mathbf{p}_{k-1},$$

and the scalar β_k is chosen so that $\mathbf{p}_{k-1}$ and $\mathbf{p}_k$ are conjugate with respect to $\mathbf{A}$. This implies

$$\beta_k = \frac{\mathbf{r}_k^T \mathbf{A} \mathbf{p}_{k-1}}{\mathbf{p}_{k-1}^T \mathbf{A} \mathbf{p}_{k-1}}.$$

Finally, the steepest descent direction at $\mathbf{x}_0$ is taken as the first search direction $\mathbf{p}_0$. The complete algorithm is shown in Algorithm 1.

Algorithm 1: Conjugate Gradient iteration (Preliminary Version).

input : $\mathbf{x}_0$, Tol, n_{max}
output: $\mathbf{x}_k$

$\mathbf{r}_0 \leftarrow \mathbf{f} - \mathbf{A}\mathbf{x}_0$;
$\mathbf{p}_0 \leftarrow \mathbf{r}_0$;
$k \leftarrow 0$;
while $\|\mathbf{r}_k\| \geq Tol$ **and** $k \leq n_{max}$ **do**
  $\alpha_k \leftarrow \dfrac{\mathbf{r}_k^T \mathbf{p}_k}{\mathbf{p}_k^T \mathbf{A} \mathbf{p}_k}$;
  $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \alpha_k \mathbf{p}_k$;
  $\mathbf{r}_{k+1} \leftarrow \mathbf{f} - \mathbf{A}\mathbf{x}_{k+1}$;
  $\beta_{k+1} \leftarrow \dfrac{\mathbf{r}_{k+1}^T \mathbf{A} \mathbf{p}_k}{\mathbf{p}_k^T \mathbf{A} \mathbf{p}_k}$;
  $\mathbf{p}_{k+1} \leftarrow \mathbf{r}_{k+1} + \beta_{k+1} \mathbf{p}_k$;
  $k \leftarrow k + 1$;

Note that by pre-multiplying (2.8) by $\mathbf{A}$ and from the definition of the residual $\mathbf{r}_k$ we get

$$\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k \mathbf{A}\mathbf{p}_k. \tag{2.9}$$

Then, Algorithm 1 can be improved (more economical) using (2.9) and by noticing that

$$\mathbf{r}_k^T \mathbf{r}_i = 0, \quad \text{for} \quad i = 0, 1, 2, \ldots, k-1.$$

Therefore α_k and β_k can be computed via

$$\alpha_k = \frac{\mathbf{r}_k^T \mathbf{r}_k}{\mathbf{p}_k^T \mathbf{A}\mathbf{p}_k} \quad \text{and} \quad \beta_{k+1} = \frac{\mathbf{r}_{k+1}^T \mathbf{r}_{k+1}}{\mathbf{r}_k^T \mathbf{r}_k}.$$

By using these formulae together with Algorithm 1, the standard form of the conjugate gradient method is obtained as shown in Algorithm 2.

Algorithm 2: Conjugate Gradient iteration (`CG`).

input : $\mathbf{x}_0$, Tol, n_{max}
output: $\mathbf{x}_k$

$\mathbf{r}_0 \leftarrow \mathbf{f} - \mathbf{A}\mathbf{x}_0$;
$\mathbf{p}_0 \leftarrow \mathbf{r}_0$;
$k \leftarrow 0$;
while $\|\mathbf{r}_k\| \geq Tol$ **and** $k \neq n_{max}$ **do**
  $\alpha_k \leftarrow \dfrac{\mathbf{r}_k^T \mathbf{r}_k}{\mathbf{p}_k^T \mathbf{A}\mathbf{p}_k}$;
  $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \alpha_k \mathbf{p}_k$;
  $\mathbf{r}_{k+1} \leftarrow \mathbf{r}_k - \alpha_k \mathbf{A}\mathbf{p}_k$;
  $\beta_{k+1} \leftarrow \dfrac{\mathbf{r}_{k+1}^T \mathbf{r}_{k+1}}{\mathbf{r}_k^T \mathbf{r}_k}$;
  $\mathbf{p}_{k+1} \leftarrow \mathbf{r}_{k+1} + \beta_{k+1} \mathbf{p}_k$;
  $k \leftarrow k + 1$;

Define

$$\mathcal{P}_n = \{ \text{ polynomial } p \text{ of degree} \leq n \text{ with } p(0) = 1 \}.$$

Then for CG, the appropriate approximation problem involves the $\mathbf{A}$-norm of the error.

CG Approximation Problem: Find $p^* \in \mathcal{P}_n$ such that

$$\|p(\mathbf{A})\mathbf{e}_0\|_{\mathbf{A}} = \text{minimum}, \tag{2.10}$$

where $\mathbf{e}_0$ is the initial error $\mathbf{e}_0 = \mathbf{x} - \mathbf{x}_0$.

Theorem 2.2. *If the CG iteration has not converged before step n, i.e., $\mathbf{r}_{n-1} \neq \mathbf{0}$, then (2.10) has a unique solution $p \in \mathcal{P}_n$, and the iterate $\mathbf{x}_n$ has error $\mathbf{e}_n = p(\mathbf{A})\mathbf{e}_0$. Consequently*

$$\frac{\|\mathbf{e}_n\|_{\mathbf{A}}}{\|\mathbf{e}_0\|_{\mathbf{A}}} = \inf_{p \in \mathcal{P}_n} \max_{\lambda \in \sigma(\mathbf{A})} |p(\lambda)|,$$

where $\sigma(\mathbf{A})$ denotes the spectrum of $\mathbf{A}$.

Proof. This proof can be found on page 299 in [19]. ∎

This theorem establishes the dependence between the rate of convergence of the CG algorithm and the spectrum of $\mathbf{A}$. A good spectrum is one on which polynomials $p \in \mathcal{P}_n$ can be very small, with size decreasing rapidly with n. Roughly speaking, this may happen for either or both of two reasons: eigenvalues could be agglomerated in small regions, or they could be apart in the spectrum, relatively far from the origin.

Assume that the eigenvalues of $\mathbf{A}$ are perfectly clustered in small regions, without any restriction on their location in the spectrum of $\mathbf{A}$.

Theorem 2.3. *If $\mathbf{A}$ has m distinct eigenvalues, then the CG iteration will converge in at most m steps.*

Proof. This proof can be found on page 299 in [19]. ∎

In contrast, suppose we ignore potential clustering of the eigenvalues, but we know that their distance to the origin differs at most by a factor $\kappa \geq 1$, or, equivalently, if we have only the condition number

$$\kappa = \frac{\lambda_{max}}{\lambda_{min}},$$

where λ_{max} and λ_{min} are the extreme eigenvalues of $\mathbf{A}$ $(0 < \lambda_{min} < \lambda_{max})$.

Theorem 2.4. *Applying the CG method to the problem $\mathbf{Ax} = \mathbf{f}$, where $\mathbf{A}$ is symmetric positive definite and has condition number κ. Then the $\mathbf{A}$-norms of the errors satisfy*

$$\frac{\|\mathbf{e}_n\|_{\mathbf{A}}}{\|\mathbf{e}_0\|_{\mathbf{A}}} \leq \frac{2}{\left(\frac{\sqrt{\kappa}+1}{\sqrt{\kappa}-1}\right)^n + \left(\frac{\sqrt{\kappa}+1}{\sqrt{\kappa}-1}\right)^{-n}} \leq 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^n. \tag{2.11}$$

Proof. This proof can be found on page 300 in [19]. ∎

2.2 Preconditioned Iterative Methods

Recall the Jacobi, damped Jacobi, Gauss-Seidel, and SOR iterations take the form

$$\mathbf{x}_{k+1} = \mathbf{T}\mathbf{x}_k + \mathbf{b}, \tag{2.12}$$

where $\mathbf{T}$ can be $\mathbf{T_J}, \mathbf{T}_{\omega\mathbf{J}}, \mathbf{T_{GS}}$, and $\mathbf{T_{SOR}}$, respectively.

Consider the matrix splitting $\mathbf{A} = \mathbf{M} - \mathbf{N}$, where $\mathbf{A}$ is the coefficient matrix proceeding from a linear system $\mathbf{Ax} = \mathbf{f}$. Note that recurrence (2.5) defines a linear fixed-point iteration, i.e.,

$$\mathbf{x}_{k+1} = \mathbf{M}^{-1}\mathbf{N}\mathbf{x}_k + \mathbf{M}^{-1}\mathbf{f},$$

which has the form (2.12) with $\mathbf{T} = \mathbf{M}^{-1}\mathbf{N}$ and $\mathbf{b} = \mathbf{M}^{-1}\mathbf{f}$. Thus we can use (2.5) for solving the system

$$(\mathbf{I} - \mathbf{R})\mathbf{x} = \mathbf{b}, \tag{2.13}$$

but notice that $\mathbf{T} = \mathbf{M}^{-1}\mathbf{N} = \mathbf{M}^{-1}(\mathbf{M} - \mathbf{A}) = \mathbf{I} - \mathbf{M}^{-1}\mathbf{A}$, so $\mathbf{I} - \mathbf{T} = \mathbf{M}^{-1}\mathbf{A}$. Thus (2.13) can be rewritten as $\mathbf{M}^{-1}\mathbf{Ax} = \mathbf{M}^{-1}\mathbf{f}$, or, equivalently,

$$\widehat{\mathbf{A}}\mathbf{x} = \widehat{\mathbf{f}} \quad \text{where} \quad \widehat{\mathbf{A}} = \mathbf{M}^{-1}\mathbf{A},\ \widehat{\mathbf{f}} = \mathbf{M}^{-1}\mathbf{f}.$$

The system $\widehat{\mathbf{A}}\mathbf{x} = \widehat{\mathbf{f}}$ has the same solution as $\mathbf{Ax} = \mathbf{f}$, and is called the *preconditioned system*. $\mathbf{M}$ is called the *preconditioner*. This means a fixed-point iteration on a preconditioned system is equivalent to a relaxation scheme.

The preconditioner $\mathbf{M}$ should be chosen so that the matrix $\widehat{\mathbf{A}}$ is better conditioned for the CG method.

Example 2.3. *The Jacobi, damped Jacobi, Gauss-Seidel and SOR iterations give the preconditioning matrices*

$$\mathbf{M_J} = \mathbf{D}, \quad \mathbf{M}_{\omega\mathbf{J}} = \frac{1}{\omega}\mathbf{D}, \quad \mathbf{M_{GS}} = \mathbf{D} - \mathbf{L}, \quad \mathbf{M_{SOR}} = \frac{1}{\omega}(\mathbf{D} - \omega\mathbf{L}).$$

Iterative solvers have a well known weakness, a lack of robustness compared to direct solvers, but robustness and efficiency of iterative techniques usually are improved by *preconditioners*. To begin with, we should consider and evaluate the possible options for preconditioning a system. Then, we should obtain a preconditioner $\mathbf{M}$. There is not a specific way for defining $\mathbf{M}$, but it must satisfy at least the following requirements.

- The primarily requirement is that solving $\mathbf{Mx} = \mathbf{f}$ must be cheap, since such a linear system will be solved at each step of the preconditioned algorithms;

- $\mathbf{M}$ should clearly be nonsingular and closely related to $\mathbf{A}$ in some sense.

Once a preconditioner $\mathbf{M}$ is identified, there are three well known ways to use it. It can be applied to the left:

$$\mathbf{M}^{-1}\mathbf{A} = \mathbf{M}^{-1}\mathbf{f}.$$

Conversely, $\mathbf{M}$ can be applied to the right:

$$\mathbf{A}\mathbf{M}^{-1}\mathbf{y} = \mathbf{f}, \quad \mathbf{x} = \mathbf{M}^{-1}\mathbf{y}.$$

The above formulation makes the change of variables $\mathbf{y} = \mathbf{M}\mathbf{x}$, then solving the system with respect to $\mathbf{y}$. Finally, another possible situation is when the preconditioner is present in a factored form,

$$\mathbf{M} = \mathbf{M}_\mathbf{L}\mathbf{M}_\mathbf{U},$$

where, typically $\mathbf{M}_\mathbf{L}$ and $\mathbf{M}_\mathbf{U}$ are triangular matrices. In this case, the preconditioning can be split

$$\mathbf{M}_\mathbf{L}^{-1}\mathbf{A}\mathbf{M}_\mathbf{U}^{-1}\mathbf{y} = \mathbf{M}_\mathbf{L}^{-1}\mathbf{f}, \quad \mathbf{x} = \mathbf{M}_\mathbf{U}^{-1}\mathbf{y}.$$

When we compute $\mathbf{M}$ as an incomplete Cholesky factorization, i.e., we have $\mathbf{M} = \mathbf{L}\mathbf{L}^T$, then we use the "splitting", which yields the system

$$\mathbf{L}^{-1}\mathbf{A}\mathbf{L}^{-T}\mathbf{y} = \mathbf{L}^{-1}\mathbf{f}, \quad \mathbf{x} = \mathbf{L}^{-T}\mathbf{y}.$$

2.2.1 The Preconditioned Conjugate Gradient method

Even if $\mathbf{A}$ is symmetric, the left preconditioned matrix $\widehat{\mathbf{A}} = \mathbf{M}^{-1}\mathbf{A}$ will typically not be symmetric. To preserve symmetry in the preconditioned system, $\widehat{\mathbf{A}}\mathbf{x} = \widehat{\mathbf{f}}$, use $\mathbf{M} = \mathbf{L}\mathbf{L}^T$ in the form of an incomplete Cholesky factorization with $\mathbf{L}$ nonsingular. Then, $\mathbf{M}^{-1} = \mathbf{L}^{-T}\mathbf{L}^{-1}$, so we can write $\widehat{\mathbf{A}}\mathbf{x} = \widehat{\mathbf{f}}$ as follows

$$\begin{aligned} \mathbf{M}^{-1}\mathbf{A}\mathbf{x} &= \mathbf{M}^{-1}\mathbf{f} \\ \mathbf{L}^{-T}\mathbf{L}^{-1}\mathbf{A}\mathbf{x} &= \mathbf{L}^{-T}\mathbf{L}^{-1}\mathbf{f} \\ \mathbf{L}^{-1}\mathbf{A}\mathbf{x} &= \mathbf{L}^{-1}\mathbf{f} \\ \mathbf{L}^{-1}\mathbf{A}\mathbf{L}^{-T}\mathbf{L}^{T}\mathbf{x} &= \mathbf{L}^{-1}\mathbf{f} \end{aligned}$$

Let $\widetilde{\mathbf{A}} = \mathbf{L}^{-1}\mathbf{A}\mathbf{L}^{-T}$, $\widetilde{\mathbf{x}} = \mathbf{L}^T\mathbf{x}$ and $\widetilde{\mathbf{f}} = \mathbf{L}^{-1}\mathbf{f}$, then

$$\mathbf{A}\mathbf{x} = \mathbf{f} \quad \Longleftrightarrow \quad \widehat{\mathbf{A}}\mathbf{x} = \widehat{\mathbf{f}} \quad \Longleftrightarrow \quad \widetilde{\mathbf{A}}\widetilde{\mathbf{x}} = \widetilde{\mathbf{f}},$$

but in the last system, we need to compute $\mathbf{x} = \mathbf{L}\widetilde{\mathbf{x}}$.

We can apply the standard CG method to $\widetilde{\mathbf{A}}\widetilde{\mathbf{x}} = \widetilde{\mathbf{f}}$. However, it is more efficient if we incorporate the preconditioner directly into the method. To explain this idea, we should examine

each step in the CG method.

The initial guess and first residual become $\widetilde{\mathbf{x}}_0 = \mathbf{L}^{-1}\mathbf{x}_0$ and $\widetilde{\mathbf{r}}_0 = \mathbf{L}^T\mathbf{r}_0$; thus we start by looking at the new residuals $\widetilde{\mathbf{r}}_k$ which yield $\widetilde{\mathbf{r}}_k = \mathbf{L}^T\mathbf{r}_k$. The initial search direction $\widetilde{\mathbf{p}}_0 = \widetilde{\mathbf{r}}_0$, yields $\mathbf{p}_0 = \mathbf{M}^{-1}\mathbf{r}_0$. Then define the following abbreviations

$$\widetilde{\mathbf{p}}_k = \mathbf{L}^{-1}\mathbf{p}_k \quad \text{and} \quad \mathbf{z}_k = \mathbf{M}^{-1}\mathbf{r}_k.$$

The step length $\widetilde{\alpha}_k$ transforms to

$$\widetilde{\alpha}_k = \frac{\mathbf{r}_k^T\mathbf{z}_k}{\mathbf{p}_k^T\mathbf{A}\mathbf{p}_k}.$$

The approximate solution is updated according to $\widetilde{\mathbf{x}}_k = \widetilde{\mathbf{x}}_{k-1} + \widetilde{\alpha}_k\widetilde{\mathbf{p}}_{k-1}$, yielding

$$\mathbf{x}_k = \mathbf{x}_{k-1} + \widetilde{\alpha}_{k-1}\mathbf{p}_{k-1}.$$

The residuals are updated as $\widetilde{\mathbf{r}}_k = \widetilde{\mathbf{r}}_{k-1} - \widetilde{\alpha}_{k-1}\widetilde{\mathbf{A}}\widetilde{\mathbf{p}}_{k-1}$, yielding

$$\mathbf{r}_k = \mathbf{r}_{k-1} - \widetilde{\alpha}_{k-1}\mathbf{A}\mathbf{p}_{k-1}.$$

The gradient correction factor β_k transforms to

$$\widetilde{\beta}_k = \frac{\mathbf{r}_k^T\mathbf{z}_k}{\mathbf{r}_{k-1}^T\mathbf{z}_{k-1}}.$$

Finally, the new search direction $\widetilde{\mathbf{p}}_k = \widetilde{\mathbf{r}}_k + \widetilde{\beta}_{k-1}\widetilde{\mathbf{p}}_{k-1}$ yields

$$\mathbf{p}_k = \mathbf{z}_k + \widetilde{\beta}_{k-1}\mathbf{p}_{k-1}.$$

It is not mandatory to have $\mathbf{M}$ as an incomplete Cholesky factorization ($\mathbf{M} = \mathbf{L}\mathbf{L}^T$) to conserve symmetry. Note that $\mathbf{M}^{-1}\mathbf{A}$ is selfadjoint in the $\mathbf{M}$-inner product,

$$\langle \mathbf{x}, \mathbf{y}\rangle_{\mathbf{M}} \equiv \langle \mathbf{M}\mathbf{x}, \mathbf{y}\rangle = \langle \mathbf{x}, \mathbf{M}\mathbf{y}\rangle,$$

where $\langle \cdot, \cdot \rangle$ denotes the standard Euclidean inner product. Since

$$\langle \mathbf{M}^{-1}\mathbf{A}\mathbf{x}, \mathbf{y}\rangle_{\mathbf{M}} = \langle \mathbf{A}\mathbf{x}, \mathbf{y}\rangle = \langle \mathbf{x}, \mathbf{A}\mathbf{y}\rangle = \langle \mathbf{x}, \mathbf{M}\mathbf{M}^{-1}\mathbf{A}\mathbf{y}\rangle = \langle \mathbf{x}, \mathbf{M}^{-1}\mathbf{A}\mathbf{y}\rangle_{\mathbf{M}}.$$

Therefore, the Euclidean inner products inside the CG method can be replaced by $\mathbf{M}$-inner products.

If the CG method is rewritten using the $\mathbf{M}$-inner product, we can use the original residual $\mathbf{r}_k = \mathbf{f} - \mathbf{A}\mathbf{x}_k$ and the residual for the preconditioned system $\mathbf{z}_k = \mathbf{M}^{-1}\mathbf{r}_k$. Then, ignoring the initial step, we get the following steps:

1. $\alpha_k = \dfrac{\langle \mathbf{z}_k, \mathbf{z}_k \rangle_{\mathbf{M}}}{\langle \mathbf{M}^{-1}\mathbf{A}\mathbf{p}_k, \mathbf{p}_k \rangle_{\mathbf{M}}}$,
2. $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$,
3. $\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k \mathbf{A}\mathbf{p}_k$,
4. $\mathbf{z}_{k+1} = \mathbf{M}^{-1}\mathbf{r}_{k+1}$,
5. $\beta_k = \dfrac{\langle \mathbf{z}_{k+1}, \mathbf{z}_{k+1} \rangle_{\mathbf{M}}}{\langle \mathbf{z}_k, \mathbf{z}_k \rangle_{\mathbf{M}}}$,
6. $\mathbf{p}_{k+1} = \mathbf{z}_{k+1} + \beta_k \mathbf{p}_k$.

Since

$$\langle \mathbf{z}_k, \mathbf{z}_k \rangle_{\mathbf{M}} = \langle \mathbf{r}_k, \mathbf{z}_k \rangle = \mathbf{z}_k^T \mathbf{r}_k \quad \text{and} \quad \langle \mathbf{M}^{-1}\mathbf{A}\mathbf{p}_k, \mathbf{p}_k \rangle_{\mathbf{M}} = \langle \mathbf{A}\mathbf{p}_k, \mathbf{p}_k \rangle = \mathbf{p}_k^T \mathbf{A}\mathbf{p}_k,$$

then $\mathbf{M}$-inner products are not computed explicitly and so the same algorithm is obtained by both approaches. The PCG iteration is given by Algorithm 3.

Algorithm 3: Preconditioned Conjugate Gradient (`PCG`).

input : $\mathbf{A}$, $\mathbf{f}$, $\mathbf{x}_0$, Tol, N_{max}
output: $\mathbf{x}_k$

$\mathbf{r}_0 \leftarrow \mathbf{f} - \mathbf{A}\mathbf{x}_0$;
$\mathbf{z}_0 \leftarrow \mathbf{M}^{-1}\mathbf{r}_0$;
$\mathbf{p}_0 \leftarrow \mathbf{z}_0$;
$k \leftarrow 0$;
while $\|\mathbf{r}_k\| \geq Tol$ **and** $k < N_{max}$ **do**
  $\alpha_k \leftarrow \dfrac{\mathbf{z}_k^T \mathbf{r}_k}{\mathbf{p}_k^T \mathbf{A}\mathbf{p}_k}$;
  $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \alpha_k \mathbf{p}_k$;
  $\mathbf{r}_{k+1} \leftarrow \mathbf{r}_k - \alpha_k \mathbf{A}\mathbf{p}_k$;
  $\mathbf{z}_{k+1} \leftarrow \mathbf{M}^{-1}\mathbf{r}_{k+1}$;
  $\beta_k \leftarrow \dfrac{\mathbf{z}_{k+1}^T \mathbf{r}_{k+1}}{\mathbf{z}_k^T \mathbf{r}_k}$;
  $\mathbf{p}_{k+1} \leftarrow \mathbf{z}_{k+1} + \beta_k \mathbf{p}_k$;
  $k \leftarrow k + 1$;

We mentioned before that iteration count in which the CG method converges is proportional to $\kappa \equiv \kappa(\mathbf{A})$. Therefore, if $\mathbf{A}$ is poorly conditioned the convergence is very slow. Thus, we

should pick $\mathbf{M}$ so that $\kappa(\widehat{\mathbf{A}}) \ll \kappa(\mathbf{A})$, resulting in faster convergence.

If $\mathbf{M}^{-1} = \mathbf{L}\mathbf{L}^T$, then

$$\widehat{\mathbf{A}} = \mathbf{M}^{-1}\mathbf{A} = \mathbf{L}\mathbf{L}^T\mathbf{A}\mathbf{L}\mathbf{L}^{-1} = \mathbf{L}(\mathbf{L}^T\mathbf{A}\mathbf{L})\mathbf{L}^{-1} = \mathbf{L}\widetilde{\mathbf{A}}\mathbf{L}^{-1}.$$

This means $\widehat{\mathbf{A}}$ and $\widetilde{\mathbf{A}}$ are similar matrices, so they have the same spectrum and the estimate given by Theorem 2.4 applied to the system $\widetilde{\mathbf{A}}\widetilde{\mathbf{x}} = \widetilde{\mathbf{f}}$ is the same as for the system $\hat{\mathbf{A}}\mathbf{x} = \widehat{\mathbf{f}}$ (although note that Theorem 2.4 does not apply to $\widehat{\mathbf{A}}$, since this matrix is not symmetric).

Remark 2.2. *Applying the CG method to the system* $\widetilde{\mathbf{A}}\widetilde{\mathbf{x}} = \widetilde{\mathbf{f}}$ *is equivalent to using PCG on* $\mathbf{A}\mathbf{x} = \mathbf{f}$.

2.3 Multigrid Methods

Multigrid methods exploit discretizations of a given problem using different mesh sizes, seeking better convergence than iterative methods. These techniques are founded in the principle of divide and conquer.

As Saad notes [18, Chap. 13, Pag. 423]

> "Most relaxation-type iterative processes, such as Gauss-Seidel, may converge slowly for typical problems, it can be noticed that the components of the errors (or residuals) in the directions of the eigenvectors of the iteration matrix corresponding to the *large* eigenvalues are damped very rapidly. These eigenvectors are known as the *oscillatory* or *high-frequency* modes. The other components, associated with low-frequency or smooth modes, are difficult to damp with standard relaxation. This causes the observed slow down of all basic iterative methods."

Iterative methods do a good job of damping high-frequency modes, but low-frequency modes remains almost the same, converging very slow. But there low-frequency modes can be represented well on coarse meshes, which are low in dimension and thus easier to solve. Therefore more error components can be eliminated by transferring the problem from a fine mesh to a coarse mesh. Obviously, this process can be repeated using a hierarchy of meshes or grids.

2.3.1 Two-Grid Cycle

Let us start explaining the case where there are only two levels (grids). Let Ω_1 be a space of dimension n_1 and Ω_2 of dimension n_2, such that $n_2 < n_1$. Note that Ω_2 is not necessarily

a subspace of Ω_1, but those two spaces are related by two operators.

1. **Prolongation (P)**. This operator is also called interpolation and, as the name suggests, it maps Ω_2 into Ω_1, i.e.,
$$\mathbf{P} : \Omega_2 \to \Omega_1.$$
Note that this is not a surjective map.

2. **Restriction (R)**. This operator maps Ω_1 into Ω_2, i.e.,
$$\mathbf{R} : \Omega_1 \to \Omega_2.$$

Since the spaces Ω_1 and Ω_2 are related to linear systems arising from finite element methods, usually the operators $\mathbf{P}$ and $\mathbf{R}$ satisfy the *Galerkin property*

$$\mathbf{R} = \mathbf{P}^T,$$

and we refer to Ω_1 as the *fine level* and Ω_2 as the *coarse level*. A geometrical interpretation of the relationship between $\Omega_1, \Omega_2, \mathbf{P}$, and $\mathbf{R}$ is shown in the Figure 2.3.

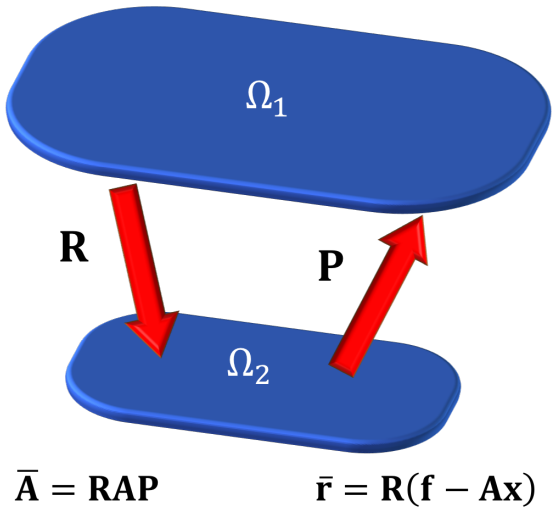


Figure 2.3: Relationship between the spaces Ω_1, Ω_2 and the operators $\mathbf{P}$ and $\mathbf{R}$.

Assume we have a discretization of Ω_1 and Ω_2. We can think in these discretizations as grids, called the *fine grid* and *coarse grid*, and we will still refer to them by Ω_1 and Ω_2, with grid-size h_1 and h_2, respectively. Moreover $h_1 < h_2$, since $n_1 > n_2$.

In the Geometric Multigrid (GMG) framework, $\Omega_2 \subset \Omega_1$ (as grids), so $\mathbf{R}$ can be the *canonical injection*, but in the Algebraic Multigrid (AMG) framework, grids are replaced by algebraic equations, which can be thought as linear systems of the form $\mathbf{Ax} = \mathbf{f}$. Since under certain

properties each matrix $\mathbf{A}$ induces a grid or mesh, then most of the GMG framework's ideas work in the AMG framework. The main difference relies on the construction of $\mathbf{R}$; since we are assuming the Galerkin property, constructing an appropriate $\mathbf{R}$ is equivalent to constructing an appropriate $\mathbf{P}$.

In addition to the restriction and prolongation operators we will also use a **smoother** which is any iterative method that quickly damps high-frequency components of the error. Classical smoothers are damped Jacobi, Gauss-Seidel, and SOR iterations.

In our two-level explanation, the highest level Ω_1 (fine grid) a mesh of size h_1 results in a problem of the form

$$\mathbf{Ax} = \mathbf{f}.$$

Let $\mathbf{x}_0$ denote the starting vector or initial guess, giving the algebraic error

$$\mathbf{e}_0 = \mathbf{u} - \mathbf{x}_0, v \quad \text{where} \quad \mathbf{u} \quad \text{is the exact solution.}$$

Note that $\mathbf{e}_0$ can be decomposed as

$$\mathbf{e}_0 = \mathbf{P}\mathbf{e}_0^{(2)} + \mathbf{e}_0^{(1)} \quad \text{where} \quad \mathbf{e}_0^{(2)} \in \Omega_2 \quad \text{and} \quad \mathbf{e}_0^{(1)} \in \Omega_1.$$

Since $\mathbf{e}_0^{(2)} \in \Omega_2$, then $\mathbf{P}\mathbf{e}_0^{(2)} \in \Omega_1$, the decomposition of $\mathbf{e}_0$ is well defined. This means, if the interpolated error coming from the coarse level does not capture all the error existing at the fine level, there is a portion of the error that only lives in the fine level.

The first step is to use a smother that rapidly reduces the fine level component of the error. Typically we use two or three iterations of the form (2.5). Although it is convenient to describe the smoothing iteration in terms of its effects on the algebraic error, in practice the smoother is applied to the residual:

$$\mathbf{r}_k = \mathbf{f} - \mathbf{A}\mathbf{x}_k = \mathbf{A}\mathbf{x} - \mathbf{A}\mathbf{x}_k = \mathbf{A}(\mathbf{x} - \mathbf{x}_k) = \mathbf{A}\mathbf{e}_k.$$

By equation (2.6), we get

$$\mathbf{r}_k = \mathbf{A}\mathbf{e}_k = \mathbf{A}(\mathbf{I} - \mathbf{M}^{-1}\mathbf{A})^k\mathbf{e}_0 = (\mathbf{I} - \mathbf{A}\mathbf{M}^{-1})^k\mathbf{A}\mathbf{e}_k.$$

The last equality holds since $\mathbf{A}(\mathbf{I} - \mathbf{M}^{-1}\mathbf{A}) = \mathbf{A} - \mathbf{A}\mathbf{M}^{-1}\mathbf{A} = (\mathbf{I} - \mathbf{A}\mathbf{M}^{-1})\mathbf{A}$.

Now, using the algebraic error $\mathbf{e}_0$, we get

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

Restricting the smoothed residual to the coarse level and setting $\bar{\mathbf{r}} = \mathbf{R}\mathbf{r}_k$, we get

$$\bar{\mathbf{r}} = \mathbf{R}\mathbf{r}_k = \mathbf{P}^T\mathbf{r}_k = \mathbf{P}^T(\mathbf{I} - \mathbf{A}\mathbf{M}^{-1})^k\mathbf{r}_0.$$

On the other hand $\mathbf{P}^T\mathbf{A}\mathbf{e}_k = \mathbf{P}^T\mathbf{A}(\mathbf{x} - \mathbf{x}_k) = \mathbf{P}^T(\mathbf{f} - \mathbf{A}\mathbf{x}_k) = \mathbf{P}^T\mathbf{r}_k$, or equivalently,

$$\mathbf{R}\mathbf{A}\mathbf{e}_k = \mathbf{R}\mathbf{r}_k. \tag{2.14}$$

Setting $\mathbf{e}_k = \mathbf{P}\bar{\mathbf{e}}$, by (2.14) we get

$$\mathbf{R}\mathbf{A}\mathbf{P}\bar{\mathbf{e}} = \mathbf{R}\mathbf{A}\mathbf{e}_k = \mathbf{R}\mathbf{r}_k = \bar{\mathbf{r}}.$$

Thus, we get the coarse level problem.

Find $\bar{\mathbf{e}}$ such that

$$\mathbf{R}\mathbf{A}\mathbf{P}\bar{\mathbf{e}} = \bar{\mathbf{r}}, \tag{2.15}$$

where $\bar{\mathbf{e}}$ is called the *coarse level correction* and $\bar{\mathbf{A}} = \mathbf{R}\mathbf{A}\mathbf{P}$ is called the *coarse level representation of* $\mathbf{A}$.

Proposition 2.1. *If* $\mathbf{A}$ *is a symmetric matrix, then* $\bar{\mathbf{A}}$ *and* $\bar{\mathbf{A}}^{-1}$ *are also symmetric matrices.*

Proof. Since $\mathbf{A}^T = \mathbf{A}$ and $\mathbf{R} = \mathbf{P}^T$ then

$$\bar{\mathbf{A}}^T = (\mathbf{R}\mathbf{A}\mathbf{P})^T = \mathbf{P}^T\mathbf{A}^T\mathbf{R}^T = \mathbf{R}\mathbf{A}\mathbf{P} = \bar{\mathbf{A}} \quad \text{and} \quad \bar{\mathbf{A}}^{-T} = \left(\bar{\mathbf{A}}^T\right)^{-1} = \bar{\mathbf{A}}^{-1}.$$

■

The second step is the coarse level correction. Since the prolonged solution of (2.3.1) represents a good correction to the coarse level component of the error, we update $\mathbf{x}_k$ as

$$\mathbf{x}_{k+1} \longleftarrow \mathbf{x}_k + \mathbf{P}\bar{\mathbf{e}},$$

leading to the Two-Level iteration shown in the Algorithm 4.

Algorithm 4: Two-Level iteration

input : $\mathbf{A}, \mathbf{x}_k, \mathbf{f}, \nu$
output: $\mathbf{x}_{k+1}$

$\mathbf{x}_k \leftarrow$ `pre_smooth(`$\mathbf{A}, \mathbf{x}_k, \mathbf{f}, \nu$`)` `// Pre-smooth` ν `times`
$\bar{\mathbf{r}} \leftarrow \mathbf{R}(\mathbf{f} - \mathbf{A}\mathbf{x}_k)$ `// Restrict the residual`
Solve $\bar{\mathbf{A}}\bar{\mathbf{e}} = \bar{\mathbf{r}}$ for $\bar{\mathbf{e}}$ `// Solve the coarse level problem`
$\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \mathbf{P}\bar{\mathbf{e}}$ `// Prolong and update`

The notation

$$\mathbf{x}_k \longleftarrow \texttt{pre\_smooth}(\mathbf{A}, \mathbf{x}_k, \mathbf{f}, \nu)$$

means that $\mathbf{x}_k$ is overwritten by the result of ν smoothing steps applied to the system with initial guess $\mathbf{x}_k$.

To write the Two-Level iteration in terms of the error $\mathbf{e}_k = \mathbf{x} - \mathbf{x}_k$, we should recall that smoothing iterations are of form (2.12), i.e.,

$$\mathbf{x}_{k+1} = \mathbf{T}\mathbf{x}_k + \mathbf{b}, \quad \text{where} \quad \mathbf{T} = \mathbf{M}^{-1}\mathbf{N} \quad \text{and} \quad \mathbf{b} = \mathbf{M}^{-1}\mathbf{f}.$$

Thus equality (2.6) holds, i.e., after ν smoothing steps, the error $\mathbf{e}_k$ is updated by

$$\mathbf{e}_k = (\mathbf{I} - \mathbf{M}^{-1}\mathbf{A})^{\nu}\mathbf{e}_k.$$

Then, restricting the residual, we get

$$\bar{\mathbf{r}} = \mathbf{R}(\mathbf{f} - \mathbf{A}\mathbf{x}_k) = \mathbf{R}(\mathbf{A}\mathbf{x} - \mathbf{A}\mathbf{x}_k) = \mathbf{R}\mathbf{A}(\mathbf{x} - \mathbf{x}_k) = \mathbf{R}\mathbf{A}\mathbf{e}_k.$$

Thus $\bar{\mathbf{r}}$ is updated by

$$\bar{\mathbf{r}} = \mathbf{R}\mathbf{A}(\mathbf{I} - \mathbf{M}^{-1}\mathbf{A})^{\nu}\mathbf{e}_k.$$

Solving the coarse level problem $\bar{\mathbf{A}}\bar{\mathbf{e}} = \bar{\mathbf{r}}$ exactly means $\bar{\mathbf{e}} = \bar{\mathbf{A}}^{-1}\bar{\mathbf{r}}$, so

$$\bar{\mathbf{e}} = \bar{\mathbf{A}}^{-1}\mathbf{R}\mathbf{A}(\mathbf{I} - \mathbf{M}^{-1}\mathbf{A})^{\nu}\mathbf{e}_k.$$

Finally, prolong and update $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \mathbf{P}\bar{\mathbf{e}}$. In terms of the error $\mathbf{e}_k$, this last step becomes

$$\mathbf{e}_{k+1} = \mathbf{x} - \mathbf{x}_{k+1} = \mathbf{x} - \mathbf{x}_k - \mathbf{P}\bar{\mathbf{e}}.$$

Since $\mathbf{e}_k$ was already smoothed, we get

$$\begin{aligned} \mathbf{e}_{k+1} &= \mathbf{e}_k - \mathbf{P}\bar{\mathbf{e}} \\ &= (\mathbf{I} - \mathbf{M}^{-1}\mathbf{A})^{\nu}\mathbf{e}_k - \mathbf{P}\bar{\mathbf{A}}^{-1}\mathbf{R}\mathbf{A}(\mathbf{I} - \mathbf{M}^{-1}\mathbf{A})^{\nu}\mathbf{e}_k \\ &= (\mathbf{I} - \mathbf{P}\bar{\mathbf{A}}^{-1}\mathbf{R}\mathbf{A})(\mathbf{I} - \mathbf{M}^{-1}\mathbf{A})^{\nu}\mathbf{e}_k. \end{aligned}$$

Thus the Two-Level iteration affects the error as

$$\mathbf{e}_{k+1} = (\mathbf{I} - \mathbf{P}\bar{\mathbf{A}}^{-1}\mathbf{R}\mathbf{A})(\mathbf{I} - \mathbf{M}^{-1}\mathbf{A})^{\nu}\mathbf{e}_k. \tag{2.16}$$

Proposition 2.2. *Define*

$$\mathcal{Q} = \mathbf{P}\bar{\mathbf{A}}^{-1}\mathbf{R}\mathbf{A} \quad \textit{and} \quad \mathcal{P} = \mathbf{I} - \mathcal{Q} = \mathbf{I} - \mathbf{P}\bar{\mathbf{A}}^{-1}\mathbf{R}\mathbf{A}. \tag{2.17}$$

Then $\mathcal{P}$ and $\mathcal{Q}$ are projections.

Proof. Let us start with $\mathcal{Q}$. Since $\bar{\mathbf{A}} = \mathbf{RAP}$, then

$$\mathcal{Q}^2 = (\mathbf{P}\bar{\mathbf{A}}^{-1}\mathbf{RA})(\mathbf{P}\bar{\mathbf{A}}^{-1}\mathbf{RA}) = \mathbf{P}\bar{\mathbf{A}}^{-1}(\mathbf{RAP})\bar{\mathbf{A}}^{-1}\mathbf{RA} = \mathbf{P}\bar{\mathbf{A}}^{-1}\mathbf{RA} = \mathcal{Q}.$$

Similarly $\mathcal{P}$:

$$\mathcal{P}^2 = (\mathbf{I} - \mathcal{Q})(\mathbf{I} - \mathcal{Q}) = \mathbf{I} - 2\mathcal{Q} + \mathcal{Q}^2 = \mathbf{I} - \mathcal{Q} = \mathcal{P}.$$

■

Therefore the action of the Two-Level iteration over $\mathbf{e}_k$ is given by

$$\mathbf{e}_{k+1} = \mathcal{P}(\mathbf{I} - \mathbf{M}^{-1}\mathbf{A})^{\nu}\mathbf{e}_k.$$

Moreover $\mathcal{P}$ only can have eigenvalues 0 and 1, since $\mathcal{P}\mathbf{v} = \lambda\mathbf{v}$ implies

$$\lambda\mathbf{v} = \mathcal{P}\mathbf{v} = \mathcal{P}^2\mathbf{v} = \mathcal{P}(\mathcal{P}\mathbf{v}) = \mathcal{P}(\lambda\mathbf{v}) = \lambda\mathcal{P}\mathbf{v} = \lambda^2\mathbf{v}.$$

Later on, we will see that Ω_1 can be decomposed into two subspaces: one annihilated by $\mathcal{P}$, and its $\mathbf{A}$-orthogonal complement, which is left unchanged by $\mathcal{P}$.

The iteration matrix defined by (2.16) is not symmetric with respect to the $\langle\cdot,\cdot\rangle_{\mathbf{A}}$ inner product. Symmetry can be imposed by introducing post-smoothing after the correction step, i.e., further iterations of the transposed smoother are applied. Recall that the splitting $\mathbf{A} = \mathbf{M} - \mathbf{N}$ gave the smoothing stationary iteration $\mathbf{x}_{k+1} = \mathbf{M}^{-1}\mathbf{N}\mathbf{x}_k + \mathbf{M}^{-1}\mathbf{f}$. For the transposed smoother, we let

$$\mathbf{A} = \mathbf{A}^T = (\mathbf{M} - \mathbf{N})^T = \mathbf{M}^T - \mathbf{N}^T.$$

Thus the smoother iteration becomes

$$\mathbf{x}_{k+1} = \mathbf{M}^{-T}\mathbf{N}^T\mathbf{x}_k + \mathbf{M}^{-T}\mathbf{f} = \mathbf{M}^{-T}(\mathbf{M}^T - \mathbf{A})\mathbf{x}_k + \mathbf{M}^{-T}\mathbf{f} = (\mathbf{I} - \mathbf{M}^{-T}\mathbf{A})\mathbf{x}_k + \mathbf{M}^{-T}\mathbf{f}.$$

Since the exact solution $\mathbf{x}$, also satisfies this new split iteration, i.e.,

$$\mathbf{x} = (\mathbf{I} - \mathbf{M}^{-T}\mathbf{A})\mathbf{x} + \mathbf{M}^{-T}\mathbf{f}.$$

Then, the error $\mathbf{e}_{k+1} = \mathbf{x} - \mathbf{x}_{k+1}$ is given by

$$\mathbf{x} - \mathbf{x}_{k+1} = (\mathbf{I} - \mathbf{M}^{-T}\mathbf{A})\mathbf{x} + \mathbf{M}^{-T}\mathbf{f} - (\mathbf{I} - \mathbf{M}^{-T}\mathbf{A})\mathbf{x}_k - \mathbf{M}^{-T}\mathbf{f} = (\mathbf{I} - \mathbf{M}^{-T}\mathbf{A})(\mathbf{x} - \mathbf{x}_k).$$

Therefore

$$\mathbf{e}_{k+1} = (\mathbf{I} - \mathbf{M}^{-T}\mathbf{A})\mathbf{e}_k.$$

This means, k steps of the transposed smoother affect the error $\mathbf{e}_0$ of the stationary iteration:

$$\mathbf{e}_k = (\mathbf{I} - \mathbf{M}^{-T}\mathbf{A})^k\mathbf{e}_0. \tag{2.18}$$

Now adding ν post-smoothing steps to the Two-Level algorithm,

$$\mathbf{x}_{k+1} \leftarrow \texttt{post\_smooth}(\mathbf{A}, \mathbf{x}_{k+1}, \mathbf{f}, \nu),$$

the error (2.16) is modified by (2.18) as

$$\mathbf{e}_{k+1} = (\mathbf{I} - \mathbf{M}^{-T}\mathbf{A})^{\nu}(\mathbf{I} - \mathbf{P}\bar{\mathbf{A}}^{-1}\mathbf{R}\mathbf{A})(\mathbf{I} - \mathbf{M}^{-1}\mathbf{A})^{\nu}\mathbf{e}_k. \tag{2.19}$$

Proposition 2.3. *Define*

$$\mathcal{M} = (\mathbf{I} - \mathbf{M}^{-T}\mathbf{A})^{\nu}(\mathbf{I} - \mathbf{P}\bar{\mathbf{A}}^{-1}\mathbf{R}\mathbf{A})(\mathbf{I} - \mathbf{M}^{-1}\mathbf{A})^{\nu}. \tag{2.20}$$

If $\mathbf{A}$ *is symmetric, then* $\mathcal{M}$ *is symmetric with respect to* $\langle\cdot,\cdot\rangle_{\mathbf{A}}$*, i.e.,*

$$\mathbf{A}\mathcal{M} = \mathcal{M}^T\mathbf{A}.$$

Proof. Let $\nu = 1$ so

$$\begin{aligned}
\mathcal{M}^T\mathbf{A} &= ((\mathbf{I} - \mathbf{M}^{-T}\mathbf{A})(\mathbf{I} - \mathbf{P}\bar{\mathbf{A}}^{-1}\mathbf{R}\mathbf{A})(\mathbf{I} - \mathbf{M}^{-1}\mathbf{A}))^T\mathbf{A} \\
&= (\mathbf{I} - \mathbf{M}^{-1}\mathbf{A})^T(\mathbf{I} - \mathbf{P}\bar{\mathbf{A}}^{-1}\mathbf{R}\mathbf{A})^T(\mathbf{I} - \mathbf{M}^{-T}\mathbf{A})^T\mathbf{A} \\
&= (\mathbf{I} - \mathbf{A}^T\mathbf{M}^{-T})(\mathbf{I} - (\mathbf{P}\bar{\mathbf{A}}^{-1}\mathbf{R}\mathbf{A})^T)(\mathbf{I} - \mathbf{A}^T\mathbf{M}^{-1})\mathbf{A} \\
&= (\mathbf{I} - \mathbf{A}\mathbf{M}^{-T})(\mathbf{I} - \mathbf{A}^T\mathbf{R}^T\bar{\mathbf{A}}^{-T}\mathbf{P}^T)(\mathbf{I} - \mathbf{A}\mathbf{M}^{-1})\mathbf{A}.
\end{aligned}$$

Since $\mathbf{A}$ is symmetric by Proposition 2.1 and since $\mathbf{R} = \mathbf{P}^T$, we get

$$\begin{aligned}
\mathcal{M}^T\mathbf{A} &= (\mathbf{I} - \mathbf{A}\mathbf{M}^{-T})(\mathbf{I} - \mathbf{A}\mathbf{P}\bar{\mathbf{A}}^{-1}\mathbf{R})(\mathbf{I} - \mathbf{A}\mathbf{M}^{-1})\mathbf{A} \\
&= (\mathbf{I} - \mathbf{A}\mathbf{M}^{-T})(\mathbf{I} - \mathbf{A}\mathbf{P}\bar{\mathbf{A}}^{-1}\mathbf{R})\mathbf{A}(\mathbf{I} - \mathbf{M}^{-1}\mathbf{A}) \\
&= (\mathbf{I} - \mathbf{A}\mathbf{M}^{-T})\mathbf{A}(\mathbf{I} - \mathbf{P}\bar{\mathbf{A}}^{-1}\mathbf{R}\mathbf{A})(\mathbf{I} - \mathbf{M}^{-1}\mathbf{A}) \\
&= \mathbf{A}(\mathbf{I} - \mathbf{M}^{-T}\mathbf{A})(\mathbf{I} - \mathbf{P}\bar{\mathbf{A}}^{-1}\mathbf{R}\mathbf{A})(\mathbf{I} - \mathbf{M}^{-1}\mathbf{A}) \\
&= \mathbf{A}\mathcal{M}.
\end{aligned}$$

When $\nu > 1$, since $(\mathbf{I} - \mathbf{A}\mathbf{M}^{-1})\mathbf{A} = \mathbf{A}(\mathbf{I} - \mathbf{M}^{-1}\mathbf{A})$, we see

$$(\mathbf{I} - \mathbf{A}\mathbf{M}^{-1})^{\nu}\mathbf{A} = \mathbf{A}(\mathbf{I} - \mathbf{M}^{-1}\mathbf{A})^{\nu}.$$

Similarly, we can show that

$$(\mathbf{I} - \mathbf{A}\mathbf{M}^{-T})^{\nu}\mathbf{A} = \mathbf{A}(\mathbf{I} - \mathbf{M}^{-T}\mathbf{A})^{\nu}.$$

Therefore $\mathbf{A}\mathcal{M} = \mathcal{M}^T\mathbf{A}$ for all $\nu \geq 1$ and so $\mathcal{M}$ is symmetric with respect to $\langle\cdot,\cdot\rangle_{\mathbf{A}}$. ∎

In summary we have Algorithm 5, which is the generalized Two-Level iteration, also known as the Two-Grid cycle.

Algorithm 5: Generalized Two-Level iteration (`Two-Grid cycle`)

input : $\mathbf{A}, \mathbf{x}_k, \mathbf{f}, \nu$
output: $\mathbf{x}_{k+1}$

$\mathbf{x}_k \leftarrow$ `pre_smooth(`$\mathbf{A}, \mathbf{x}_k, \mathbf{f}, \nu$`)`	`// Pre-smooth` ν `times`
$\bar{\mathbf{r}} \leftarrow \mathbf{R}(\mathbf{f} - \mathbf{A}\mathbf{x}_k)$	`// Restrict the residual`
Solve $\bar{\mathbf{A}}\bar{\mathbf{e}} = \bar{\mathbf{r}}$ for $\bar{\mathbf{e}}$	`// Solve the coarse level problem`
$\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \mathbf{P}\bar{\mathbf{e}}$	`// Prolong and update`
$\mathbf{x}_{k+1} \leftarrow$ `post_smooth(`$\mathbf{A}, \mathbf{x}_{k+1}, \mathbf{f}, \nu$`)`	`// Post-smooth` ν `times`

Remark 2.3.

1. *The number of pre/post-smoothing steps are equal to ensure that* $\mathcal{M}$ *is symmetric with respect to the* $\mathbf{A}$*-inner product.*

2. *If the splitting* $\mathbf{A} = \mathbf{M} - \mathbf{N}$ *is such that* $\mathbf{M} = \mathbf{M}^T$ *then*

$$\mathbf{x}_{k+1} = (\mathbf{I} - \mathbf{M}^{-1}\mathbf{A})\mathbf{x}_k + \mathbf{M}^{-1}\mathbf{f} = (\mathbf{I} - \mathbf{M}^{-T}\mathbf{A})\mathbf{x}_k + \mathbf{M}^{-T}\mathbf{f},$$

   which means the pre/post-smoothing are equivalent. This happens when Jacobi or damped Jacobi iterations are used as smoothers, but not with Gaus-Seidel or SOR. Thus we can use their symmetric version SGS and SSOR, but we will not discuss these algorithms.

3. *When a smoother is applied at the fine level, the residual* $\mathbf{r}_k$ *obtained after the smoothing step will usually remain large. Even so* $\mathbf{r}_k$ *will have small components in the high-frequency modes. If solving exactly the coarse problem can remove these components, then we expect a better solution.*

4. *Solvers based on the Two-Grid cycle are hardly ever used, since the coarse-level problem remains too large to be solved exactly. However, the Two-Grid cycle is convenient for theoretical purposes.*

2.3.2 V-cycles and W-cycles

The Two-Grid cycle can be generalized by using recursion: apply the Two-Grid cycle recursively on the obtained coarse grids until a coarse enough grid is reached; then solve the coarsest grid problem exactly (usually via Gaussian elimination). This idea gives a multigrid cycle called V-cycle. In the GMG framework, h_k stands for the grid-size of the discretization of the space Ω_k, so the hierarchy of grid levels is built until an appropriate level, the coarsest level of grid-size h_p, is reached. In the AMG, since there are not grids, the hierarchy of levels is built depending on the number of *degrees of freedom* (DOF) per level.

Since the number of DOF gives an idea of the dimension of the space, we will denote by n_k the number of DOF at level k. Then, the hierarchy of levels is built until the last level has an appropriate number of DOF, let us say n_{p+1}.

Algorithm 6: V-Cycle iteration (`V-Cycle`)

input : $\mathbf{A}, \mathbf{x}_k, \mathbf{f}, \nu$
output: $\mathbf{x}_{k+1}$

$\mathbf{x}_k \leftarrow$ `pre_smooth`$(\mathbf{A}, \mathbf{x}_k, \mathbf{f}, \nu)$ `// Pre-smooth` ν `times`
$\bar{\mathbf{r}} \leftarrow \mathbf{R}(\mathbf{f} - \mathbf{A}\mathbf{x}_k)$ `// Restrict the residual`
if $h = h_{p+1}$ **then**
  Solve $\bar{\mathbf{A}}\bar{\mathbf{e}} = \bar{\mathbf{r}}$ for $\bar{\mathbf{e}}$ `// Solve the coarsest level problem`
else
  $\bar{\mathbf{e}} \leftarrow$ `V-Cycle`$(\bar{\mathbf{A}}, \mathbf{0}, \bar{\mathbf{r}}, \nu)$ `// Recursion`
$\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \mathbf{P}\bar{\mathbf{e}}$ `// Prolong and correct`
$\mathbf{x}_{k+1} \leftarrow$ `post_smooth`$(\mathbf{A}, \mathbf{x}_{k+1}, \mathbf{f}, \nu)$ `// Post-smooth` ν `times`

Note that Algorithm 6 comes from the GMG framework, so we have a grid-size condition $h = h_{p+1}$, but in the AMG framework that condition is replaced by $n = n_{p+1}$.

Remark 2.4. *In further algorithms we will assume the GMG framework, but replacing the grid-size condition by a number of DOF condition gives a suitable version for AMG.*

Remark 2.5.

1. *If the V-cycle has $p+1$ spaces Ω_i with discretization mesh-size h_i, where $h_{i+1} > h_i$ for $i = 1, 2, \ldots, p$, or, equivalently, with $n_i > n_{i+1}$ DOFs for $i = 1, 2, \ldots, p$, then there are p prolongation and restriction operators*

$$\mathbf{P}_i : \Omega_{i+1} \to \Omega_i \quad \text{and} \quad \mathbf{R}_i : \Omega_i \to \Omega_{i+1} \quad \text{for} \quad i = 1, 2, \ldots, p$$

and $p+1$ operators $\bar{\mathbf{A}}_i : \Omega_i \to \Omega_i$ for $i = 1, 2, \ldots, p+1$ such that

$$\mathbf{R}_i = \mathbf{P}_i^T \quad \text{and} \quad \bar{\mathbf{A}}_{i+1} = \mathbf{R}_i \bar{\mathbf{A}}_i \mathbf{P}_i, \quad \text{where} \quad \bar{\mathbf{A}}_1 = \mathbf{A}.$$

2. *In the* `V-Cycle` *recursion, we set $\mathbf{A} = \bar{\mathbf{A}}_i$, the initial guess $\mathbf{x} = \mathbf{0}$ and $\mathbf{f} = \bar{\mathbf{r}}$, so at each level the residuals are different but of the form*

$$\bar{\mathbf{r}}_i = \mathbf{R}_i(\bar{\mathbf{r}}_i - \bar{\mathbf{A}}_i \mathbf{x}_k^{(i)}) \quad \text{for} \quad i = 1, 2, \ldots, p;$$

where $\mathbf{x}_k^{(i)}$ has been modified by ν smoothing steps so that the components of the residual associated with the high-frequency modes could be removed in the next coarse levels and so a better approximation should result at the end of the V-cycle.

A geometric interpretation of the relationship between the spaces Ω_i, Ω_{i+1} and the operators $\mathbf{P}_i, \mathbf{R}_i$ is shown in Figure 2.4.

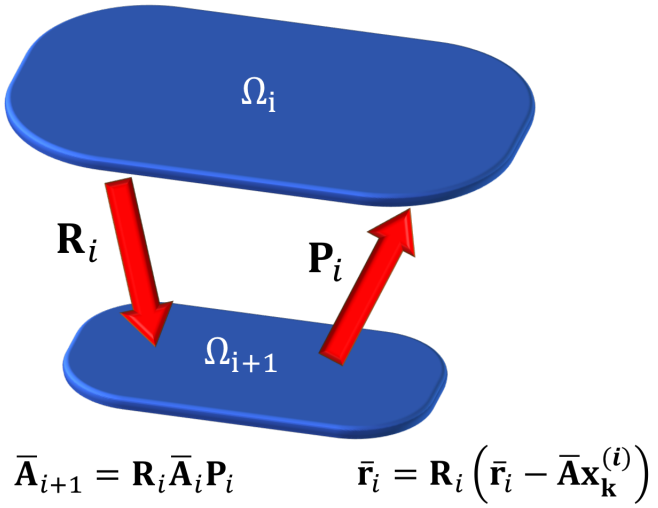


Figure 2.4: Relationship between the spaces Ω_i, Ω_{i+1} and the operators $\mathbf{P}_i$, $\mathbf{R}_i$.

Now, we introduce a general `multigrid cycle (MG`γ`)`, which can be seen as a generalization of the `V-cycle` seen before, and its implementation is recursive as well.

Algorithm 7: Multigrid Cycle (`MG`γ)

input : $\mathbf{A}, \mathbf{x}_k, \mathbf{f}, \nu, \gamma$
output: $\mathbf{x}_{k+1}$

$\mathbf{x}_k \leftarrow$ `pre_smooth(`$\mathbf{A}, \mathbf{x}_k, \mathbf{f}, \nu$`)` `// Pre-smooth` ν `times`
$\bar{\mathbf{r}} \leftarrow \mathbf{R}(\mathbf{f} - \mathbf{A}\mathbf{x}_k)$ `// Restrict the residual`
if $h = h_{p+1}$ **then**
  Solve $\bar{\mathbf{A}}\bar{\mathbf{e}} = \bar{\mathbf{r}}$ for $\bar{\mathbf{e}}$ `// Solve the coarsest level`
else
  for $i \leftarrow 1$ **to** γ **do**
    $\bar{\mathbf{e}} \leftarrow$ `MG`γ`(`$\bar{\mathbf{A}}, \mathbf{0}, \bar{\mathbf{r}}, \nu, \gamma$`)` `// Recursion`
$\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \mathbf{P}\bar{\mathbf{e}}$ `// Prolong and correct`
$\mathbf{x}_{k+1} \leftarrow$ `post_smooth(`$\mathbf{A}, \mathbf{x}_{k+1}, \mathbf{f}, \nu$`)` `// Post-smooth` ν `times`

Note the recursive step in Algorithm 7 uses a new parameter γ, which indicates the number of `MG`γ iterations with initial guess $\mathbf{x} = \mathbf{0}$. The case $\gamma = 1$ gives `V-Cycle` multigrid, $\gamma = 2$ gives `W-Cycle` multigrid, and $\gamma = 3$ is rarely used. For a given γ we can have complex inter-grid up and down moves. The most used cycles are illustrated in Figure 2.5.

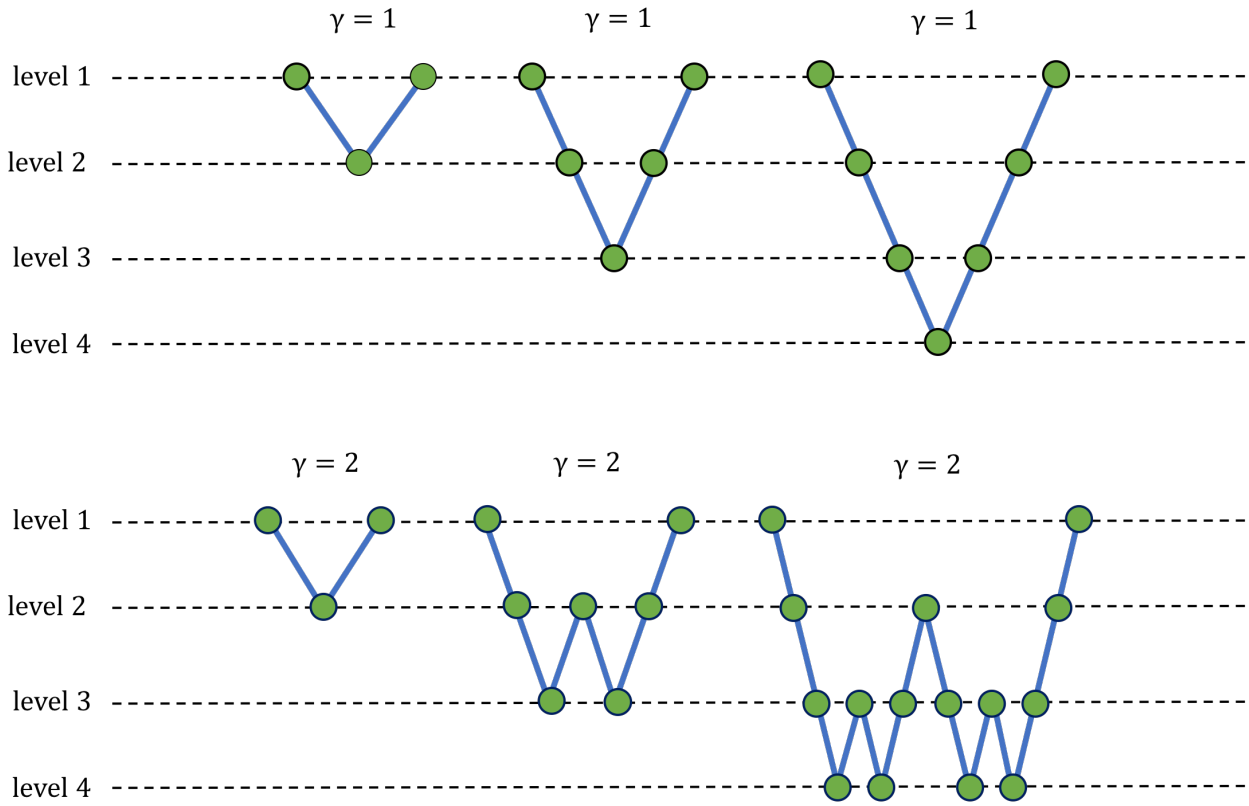


Figure 2.5: Representation of some V-Cycles and W-Cycles. Image adapted from [18].

2.3.3 Full Multigrid

In general, multigrid cycling schemes start with some smoothing steps on the fine grid, then restrict the problem to coarser levels to reduce low-frequency error components. Perhaps rather than start in the fine level, starting in the coarse level and proceeding to finer levels (only when we achieve a sufficiently good approximation) is better. In effect, we can use nested iteration on the multigrid cycling schema. This gives the *full multigrid* (FMG) cycling schema.

We can view FMG as an improvement on nested iteration, where rather than a smoothing step we use an MGγ cycle, so FMG goes from the bottom to the top, approximating the solution with only one sweep through the hierarchy of meshes.

Suppose there are $p+1$ spaces Ω_i with mesh-size h_i, where

$$h_{i+1} > h_i \quad \text{for} \quad i = 1, 2, \ldots, p,$$

and we keep using the notation introduced in Remark 2.5. Moreover, we define the coarse-grid right hand side

$$\bar{\mathbf{f}}_{p+1} = \mathbf{R}_p \bar{\mathbf{f}}_p \quad \text{and} \quad \bar{\mathbf{f}}_1 = \mathbf{f}.$$

The FMG algorithm starts by solving $\bar{\mathbf{A}}_{p+1}\bar{\mathbf{e}}_{p+1} = \bar{\mathbf{f}}_{p+1}$, then prolongs $\bar{\mathbf{e}}_{p+1}$ to the previous level and uses $\mathbf{P}_p\bar{\mathbf{e}}_{p+1}$ as the initial guess in the MGγ cycle starting at level p. The FMG iteration is given in Algorithm 8.

Algorithm 8: Full Multigrid (FMG)

input : $\mathbf{A}, \mathbf{f}, \nu, \gamma$
output: $\bar{\mathbf{e}}$

Solve $\bar{\mathbf{A}}_p\bar{\mathbf{e}} = \bar{\mathbf{f}}_p$ for $\bar{\mathbf{e}}$ // Solve the coarsest level
for $k \leftarrow p+1$ **to** 1 **do**
  $\bar{\mathbf{x}} \leftarrow \mathbf{P}_k\bar{\mathbf{e}}$ // Prolong to the next level
  $\bar{\mathbf{e}} \leftarrow \text{MG}\gamma(\bar{\mathbf{A}}_{k-1}, \bar{\mathbf{x}}, \bar{\mathbf{f}}_{k-1}, \nu, \gamma)$ // Apply the MGγ iteration

Figure 2.6 illustrates some FMG cycles where the double lines mean the interpolation is done inside the loop, and the single lines correspond to the MGγ cycle.

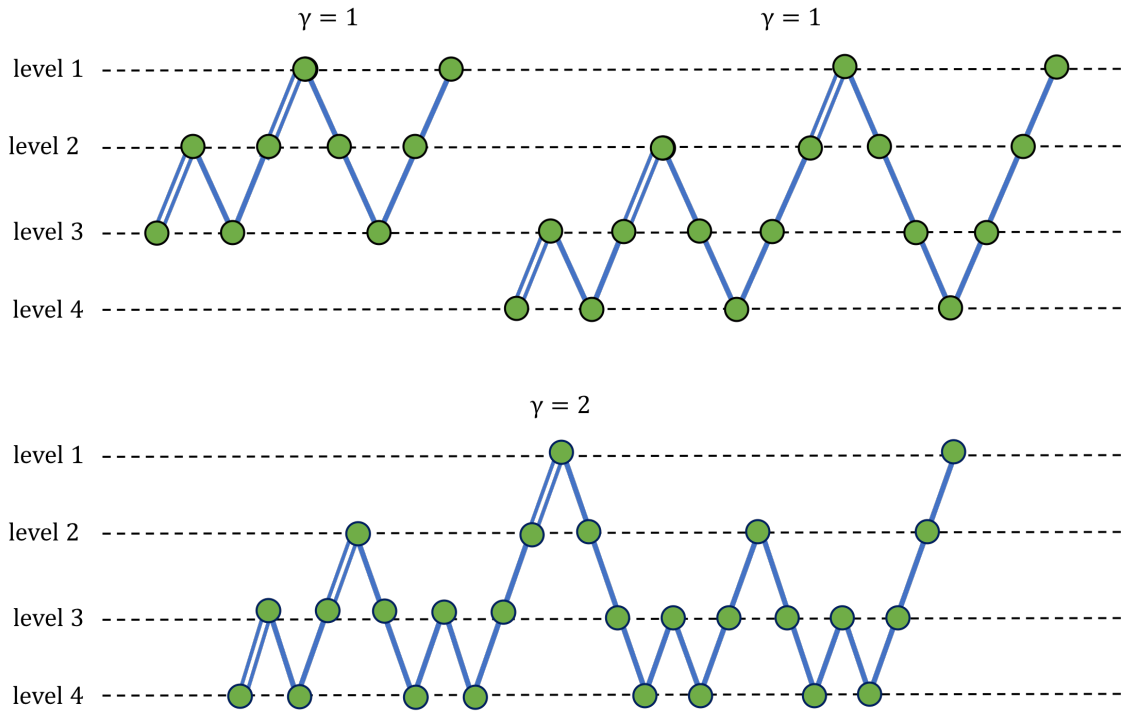


Figure 2.6: Representation of some FMG cycles. Image adapted from [18].

2.4 Analysis and Convergence

The Two-Grid cycle (Algorithm 5) is the foundation of most of the complex multigrid cycles. Recall that the general multigrid cycle (Algorithm 7) can be seen as a generalization of the Two-Grid cycle. Likewise, the FMG cycle (Algorithm 8) uses a V-cycle iteration in its inner loop. In this section we will explain formally the convergence of the Two-Grid cycle, when the inter-grid transfer operators satisfy the Galerkin property.

The following content is based on the section "Analysis for the two-grid cycle" in [18]. Further information about convergence in multigrid cycles can be found in [2], [6], and [16].

2.4.1 Important Spaces

As stated before in Proposition 2.2, the operators $\mathcal{Q}$ and $\mathcal{P}$ defined by (2.17), i.e.,

$$\mathcal{Q} = \mathbf{P}\bar{\mathbf{A}}^{-1}\mathbf{R}\mathbf{A} \quad \text{and} \quad \mathcal{P} = \mathbf{I} - \mathcal{Q} = \mathbf{I} - \mathbf{P}\bar{\mathbf{A}}^{-1}\mathbf{R}\mathbf{A}$$

are projectors onto Ω_0. Moreover, they are $\mathbf{A}$-orthogonal projectors, since they are self-adjoint with respect to the $\mathbf{A}$-inner product:

$$\mathcal{Q}^T\mathbf{A} = (\mathbf{P}\bar{\mathbf{A}}^{-1}\mathbf{R}\mathbf{A})^T\mathbf{A} = \mathbf{A}^T\mathbf{R}^T\bar{\mathbf{A}}^{-T}\mathbf{P}^T\mathbf{A} = \mathbf{A}\mathbf{P}\bar{\mathbf{A}}^{-1}\mathbf{R}\mathbf{A} = \mathbf{A}\mathcal{Q}.$$

Similarly

$$\mathcal{P}^T\mathbf{A} = (\mathbf{I} - \mathcal{Q})^T\mathbf{A} = \mathbf{A} - \mathcal{Q}^T\mathbf{A} = \mathbf{A} - \mathbf{A}\mathcal{Q} = \mathbf{A}(\mathbf{I} - \mathcal{Q}) = \mathbf{A}\mathcal{P}.$$

Thus

$$\langle \mathcal{P}\mathbf{z}, \mathbf{y}\rangle_{\mathbf{A}} = \langle \mathbf{z}, \mathcal{P}\mathbf{y}\rangle_{\mathbf{A}} \quad \text{and} \quad \langle \mathcal{Q}\mathbf{z}, \mathbf{y}\rangle_{\mathbf{A}} = \langle \mathbf{z}, \mathcal{Q}\mathbf{y}\rangle_{\mathbf{A}} \quad \text{for all} \quad \mathbf{y}, \mathbf{z} \in \Omega_0.$$

Clearly $\mathcal{N}(\mathcal{Q}) \subset \mathcal{R}(\mathbf{I} - \mathcal{Q})$; the converse inclusion also holds. To show it, take $\mathbf{z}$ in the range of $\mathbf{I} - \mathcal{Q}$, i.e., $\mathbf{z} = (\mathbf{I} - \mathcal{Q})\mathbf{y}$ for some $\mathbf{y} \in \Omega_0$. Then

$$\mathcal{Q}\mathbf{z} = \mathcal{Q}(\mathbf{I} - \mathcal{Q})\mathbf{y} = \mathcal{Q}(\mathbf{y} - \mathcal{Q}\mathbf{y}) = \mathcal{Q}\mathbf{y} - \mathcal{Q}^2\mathbf{y} = \mathcal{Q}\mathbf{y} - \mathcal{Q}\mathbf{y} = \mathbf{0}.$$

Therefore $\mathcal{R}(\mathbf{I} - \mathcal{Q}) \subset \mathcal{N}(\mathcal{Q})$, and hence

$$\mathcal{N}(\mathcal{Q}) = \mathcal{R}(\mathbf{I} - \mathcal{Q}).$$

Additionally, the subspaces $\mathcal{N}(\mathcal{Q})$ and $\mathcal{R}(\mathcal{Q})$ intersect trivially, i.e., the zero element is the unique element in their intersection. Indeed, if a vector $\mathbf{z}$ belongs to $\mathcal{R}(\mathcal{Q})$, i.e. $\mathbf{z} = \mathcal{Q}\mathbf{y}$ for some $\mathbf{y} \in \Omega_0$, then $\mathcal{Q}\mathbf{z} = \mathcal{Q}^2\mathbf{y} = \mathcal{Q}\mathbf{y} = \mathbf{z}$. Thus $\mathcal{Q}\mathbf{z} = \mathbf{z}$. If $\mathbf{z}$ is also in $\mathcal{N}(\mathcal{Q})$, then $\mathcal{Q}\mathbf{z} = \mathbf{0}$. Hence, $\mathbf{z} = \mathcal{Q}\mathbf{z} = \mathbf{0}$ which ends the proof.

Also, every element of Ω_0 can be expressed as $\mathbf{z} = \mathcal{Q}\mathbf{z} + (\mathbf{I} - \mathcal{Q})\mathbf{z}$. Thus, the space Ω_0 can be decomposed as the direct sum

$$\Omega_0 = \mathcal{R}(\mathcal{Q}) \oplus \mathcal{R}(\mathbf{I} - \mathcal{Q}) = \mathcal{R}(\mathcal{Q}) \oplus \mathcal{N}(\mathcal{Q}). \tag{2.21}$$

Note that the definition of $\mathcal{Q}$ implies that $\mathcal{R}(\mathcal{Q}) \subset \mathcal{R}(\mathbf{P})$. Moreover, the other direction of the inclusion also holds, which means that both subspaces are equal. This can be shown by taking a vector $\mathbf{z}$ in the range of $\mathcal{P}$, i.e., $\mathbf{z} \in \mathcal{R}(\mathbf{P})$, so $\mathbf{z} = \mathbf{P}\mathbf{y}$ for a certain $\mathbf{y} \in \Omega_0$. Remembering that $\bar{\mathbf{A}} = \mathbf{RAP}$, we obtain

$$\mathcal{Q}\mathbf{z} = \mathbf{P}\bar{\mathbf{A}}^{-1}\mathbf{RAP}\mathbf{y} = \mathbf{P}\bar{\mathbf{A}}^{-1}\bar{\mathbf{A}}\mathbf{y} = \mathbf{P}\mathbf{y} = \mathbf{z}.$$

Thus $\mathbf{z}$ belongs to $\mathcal{R}(\mathcal{Q})$. As a consequence,

$$\mathcal{R}(\mathcal{Q}) = \mathcal{R}(\mathbf{P}). \tag{2.22}$$

This means $\mathcal{Q}$ is the $\mathbf{A}$-orthogonal projector onto the space $\mathcal{R}(\mathbf{P})$. Meanwhile, $\mathcal{P}$ is the $\mathbf{A}$-orthogonal projector onto the $\mathbf{A}$-orthogonal complement, which is the range of $\mathbf{I} - \mathcal{Q}$ or the null space of $\mathcal{Q}$.

Finally, the null space of the restriction operator $\mathbf{R}$ is identical to the null space of $\mathcal{Q}$. From the definition of $\mathbf{R}$, it is clear that $\mathcal{N}(\mathbf{R}) \subset \mathcal{N}(\mathcal{Q})$. The reverse inclusion can be derived by applying Fundamental Theorem of Linear Algebra to $\mathbf{P}$ and using (2.22) as follows:

$$\Omega_0 = \mathcal{R}(\mathbf{P}) \oplus \mathcal{N}(\mathbf{P}^T) = \mathcal{R}(\mathcal{Q}) \oplus \mathcal{N}(\mathbf{P}^T) = \mathcal{R}(\mathcal{Q}) \oplus \mathcal{N}(\mathbf{R}).$$

Comparing with (2.21), it follows that $\mathcal{N}(\mathcal{Q}) = \mathcal{N}(\mathbf{R})$.

In summary, if we set $\mathcal{S} \equiv \mathcal{R}(\mathcal{Q})$ and $\mathcal{T} \equiv \mathcal{R}(\mathcal{P})$, then the following relations can be stated:

$$\begin{aligned} \Omega_0 &= \mathcal{S} \oplus \mathcal{T} \\ \mathcal{S} &= \mathcal{R}(\mathcal{Q}) = \mathcal{N}(\mathcal{P}) = \mathcal{R}(\mathbf{P}) \\ \mathcal{T} &= \mathcal{N}(\mathcal{Q}) = \mathcal{R}(\mathcal{P}) = \mathcal{N}(\mathbf{R}). \end{aligned}$$

The subspaces $\mathcal{S}$ and $\mathcal{T}$ are very important when studying MG methods. Roughly speaking, one can think that the null space of $\mathcal{P}$ (the range of $\mathbf{P}$) is in some sense close to the space of smooth modes, since $\mathcal{P} = \mathbf{I} - \mathcal{Q}$ is constructed so that its desired action is is annihilating smooth components. In contrast $\mathcal{P}$, should let oscillatory components almost remain unchanged. Let $\mathbf{s}$ be a smooth mode and $\mathbf{t}$ an oscillatory mode. Then this statement translates as follows:

$$\mathcal{P}(\mathbf{s}) \approx \mathbf{0}, \quad \mathcal{P}(\mathbf{t}) \approx \mathbf{t}, \quad \mathcal{Q}(\mathbf{t}) \approx \mathbf{0}, \quad \text{and} \quad \mathcal{Q}(\mathbf{s}) \approx \mathbf{s}.$$

2.4.2 Convergence Analysis

For the Galerkin case, convergence is often analyzed using $\mathbf{A}$ in conjunction with 2-norms weighted by $\mathbf{D}^{1/2}$, or $\mathbf{D}^{-1/2}$, where $\mathbf{D}$ is the diagonal part of $\mathbf{A}$. The following notation is adopted:

$$\|\mathbf{x}\|_{\mathbf{D}} = \sqrt{\langle \mathbf{D}\mathbf{x}, \mathbf{x}\rangle} = \|\mathbf{D}^{1/2}\mathbf{x}\|_2.$$

The next norm is also important:

$$\|\mathbf{e}\|_{\mathbf{A}\mathbf{D}^{-1}\mathbf{A}} = \sqrt{\langle \mathbf{D}^{-1}\mathbf{A}\mathbf{e}, \mathbf{A}\mathbf{e}\rangle} \equiv \|\mathbf{A}\mathbf{e}\|_{\mathbf{D}^{-1}}.$$

To keep the notation simple, we will use $\|\mathbf{A}\mathbf{e}\|_{\mathbf{D}^{-1}}$ as notation for this particular norm of the error.

It can be shown that the Two-Grid cycle satisfies the next inequality.

Smoothing property: There exists some $\alpha > 0$ such that

$$\|\mathbf{S}\mathbf{e}\|_{\mathbf{A}}^2 \leq \|\mathbf{e}\|_{\mathbf{A}}^2 - \alpha\|\mathbf{A}\mathbf{e}\|_{\mathbf{D}^{-1}}^2 \quad \text{for all} \quad \mathbf{e} \in \Omega_1, \tag{2.23}$$

where $\mathbf{S} = \mathbf{I} - \mathbf{M}^{-1}\mathbf{A}$ is the action of the smoother.

The smoothing property is not enough; another assumption is needed to characterize the discretization.

Approximation property: There exists some $\beta > 0$ such that

$$\min_{\widehat{\mathbf{e}} \in \Omega_2} \|\mathbf{e} - \mathbf{P}\widehat{\mathbf{e}}\|_{\mathbf{D}}^2 \leq \beta\|\mathbf{e}\|_{\mathbf{A}}^2 \quad \text{for all} \quad \mathbf{e} \in \Omega_1, \tag{2.24}$$

where β is independent of h.

The next theorem is an adaptation of Theorem 13.3 in [18]. As usual we assume that $\mathbf{A}$ is symmetric positive definite, and that the inter-grid transfer operators satisfy $\mathbf{R}^T = \mathbf{P}$, with $\mathbf{P}$ being of full rank.

Theorem 2.5. *Suppose that inequalities (2.23) and (2.24) hold for a certain smoother, with* $0 < \alpha \leq \beta$. *Then*

$$\|\mathbf{S}\mathcal{P}\|_{\mathbf{A}} \leq \sqrt{1 - \frac{\alpha}{\beta}}.$$

Proof. Since $\mathcal{R}(\mathcal{P}) = \mathcal{T}$ is $\mathbf{A}$-orthogonal to $\mathcal{R}(\mathbf{P}) = \mathcal{S}$,

$$\langle \mathbf{e}, \mathbf{P}\widehat{\mathbf{e}}\rangle_{\mathbf{A}} = 0 \quad \text{for any} \quad \mathbf{e} \in \mathcal{T}, \mathbf{P}\widehat{\mathbf{e}} \in \mathcal{S}.$$

Rather than $\mathbf{P}\widehat{\mathbf{e}} \in \mathcal{S}$, we can take $\widehat{\mathbf{e}} \in \Omega_2$, then

$$\|\mathbf{e}\|_{\mathbf{A}}^2 = \langle \mathbf{e}, \mathbf{e} \rangle_{\mathbf{A}} - \langle \mathbf{e}, \mathbf{P}\widehat{\mathbf{e}} \rangle_{\mathbf{A}} = \langle \mathbf{Ae}, \mathbf{e} - \mathbf{P}\widehat{\mathbf{e}} \rangle \quad \text{for all} \quad \mathbf{e} \in \mathcal{T}, \widehat{\mathbf{e}} \in \Omega_2.$$

For any $\mathbf{e} \in \mathcal{T}$, using the Cauchy-Schwarz inequality,

$$\begin{aligned} \|\mathbf{e}\|_{\mathbf{A}}^2 &= \langle \mathbf{Ae}, \mathbf{D}^{-1/2}\mathbf{D}^{1/2}(\mathbf{e} - \mathbf{P}\widehat{\mathbf{e}}) \rangle \\ &= \langle \mathbf{D}^{-1/2}\mathbf{Ae}, \mathbf{D}^{1/2}(\mathbf{e} - \mathbf{P}\widehat{\mathbf{e}}) \rangle \\ &\leq \|\mathbf{D}^{-1/2}\mathbf{Ae}\|_2 \|\mathbf{D}^{1/2}(\mathbf{e} - \mathbf{P}\widehat{\mathbf{e}})\|_2 \\ &= \|\mathbf{Ae}\|_{\mathbf{D}^{-1}} \|\mathbf{e} - \mathbf{P}\widehat{\mathbf{e}}\|_{\mathbf{D}} \end{aligned}$$

Squaring and taking the minimum over all $\widehat{\mathbf{e}} \in \Omega_2$, the second inequality follows from the approximation property (2.24). We obtain

$$\|\mathbf{e}\|_{\mathbf{A}}^4 \leq \|\mathbf{Ae}\|_{\mathbf{D}^{-1}}^2 \min_{\widehat{\mathbf{e}} \in \Omega_2} \|\mathbf{e} - \mathbf{P}\widehat{\mathbf{e}}\|^2 \leq \|\mathbf{Ae}\|_{\mathbf{D}^{-1}}^2 \beta \|\mathbf{e}\|_{\mathbf{A}}^2.$$

Thus

$$\|\mathbf{e}\|_{\mathbf{A}} \leq \sqrt{\beta} \|\mathbf{Ae}\|_{\mathbf{D}^{-1}} \quad \text{for any} \quad \mathbf{e} \in \mathcal{T},$$

or equivalently

$$\frac{1}{\beta} \|\mathcal{P}\mathbf{e}\|_{\mathbf{A}}^2 \leq \|\mathbf{A}\mathcal{P}\mathbf{e}\|_{\mathbf{D}^{-1}}^2 \quad \text{for any} \quad \mathbf{e} \in \Omega_1.$$

Since $\mathcal{P}\mathbf{e} \in \mathcal{T} \subset \Omega_1$, by the smoothing property (2.23), we get

$$\|\mathbf{S}\mathcal{P}\mathbf{e}\|_{\mathbf{A}}^2 \leq \|\mathcal{P}\mathbf{e}\|_{\mathbf{A}}^2 - \alpha \|\mathbf{A}\mathcal{P}\mathbf{e}\|_{\mathbf{D}^{-1}}^2 \leq \|\mathcal{P}\mathbf{e}\|_{\mathbf{A}}^2 - \frac{\alpha}{\beta} \|\mathcal{P}\mathbf{e}\|_{\mathbf{A}}^2 \leq \left(1 - \frac{\alpha}{\beta}\right) \|\mathcal{P}\mathbf{e}\|_{\mathbf{A}}^2.$$

Since $\mathcal{P}$ is an $\mathbf{A}$-orthogonal projector, by the Cauchy-Schwarz inequality we get

$$\|\mathcal{P}\mathbf{e}\|_{\mathbf{A}}^2 = \langle \mathcal{P}\mathbf{e}, \mathcal{P}\mathbf{e} \rangle_{\mathbf{A}} = \langle \mathcal{P}^2\mathbf{e}, \mathbf{e} \rangle_{\mathbf{A}} \leq \|\mathcal{P}\mathbf{e}\|_{\mathbf{A}} \|\mathbf{e}\|_{\mathbf{A}}.$$

Therefore $\|\mathcal{P}\mathbf{e}\|_{\mathbf{A}} \leq \|\mathbf{e}\|_{\mathbf{A}}$, which implies $\dfrac{\|\mathbf{S}\mathcal{P}\mathbf{e}\|_{\mathbf{A}}}{\|\mathbf{e}\|_{\mathbf{A}}} \leq \sqrt{1 - \dfrac{\alpha}{\beta}}$ and so

$$\|\mathbf{S}\mathcal{P}\|_{\mathbf{A}} = \max_{\mathbf{e} \neq \mathbf{0}} \frac{\|\mathbf{S}\mathcal{P}\mathbf{e}\|_{\mathbf{A}}}{\|\mathbf{e}\|_{\mathbf{A}}} \leq \sqrt{1 - \frac{\alpha}{\beta}}.$$

■

Theorem 2.5 ensures convergence of the Two-Level iteration (Algorithm 4), provided the smoothing and approximation properties hold. In Algorithm 4 we only use the pre-smoothing, but for the Generalized Two-Level iteration (Algorithm 4) we added the post-smoothing. In general the pre/post-smoothing are different ($\mathbf{I} - \mathbf{M}^{-1}\mathbf{A}$ and $\mathbf{I} - \mathbf{M}^{-T}\mathbf{A}$). In this case we have the next corollary.

Corollary 2.1. *Suppose (2.23) is satisfied for the pre-smoother with constant α_1 and the post-smoother with constant α_2, and let $\alpha = \min\{\alpha_1, \alpha_2\}$. Suppose also that (2.24) is satisfied with constant β such that $0 < \alpha \leq \beta$. Then the generalized Two-Level iteration shown in Algorithm 5 converges and the norm of its operator defined by (2.20) is bounded by*

$$\|\mathcal{M}\|_{\mathbf{A}} < \sqrt{1 - \frac{\alpha}{\beta}}.$$

Proof. In the case that the pre/post-smoothing are different i.e., $\mathbf{M}^T \neq \mathbf{M}$, there are two nonnegative constants α_1 and α_2. Let us say $\mathbf{S}$ is the pre-smoother and $\bar{\mathbf{S}}$ is the post-smoother, such that

$$\|\mathbf{Se}\|_{\mathbf{A}}^2 \leq \|\mathbf{e}\|_{\mathbf{A}}^2 - \alpha_1 \|\mathbf{Ae}\|_{\mathbf{D}^{-1}}^2 \quad \text{for all} \quad \mathbf{e} \in \Omega_1,$$

and

$$\|\bar{\mathbf{S}}\mathbf{e}\|_{\mathbf{A}}^2 \leq \|\mathbf{e}\|_{\mathbf{A}}^2 - \alpha_2 \|\mathbf{Ae}\|_{\mathbf{D}^{-1}}^2 \quad \text{for all} \quad \mathbf{e} \in \Omega_1.$$

Since $\alpha = \min\{\alpha_1, \alpha_2\}$, then

$$\|\mathbf{Se}\|_{\mathbf{A}}^2 \leq \|\mathbf{e}\|_{\mathbf{A}}^2 - \alpha \|\mathbf{Ae}\|_{\mathbf{D}^{-1}}^2 \quad \text{and} \quad \|\bar{\mathbf{S}}\mathbf{e}\|_{\mathbf{A}}^2 \leq \|\mathbf{e}\|_{\mathbf{A}}^2 - \alpha \|\mathbf{Ae}\|_{\mathbf{D}^{-1}}^2 \text{ for all} \quad \mathbf{e} \in \Omega_0.$$

Taking $\mathbf{e} \neq \mathbf{0}$ in the smoothing property implies

$$\frac{\|\mathbf{Se}\|_{\mathbf{A}}^2}{\|\mathbf{e}\|_{\mathbf{A}}^2} \leq 1 - \alpha \frac{\|\mathbf{Ae}\|_{\mathbf{D}^{-1}}^2}{\|\mathbf{e}\|_{\mathbf{A}}^2}.$$

Since $\mathbf{A}$ is nonsingular, then $\alpha \dfrac{\|\mathbf{Ae}\|_{\mathbf{D}^{-1}}^2}{\|\mathbf{e}\|_{\mathbf{A}}^2} > 0$, thus

$$\frac{\|\mathbf{Se}\|_{\mathbf{A}}^2}{\|\mathbf{e}\|_{\mathbf{A}}^2} < 1.$$

Therefore, taking the maximum over all $\mathbf{e} \neq \mathbf{0}$, we get $\|\mathbf{S}\|_{\mathbf{A}} < 1$ and, similarly, $\|\bar{\mathbf{S}}\|_{\mathbf{A}} < 1$.

On the other hand, from the definition of $\mathcal{M}$ in (2.20) for $\nu > 0$, we get

$$\begin{aligned} \|\mathcal{M}\mathbf{e}\| &= \|\bar{\mathbf{S}}^{\nu}\mathcal{P}\mathbf{S}^{\nu}\mathbf{e}\|_{\mathbf{A}} \\ &\leq \|\bar{\mathbf{S}}^{\nu}\mathcal{P}\|_{\mathbf{A}} \|\mathbf{S}^{\nu}\mathbf{e}\|_{\mathbf{A}} \\ &\leq \|\bar{\mathbf{S}}\|_{\mathbf{A}}^{\nu-1} \|\bar{\mathbf{S}}\mathcal{P}\|_{\mathbf{A}} \|\mathbf{S}\|_{\mathbf{A}}^{\nu} \|\mathbf{e}\|_{\mathbf{A}} \\ &< \|\bar{\mathbf{S}}\mathcal{P}\|_{\mathbf{A}} \|\mathbf{e}\|_{\mathbf{A}}. \end{aligned}$$

In particular, for $\mathbf{e} \neq \mathbf{0}$, using Theorem 2.5 we get $\dfrac{\|\mathcal{M}\mathbf{e}\|_{\mathbf{A}}}{\|\mathbf{e}\|_{\mathbf{A}}} < \sqrt{1 - \dfrac{\alpha}{\beta}}$, and so

$$\|\mathcal{M}\|_{\mathbf{A}} = \max_{\mathbf{e} \neq \mathbf{0}} \frac{\|\bar{\mathbf{S}}\mathcal{P}\mathbf{e}\|_{\mathbf{A}}}{\|\mathbf{e}\|_{\mathbf{A}}} < \sqrt{1 - \frac{\alpha}{\beta}}.$$

Since $0 < \alpha \leq \beta$, then $\|\mathcal{M}\|_{\mathbf{A}} < 1$ and so the Two-Grid iteration converges. ∎

Chapter 3

Constructing the Coarse Level

Standard multigrid methods rely greatly on the geometry of the grid, but in real applications the geometry of the grid can be very complex. This complexity makes choosing a coarse grid difficult, especially if an unstructured mesh is used. Alternatively the $\mathbf{Ax} = \mathbf{b}$ problem might not derive from a PDE discretization at all, or the underlying mesh might be unavailable. In these cases, an algebraic multigrid method is often a compelling choice. There are different ways to construct the coarse level: graph-based methods like Ruge-Stüben, the Beck Corsener, and Boostrap AMG, as well as aggregation methods. We will focus on the latter class of methods. The primary difference between aggregation methods and graph-based methods is the philosophy by which the coarse grid is selected. In graph-based methods, the only factor considered when selecting coarse points is the number of strong connections. Aggregation methods also make use of connection strength, but only to form neighborhoods of points. These neighborhoods are then taken as a unit when selecting the coarse grid. Since our case of study is the linear system arising from the Poisson equation, $\mathbf{A}$ is symmetric positive definite throughout this chapter. Additional information can be found in [5], [9], and [14].

3.1 Standard Aggregation

Vaněk introduced aggregation methods for AMG in the early 1990s, although the underlying ideas are older. After their introduction, aggregation methods have evolved, producing many variants; most of them use the same idea for selecting the coarse grid. We present Vaněk's aggregation method introduced in [21], since it is considered the base method upon which other methods are designed. Let us start with some definitions required for a better explanation.

Definition 3.1 (Aggregate)**.** *Let Ω_h denote the fine level, which is an ordered set of elements whose set of indices is Λ_h. An aggregate is a subset of the elements of Ω_h. The i-th aggregate will be denoted $\mathcal{C}_i$ (or $\mathcal{C}[i]$ in Algorithm 10).*

The next definition is one of the most standard and perhaps the simplest metric for identifying strong connections.

Definition 3.2 (Strongly coupled neighborhood)**.** *Given $\varepsilon > 0$, define the strongly coupled (or strongly connected) neighborhood of the element i as*

$$\mathcal{N}_i(\varepsilon) = \left\{ j \in \Lambda_h \quad : \quad |\mathbf{A}_{i,j}| \geq \varepsilon \sqrt{|\mathbf{A}_{i,i}\mathbf{A}_{j,j}|} \right\}.$$

This definition is motivated by the fact that two nodes i and j are strongly coupled if $|\mathbf{A}_{i,j}|$ is relatively large compared with $\sqrt{|\mathbf{A}_{i,i}\mathbf{A}_{j,j}|}$, based on our choice of $\varepsilon \in (0, 1)$.

Remark 3.1. *Note that Definition 3.2 is not the only way to define strong connections. There are different approaches using different metrics, like the Ruge-Stüben approach and its posterior modifications. In this particular work, we will also use a different metric, but for simplicity let us start using this one.*

Note that all the elements in $\mathcal{N}_i(\varepsilon)$ are strongly coupled. It is common practice to construct a matrix to represent the strong coupling between elements, let us say $\mathbf{C}_h(\varepsilon) = [\mathbf{C}_{i,j}]$ defined by

$$\mathbf{C}_{i,j} = \begin{cases} 1, & |\mathbf{A}_{i,j}| \geq \varepsilon \sqrt{|\mathbf{A}_{i,i}\mathbf{A}_{j,j}|}; \\ \\ 0, & \text{otherwise.} \end{cases}$$

Note that $\mathbf{C}_h(\varepsilon)$ is an sparse matrix composed only by ones and zeros, thus it seems unnecessary to store the value of its entries. For sparse matrices several memory reductions can be made, depending on the distribution and quantity of nonzero entries. There are many different data structures to handle sparse matrices. General formats are divided into categories:

- Efficient modifications and updates. Here we have formats like DOK (dictionary of keys), LIL (list of lists), and COO (coordinate list).
- Efficient access and operations. Here we have CSR (compressed sparse row) and CSC (compressed sparse column).

Multigrid implementations usually use the second category.

A parallel algorithm to partially construct this matrix is shown in Algorithm 9. We say "partial" in the sense that we do not need to store the individual entries of $\mathbf{C}_h(\varepsilon)$; we only need the indices of the entries equal to one. Thus we can take advantage of the CSR format, where we use arrays to store the indices of the columns corresponding to the nonzero entries, the row indices are stored in a compressed format, and we just ignore the nonzero values, since all of them equal one.

Algorithm 9: Strong-Connectivity Matrix (`StrongGraph`)

input : A, $\varepsilon > 0$
output: C

allocate memory for C.rowStarts // Setting the $n_k + 1$ entries to 0
for $i \leftarrow 1$ **to** n_k **do** // For each node i in parallel
  $S_i \leftarrow 0$;
  for $j \in N_i$ **do** // For all neighbors of i
    if $|\mathbf{A}_{i,j}| \geq \varepsilon\sqrt{|\mathbf{A}_{i,i}\mathbf{A}_{j,j}|}$ **then** // If the neighbor j is strong
      $S_i \leftarrow S_i + 1$; // Increase the # of strong neighbors
  C.rowStarts$[i+1] \leftarrow S_i$;
for $i \leftarrow 2$ **to** $n_k + 1$ **do** // Cumulative sum
  C.rowStarts$[i] \leftarrow C$.rowStarts$[i] + C$.rowStarts$[i-1]$;
C.NNZ$\leftarrow C$.rowStarts$[n_k + 1]$; // Set the # of nonzero entries
allocate memory for C.cols and C.coefs // Set the C.NNZ entries to 0
for $i \leftarrow 1$ **to** n_k **do** // For each node i in parallel
  $id \leftarrow C$.rowStarts$[i]$ // Index into column array in CSR format
  for $j \in N_i$ **do** // For all neighbors of i
    if $|\mathbf{A}_{i,j}| \geq \varepsilon\sqrt{|\mathbf{A}_{i,i}\mathbf{A}_{j,j}|}$ **then** // If the neighbor i is strong
      C.cols$[id] \leftarrow j$ // Store the corresponding column index
      $id \leftarrow id + 1$;

All aggregation methods are based on the idea of clustering elements to construct neighborhoods of elements, which are called aggregates. Each aggregate will become a single element on the coarse level. There are many different ways of constructing aggregates depending on the problem and/or its geometric qualities. The method outlined here is the Vaněk approach.

This aggregation method works in three general phases after an initialization stage. In the initialization stage, a set $\mathcal{R} \subset \Lambda_h$ is assembled account for the elements that have been

aggregated, and to separate out any isolated nodes. The set $\mathcal{R}$ is defined by

$$\mathcal{R} = \{\ i \in \Lambda_h \quad : \quad \mathcal{N}_i(\varepsilon) \neq \{i\}\ \},$$

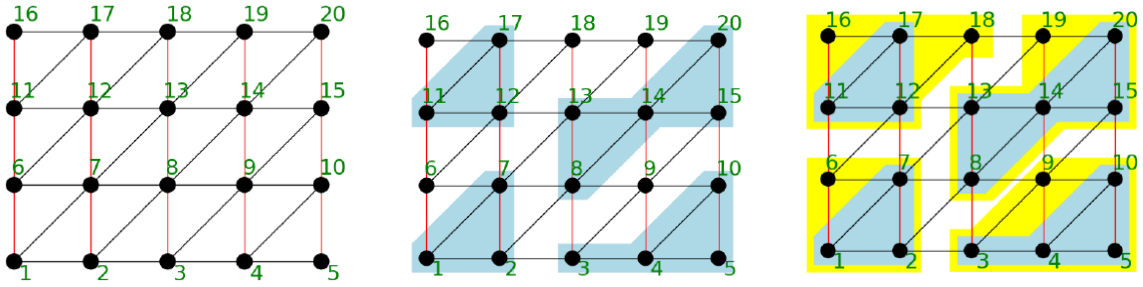
which is the set of all elements in Λ_h that are not isolated. Algorithm 10 provides the aggregation method that we will use. Phase one assembles the aggregates. The first element $i \in \mathcal{R}$ for which $\mathcal{N}_i(\varepsilon)$ is disjoint from all other aggregates becomes a new aggregate, and all the indices $\mathcal{N}_i(\varepsilon)$ are removed from $\mathcal{R}$. This process repeats until no more aggregates can be assembled, or when $\mathcal{R}$ is empty.

In the second phase the aggregates are expanded, including missing elements in phase one (elements remaining in $\mathcal{R}$) based on strong connections. Each remaining element in $i \in \mathcal{R}$ that has more than one strong connection is added to the most strongly connected aggregate, then i is removed from $\mathcal{R}$. If more than one aggregate is equally strongly connected to node i, the choice is arbitrary but consistent along all the phases.

In the third phase any remaining element is added to the most weakly connected aggregate. In most cases, this phase is not necessary since all elements are aggregated in the second phase.

To illustrate how Algorithm 10 works, consider an example from Cerwinsky in [5].

Example 3.1. *"Let Ω_h be the regular 4×5 grid shown in Figure 3.1(a). The grid points are numbered from the bottom left, row-wise. For this example the horizontal and diagonal connections are the strong connections. The vertical connections are weak."*



(a) (b) (c)

Figure 3.1: Example 3.1: (a) fine grid with strong connections shown in black; (b) Aggregates after the first phase; (c) final aggregates. Image taken from [5].

In this example $\mathcal{R}$ (the set of nodes to be aggregated) is given by

$$\mathcal{R} = \{\ j \in \mathbb{N} \quad : \quad 1 \leq j \leq 20\ \}.$$

Algorithm 10: Standard Aggregation (`FormAggregates`)

input : $\mathbf{C}$, $\varepsilon > 0$
output: $\mathcal{C}$

for $i \leftarrow 1$ **to** Λ_h **do** // Assemble strong neighborhoods
  $\mathcal{N}_i(\varepsilon) = \{\ j \in \Lambda\ :\ \mathbf{C}_{i,j} = 1\ \}$
$\mathcal{R} \leftarrow \{\ i \in \Lambda_h\ :\ \mathcal{N}_i(\varepsilon) \neq \{i\}\ \}$;
$n \leftarrow$ `length`$(\mathcal{R})$;
$k \leftarrow 0$
for $i \leftarrow 1$ **to** n **do** // First phase
  if $\mathcal{N}_{\mathcal{R}[i]}(\varepsilon) \subset \mathcal{R}$ **then** // If all the neighborhood is unaggregated
    $k \leftarrow k + 1$;
    $\mathcal{C}[k] \leftarrow \mathcal{N}_{\mathcal{R}[i]}(\varepsilon)$; // The neighborhood becomes a new aggregate
    $\mathcal{R} \leftarrow \mathcal{R} \setminus \mathcal{C}[k]$; // Remove the aggregated nodes
$n \leftarrow$ `length`$(\mathcal{R})$;
for $i \leftarrow 1$ **to** n **do** // Second phase
  $p \leftarrow -1$;
  for $j \leftarrow 1$ **to** k **do** // For each unaggregated node
    if $\mathcal{N}_{\mathcal{R}[i]}(\varepsilon) \cap \mathcal{C}[j] \neq \emptyset$ **then** // with strong connections
      if $p >$ `length`$(\mathcal{N}_{\mathcal{R}[i]}(\varepsilon) \cap \mathcal{C}[j])$ **then**
        $p \leftarrow j$ // Find the most strongly connected aggregate
  if $p > 0$ **then**
    $\mathcal{C}[p] \leftarrow \mathcal{C}[p] \cup \{\mathcal{R}[i]\}$; // Add the unaggregated node
    $\mathcal{R} \leftarrow \mathcal{R} \setminus \mathcal{R}[i]$ // Remove the aggregated node
$n \leftarrow$ `length`$(\mathcal{R})$;
for $i \leftarrow 1$ **to** n **do** // Thrird phase
  $k \leftarrow k + 1$;
  $\mathcal{C}[k] \leftarrow \mathcal{R} \cap \mathcal{N}_{\mathcal{R}[i]}(\varepsilon)$; // Aggregate the missing nodes
  $\mathcal{R} \leftarrow \mathcal{R} \setminus \mathcal{C}[k]$; // Remove the aggregated node

The first step chooses the initial aggregate; thus the first node (node 1) is chosen, and its strong neighborhood is set as the first aggregate $\mathcal{C}_1$. For the second aggregate, the next node in $\mathcal{R}$ whose strong neighborhood is disjoint from $\mathcal{C}_1$ is found. Thus, in this example, node 4 is chosen, and its strong neighborhood becomes the second aggregate. Following the process, the next node with all its strongly neighborhood disjoint from the previous aggregates is 11, and finally the node 14. Thus the strongly connected neighborhoods of nodes 11 and 14 become the last aggregates, as in Figure 3.1(b). At this point, we have four aggregates:

$$\mathcal{C}_1 = \{1, 2, 7\}, \quad \mathcal{C}_2 = \{3, 4, 5, 10\}, \quad \mathcal{C}_3 = \{11, 12, 17\}, \quad \text{and} \quad \mathcal{C}_4 = \{8, 13, 14, 15, 20\},$$

and the new set of unaggregated nodes is $\mathcal{R} = \{6, 9, 16, 18, 19\}$.

Note that node 6 is the first unaggregated node. The number of strong connections exhibit that node 6 is equally strongly connected to aggregates $\mathcal{C}_1$ and $\mathcal{C}_3$. To break the tie, we use the lower aggregate number; thus node 6 is added to $\mathcal{C}_1$. The next remaining node is node 9, which is strongly connected to the aggregates $\mathcal{C}_2$ and $\mathcal{C}_4$. Following the previous convention, node 9 is added to $\mathcal{C}_2$. We repeat the same process until all nodes are aggregated. The final aggregates are shown in Figure 3.1(c).

3.1.1 The Prolongation Operator

Once the aggregates have been selected, we should construct a corresponding prolongation matrix. This process differentiates all the aggregation methods. For example, in *Smoothed Aggregation* a preliminary or tentative prolongation operator is constructed according to the aggregates, and then we apply one smoothing step to it, creating the smooth prolongation operator; in *Rough Aggregation* the preliminary prolongation matrix is used directly as the rough prolongation operator. In particular cases, like small problems, rough prolongation matrices yield faster convergence.

To construct the prolongation operator, start with a preliminary prolongation matrix $\widehat{\mathbf{P}} = [\widehat{\mathbf{P}}_{i,j}]$ defined for each aggregate $\mathcal{C}_j$ via

$$\widehat{\mathbf{P}}_{i,j} = \begin{cases} 1, & i \in \mathcal{C}_j; \\ 0, & \text{otherwise.} \end{cases}$$

As we said before, the original rough aggregation uses $\widehat{\mathbf{P}}$ as the prolongation operator, while the smooth aggregation approach applies a smoothing step over $\widehat{\mathbf{P}}$, i.e., $(\mathbf{I} - \mathbf{M}^{-1}\mathbf{A})\widehat{\mathbf{P}}$.

Here $\mathbf{M}$ can be any of the matrices derived from the splitting methods ($\mathbf{A} = \mathbf{M} - \mathbf{N}$) in Chapter 2, as we show in Example 3.2.

Example 3.2. *Using Jacobi, damped Jacobi, Gauss-Seidel and SOR as the smoother yields the prolongation operators:*

$$\begin{aligned} \mathbf{P_J} &= (\mathbf{I} - \mathbf{D}^{-1}\mathbf{A})\widehat{\mathbf{P}}, \\ \mathbf{P}_{\omega\mathbf{J}} &= (\mathbf{I} - \omega\mathbf{D}^{-1}\mathbf{A})\widehat{\mathbf{P}}, \\ \mathbf{P_{GS}} &= (\mathbf{I} - (\mathbf{D} - \mathbf{L})^{-1}\mathbf{A})\widehat{\mathbf{P}}, \\ \mathbf{P_{SOR}} &= (\mathbf{I} - \omega(\mathbf{D} - \omega\mathbf{L})^{-1}\mathbf{A})\widehat{\mathbf{P}}. \end{aligned}$$

Here $\mathbf{D}$ *is the main diagonal of* $\mathbf{A}$, $\mathbf{L}$ *is the strictly lower triangular part of* $\mathbf{A}$, *and* ω *is the damping or relaxation parameter.*

3.2 Maximal Independent Set Aggregation

The sequential aggregation method discussed in the previous section produces aggregates based on strongly connected neighbors. Unfortunately Algorithm 10 is a greedy algorithm strongly dependent on how the nodes are sorted or labeled, which introduces sequential dependences making hard a possible parallelization. In this section we discuss a parallel approach based on a general *maximal independent set* (MIS) algorithm, which is expected to produce aggregates with similar properties to Algorithm 10.

Let us start by defining a graph and a maximal independent set of distance k. A graph is a pair $\mathbf{G} = (\mathbf{V}, \mathbf{E})$, where $\mathbf{V}$ is the set of vertices (or nodes), and $\mathbf{E}$ is the set of edges. An edge is a pair of two distinct vertices. Then, MIS(k) is defined in [1] as follow.

Definition 3.3 (MIS(k))**.** *Given a graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$, let $\mathbf{V}_{root} \subset \mathbf{V}$ be a set of root vertices, and denote by $d_{_G}(\cdot, \cdot)$ the length of the shortest path between two vertices in the graph. Then $\mathbf{V}_{root}$ is a maximal independent set of distance k, or MIS(k), if the following both hold.*

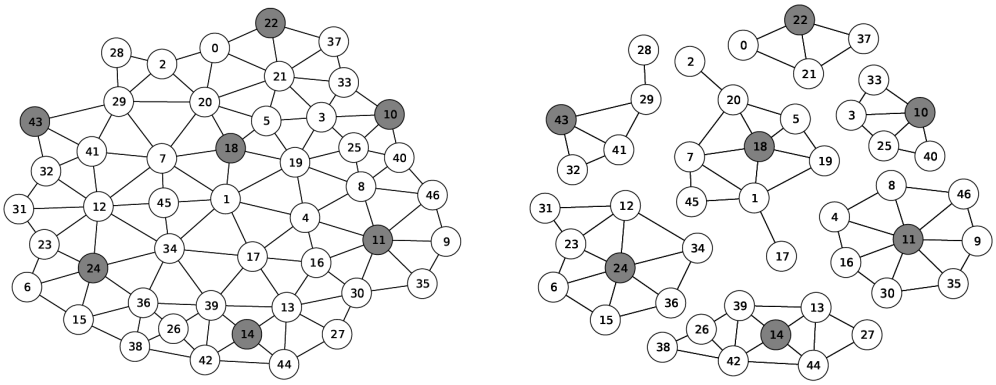
1. *Independence: Given any two vertices $u, v \in \mathbf{V}_{root}$, then $d_{_G}(u, v) > k$.*
2. *Maximality: There exists no $u \in \mathbf{V} \setminus \mathbf{V}_{root}$ such that $d_{_G}(u, v) > k$ for all $v \in \mathbf{V}_{root}$.*

In an attempt to imitate the geometry in standard multigrid methods, we consider the graph associated with the coefficient matrix $\mathbf{A}$, with some metric to construct the connectivity matrix $\mathbf{C}_h(\varepsilon)$ on which we will use the MIS(2) criterion to assemble the aggregates.

We will explain the algorithm deployed in `libparanumal`, an experimental set of finite element flow solvers for heterogeneous (GPU/CPU) systems. The initial development of `libparanumal` was performed by the Parallel Numerical Algorithms Group at Virginia Tech.

Given a valid MIS(2), the aggregates are constructed as follows: assume the MIS(2) is specified by an array of values $\{0, 1\}$, where a value of 1 means the corresponding node is a member of the MIS(2) and a value 0 otherwise. Since the k-th node in the MIS(2) serves as the root of the k-th aggregate, the only remaining task is to propagate the root indices outward. For each node j not in the MIS(2), we evaluate the first ring of nodes strongly connected to j. If there is a root node among these nodes, we will assign the node j to the most strongly connected root node. Then we evaluate the second ring of nodes that are strongly connected to j; then the node j is assigned to the aggregate whose root node is more strongly connected to j within the first and second ring of strongly connected nodes.

Note that `libparanumal`'s coarsener is based on the work of Gandham, Esler, and Zhang in [10], which is also based on the work of Bell, Dalton, and Olson in [1]. (The coarsener suggest by Bell et al. differs in the way nodes are aggregated.) Given the MIS(2), the root indices are communicated to their neighbors. In a first step each unaggregated node looks for an aggregate's index, in its first ring of strongly connected nodes; if there is an aggregate's index, the unaggregated node will take it (the node is aggregated). In the second step, unaggregated nodes will look for an aggregate's index amongst their neighbors; at this point, at least one such neighbor has been aggregated. As before, in the presence of ties, the choice is made arbitrarily.



(a) (b)

Figure 3.2: MIS(2): (*a*) the graph, with root nodes in gray; (*b*) aggregates built around root nodes. Image taken from [1].

This process can be thought as: given the root nodes defined by an MIS(2), the aggregation is made using the first ring of strongly connected nodes, rather than two rings as in `libparanumal`.

In essence, Figure 3.2 shows the initial attempts to obtain aggregates with similar properties to the ones generated by the standard aggregation approach as in Figure 3.1, but in parallel.

Another difference with the standard aggregation is the metric used. In this case, it is a variant of the strongly negatively coupled metric introduced by Ruge and Stüben in [14].

Definition 3.4. *Node i is strongly connected to j if*

$$-\mathbf{A}_{ij} \geq \varepsilon \left(\max_{k \neq i} \{-\mathbf{A}_{ik}\} \right) \qquad \textit{for some} \quad 0 < \varepsilon < 1.$$

Usually $\varepsilon = 0.25$ is used in practice.

Warburton and Gandham modified Definition 3.4 by normalizing it, in followup work to [9] in `libparanumal`. They define the strong connectivity matrix $\mathbf{C} = [\mathbf{C}_{i,j}]$ via

$$\mathbf{C}_{i,j} = \begin{cases} 1, & -\dfrac{s_i \mathbf{A}_{i,j}}{\sqrt{\mathbf{A}_{i,i}\mathbf{A}_{j,j}}} \geq \varepsilon \left(\max\limits_{k} \left\{ -\dfrac{s_i \mathbf{A}_{i,k}}{\sqrt{\mathbf{A}_{i,i}\mathbf{A}_{k,k}}} \right\} \right); \\ 0, & \text{otherwise.} \end{cases} \tag{3.1}$$

Here $s_i = \texttt{sign}(\mathbf{A}_{i,i})$ and $\mathbf{A}_{i,i}\mathbf{A}_{j,j} > 0$ since $\mathbf{A}$ is assumed to be positive definite.

In a practical way, there is no reason to construct explicitly the strong connectivity matrix $\mathbf{C}$; we only need to keep track of the indices where there is a strong connection. Thus `libparanumal` exploits the CSR matrix format to generate those indices in parallel via Algorithm 11. The implementation only constructs $\mathbf{C}$.`rowStarts` and $\mathbf{C}$.`cols`, since these two arrays contain the indices of the strong connections $\mathbf{C}_{i,j} = 1$.

In order to aggregate only strongly connected nodes, Algorithms 12 and 13 utilize the matrix $\mathbf{C}$ instead of $\mathbf{A}$. This algorithm resembles the algorithm given in [1]; additionally it incorporates the number of strong connections for robustness, as in [15].

Also note that Algorithms 12 and 13 rely on the comparison of tuples of the form

$$T = (state, value, index),$$

and thus we use a comparison function between these tuples called `CustomLess`. This function first compares the state of the node (aggregated or unaggregated), then the number of strong connections plus a random number to break possible ties, and finally the indices of the nodes.

In `libparanumal`, Algorithms 12 and 13 define the coarsener, so they are coded in one file, but to simplify the explanation of each algorithm, we describe them in two pieces.

i. Algorithm 12 is the first phase of the coarsener. Here, the MIS(2) set is built using some random numbers added to the strong connections, so that all possible ties are broken when the tuples are compared.

ii. Algorithm 13 uses the MIS(2) set from Algorithm 12, from which the aggregates are assembled using these root nodes, comparing the first and second ring of strongly connected nodes to each root node.

Algorithm 11: Strong graph connectivity (StrongGraph)

input : $\mathbf{A}$, $\varepsilon > 0$
output: $\mathbf{C}$

allocate memory for C.rowStarts // Setting the $n_k + 1$ entries to 0
for $i \leftarrow 1$ **to** n_k **do** // For each node i in parallel
  $s \leftarrow$ sign$(A_{i,i})$;
  $d \leftarrow 0$;
  for $j \in N_i$ **do** // For all neighbors of i
    if $-\frac{s\mathbf{A}_{i,j}}{\sqrt{\mathbf{A}_{i,i}\mathbf{A}_{i,j}}} \geq d$ **then** // Look for the max
      $d \leftarrow -\frac{s\mathbf{A}_{i,j}}{\sqrt{\mathbf{A}_{i,i}\mathbf{A}_{i,j}}}$ // Update the max
  for $j \in N_i$ **do** // For all neighbors of i
    $S_i \leftarrow 0$;
    if $-\frac{s\mathbf{A}_{i,j}}{\sqrt{\mathbf{A}_{i,i}\mathbf{A}_{i,j}}} \geq \varepsilon d$ **then** // If the neighbor j is strong
      $S_i \leftarrow S_i + 1$; // Increase the # of strong neighbors
  C.rowStarts$[i+1] \leftarrow S_i$;
for $i \leftarrow 2$ **to** $n_k + 1$ **do** // Cumulative sum
  C.rowStarts$[i] \leftarrow C$.rowStarts$[i] + C$.rowStarts$[i-1]$;
C.NNZ$\leftarrow C$.rowStarts$[n_k + 1]$; // Set the # of nonzero entries
allocate memory for C.cols and C.coefs // Setting the C.NNZ entries to 0
for $i \leftarrow 1$ **to** n_k **do** // For each node i in parallel
  $id \leftarrow C$.rowStarts$[i]$ // indexing into column array in CSR format
  for $j \in N_i$ **do** // For all neighbors of i
    if $-\frac{s\mathbf{A}_{i,j}}{\sqrt{\mathbf{A}_{i,i}\mathbf{A}_{i,j}}} \geq \varepsilon d$ **then** // If the neighbor i is strong
      C.cols$[id] \leftarrow j$ // Store the corresponding column index
      $id \leftarrow id + 1$;

Algorithm 12: Parallel Maximal Independent Set of distance 2 (`MIS(2)`)

input : $\mathbf{C}$ (CSR matrix) , *Level* (amgLevel)
output: $\mathbf{s}$

$GSR \leftarrow Level$.globalRowStarts;
$N \leftarrow \mathbf{C}$.Ncols;
allocate memory for $\mathbf{s}, \mathbf{r}$; `// Setting the` N `entries to 0`
for $i \leftarrow 1$ **to** N **do** `// For each column` i
  $r[i] \leftarrow$ `random( )` ; `// generates random numbers in` $[0.0, 1.0]$
for $i \leftarrow 1$ **to** $\mathbf{C}.NNZ$ **do** `// For each node` i `in parallel`
  $r[\mathbf{C}.\text{cols}[i]] \leftarrow r[\mathbf{C}.\text{cols}[i]] + 1$; `// add the # of strong neighbors`
$done \leftarrow 0$;
while $done = 0$ **do**
  for $i \leftarrow 1$ **to** N **do** `// For each column` i
    $T_{max} \equiv (s_{max}, r_{max}, i_{max}) \leftarrow (s[i], r[i], i)$; `// initialize tuples`
    if $s_{max} \neq 1$ **then**
      for $j \leftarrow \mathbf{C}.rowStarts[i] + 1$ **to** $\mathbf{C}.rowStarts[i+1]$ **do** `// First comparison`
        $col \leftarrow \mathbf{C}.\text{Cols}[j]$;
        $T \leftarrow (s[col], r[col], col + GRS[rank])$;
        if `CustomLess(`T_{max}, T`)` **then** `// Compare with strong neighbors`
          $T_{max} \leftarrow T$; `// Update` T_{max}
    $\widehat{T}[i] \leftarrow T_{max}$; `// Store` T_{max} `corresponding to` i
  for $i \leftarrow 1$ **to** N **do** `// For each column` i
    $T_{max} \leftarrow \widehat{T}[i]$; `// update` T_{max}
    for $j \leftarrow \mathbf{C}.rowStarts[i] + 1$ **to** $\mathbf{C}.rowStarts[i+1]$ **do** `// Second comparison`
      $col \leftarrow \mathbf{C}.\text{Cols}[j]$;
      if `CustomLess(`$T_{max}, \widehat{T}[col]$`)` **then** `// Compare with strong neighbors`
        $T_{max} \leftarrow \widehat{T}[col]$; `// Update` T_{max}
    if $s[i] = 0$ **and** $i_{max} = i + GRS[rank]$ **then** `// If I am strongest`
      $s[i] \leftarrow 1$; `// mark as MIS node`
    if $s[i] = 0$ **and** $s_{max} = 1$ **then** `// If there is another MIS node`
      $s[i] \leftarrow -1$; `// I am removed`
  $cnt \leftarrow$`count(`$s, s + n_k, 0$`)`;
  if $cnt > 0$ **then** `// If there are unassigned nodes`
    $done = 0$; `// keep working`
  else
    $done = 1$;

Algorithm 13: From Fine to Coarse Grid (`FormAggregates`)

input : $\mathbf{C}$, r, GSR, s (MIS(2))
output: $FineToCoarse$

for $i \leftarrow 0$ **to** N **do**
  if $s[i] = 1$ **then** // Count the # of aggregates
    $numAggs \leftarrow numAggs + 1$;
$level$.globalAggStarts[0] $\leftarrow 0$;
for $r \leftarrow 0$ **to** $Size$ **do**
  $level$.globalAggStarts[$r+1$] $\leftarrow$ $level$.globalAggStarts[r] + $gNumAggs[r]$;
$numAggs \leftarrow 0$;
for $i \leftarrow 0$ **to** N **do** // Enumerate aggregates
  if $s[i] = 1$ **then** // Count the # of aggregates
    $FineToCoarse[i] \leftarrow level$.globalAggStarts[$rank$] + $numAggs$;
    $numAggs \leftarrow numAggs + 1$;
for $i \leftarrow 0$ **to** N **do** // Form aggregates
  $T_{max} \leftarrow (s[i], r[i], i + global + GSR[rank])$; // Initialize tuples
  $c_{max} \leftarrow FineToGrid[i]$; // Initialize aggregate label
  if $s_{max} \neq 1$ **then** // For unaggregated nodes
    for $j \leftarrow \mathbf{C}.rowStarts[i] + 1$ **to** $\mathbf{C}.rowsStarts[i+1]$ **do** // First ring
      $col \leftarrow \mathbf{C}$.Cols[$j$];
      $T \leftarrow (s[col], r[col], col + GRS[rank])$;
      if `CustomLess`(T_{max}, T) **then** // Compare tuples
        $T_{max} \leftarrow T$; // Update tuples
        $c_{max} \leftarrow FineToCoarse[col]$; // Update the aggregate label
  $\widehat{T}[i] \leftarrow T_{max}$; $Tc[i] \leftarrow c_{max}$;
  if $s[i] = -1$ **and** $s_{max} = 1$ **and** $Tc[i] > -1$ **then** // If a strong root is found
    $FineToCoarse[i] \leftarrow Tc[i]$; // Update the aggregate label
for $i \leftarrow 0$ **to** N **do**
  $T_{max} \leftarrow \widehat{T}[i]$; $c_{max} \leftarrow Tc[i]$;
  for $j \leftarrow \mathbf{C}.rowStarts[i] + 1$ **to** $\mathbf{C}.rowStarts[i+1]$ **do** // Second ring
    $col \leftarrow \mathbf{C}$.Cols[$j$];
    $T \leftarrow \widehat{T}[col]$;
    if `CustomLess`(T_{max}, T) **then** // Compare tuples
      $T_{max} \leftarrow T$; $c_{max} \leftarrow Tc[col]$; // Update tuples and aggregate label
  if $s[i] = -1$ **and** $s_{max} = 1$ **and** $c_{max} > -1$ **then**
    $FineToCoarse[i] \leftarrow c_{max}$; // Update the final aggregate label

3.2.1 The Prolongation Operation

The prolongation or interpolation is defined from the aggregates and the "near null space" vectors $\mathbf{B}_k$ (which may represent low eigenmodes of the problem) of the linear operator $\mathbf{A}_k$. Suppose

$$\mathbf{A}_k = \sum_{j=1}^{N_k} \lambda_j \mathbf{v}_j \mathbf{v}_j^T$$

is a spectral decomposition of $\mathbf{A}_k$ with $0 < \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_{N_k}$. One might set $\mathbf{B}_k = [\ \mathbf{v}_1\ ]$ or $\mathbf{B}_k = [\ \mathbf{v}_1\ \mathbf{v}_2\ ]$ or similar. The library `libparanumal` uses only one near null space vector in the construction of the prolongation operator. This yields exactly one nonzero entry per row in the interpolation matrix $\mathbf{P}_k$, which is given by

$$\mathbf{B}_k = \mathbf{P}_k \mathbf{B}_{k+1}, \quad \mathbf{P}_k^T \mathbf{P}_k = \mathbf{I}.$$

This construction ensures the near null space vector $\mathbf{B}_k$ is in the range space of $\mathbf{P}_k$ and the interpolation matrix $\mathbf{P}_k$ has orthonormal columns. This is made by updating the unique nonzero entry per row in $\mathbf{P}_k$ with the corresponding values in $\mathbf{B}_k$, and then normalizing the columns of $\mathbf{P}_k$, as shown in Algorithm 14.

Algorithm 14: Prolongation (`Prolong`)

input : $FineToCoarse$, $\mathbf{B}_k$
output: $\mathbf{P}_k$, $\mathbf{B}_{k+1}$

for $i \leftarrow 1$ **to** n_{k+1} **do**
  $I \leftarrow$ `find`$(FineToCoarse = i)$; `// Get the nodes in the aggregate`
  $\mathbf{P}_k(I, i) \leftarrow \mathbf{B}_k(I)$; `// Copy null space vector entries`

for $i \leftarrow 1$ **to** n_{k+1} **do**
  $I \leftarrow$ `find`$(FineToCoarse = i)$; `// Get the nodes in the aggregate`
  $\mathbf{B}_{k+1}(i) \leftarrow \mathbf{B}_k(I)^T \mathbf{B}_k(I)$;
  $\mathbf{B}_{k+1}(i) \leftarrow \sqrt{\mathbf{B}_{k+1}(i)}$; `//` $\mathbf{B}_{k+1}(i) \leftarrow \|\mathbf{B}_k(I)\|_2$

for $i \leftarrow 1$ **to** n_{k+1} **do**
  $I \leftarrow$ `find`$(FineToCoarse = i)$; `// Get the nodes in the aggregate`
  $\mathbf{P}_k(I, i) \leftarrow \mathbf{P}_k(I, i)/\mathbf{B}_{k+1}(i)$; `// Normalize the prolongation operator`

Algorithm 14 provides $\mathbf{P}_k$ and $\mathbf{B}_{k+1}$. The prolongation operator constructed via this algorithm has the same sparsity pattern as in the classic rough aggregation approach, but rather than ones, here we copy the entry values from the corresponding entries in $\mathbf{B}_k$. This update in $\mathbf{P}_k$ can improve the convergence rate in the rough aggregation schema.

3.3 Locally partial strong connected nodes

Here we will discuss some characteristics of the rough aggregation via MIS(2) and the standard smooth aggregation view in the last two sections, and how we can combine the best of those two approaches to create a new approach keeping the essence of the standard smooth aggregation (Algorithm 10), but in parallel. Note that the algorithm used in `libparanumal` (Algorithm 13) ignores the idea of neighborhood, which is the foundation of the standard aggregation schema; also, Algorithm 13 could generate aggregates consisting of only one node, yielding slow convergence. This will be explained by the end of this section.

Let us start by summarizing some of the key points in MIS(2) and the standard aggregation.

Maximal Independent Set (2)	Standard aggregation
• The aggregates are constructed node-by-node based on the strong connections + random numbers (to break ties), then the global indices. • First select the root nodes that are strongest among all the ranks. Then each rank constructs its aggregates using its corresponding root nodes. • There are many `ogsGatherScatter` instructions to share information about the non-local information about the nodes. • The aggregates are not local; one aggregate could be in more than one rank and the strong neighborhoods from the first phase could be broken in the second phase.	• This is a greedy algorithm, highly dependent on the order of the nodes; this makes it difficult to implement in parallel. • The aggregates are constructed neighborhood-by-neighborhood, following the natural indices of the nodes. • The algorithm prioritizes strong neighborhoods, which become aggregates in the first phase. • In the second phase, nodes strongly connected to an aggregate are incorporated to that aggregate in a consistent way. • In the third phase, any missing node is aggregated using weak connections.

Of course, the standard aggregation approach is indeed a serial algorithm, while MIS(2) is supposed to be a roughly equivalent but parallel algorithm to generate aggregates with similar properties to the standard aggregation, under some relabeling.

As we said before, MIS(2) (Algorithm 12) generates the root nodes for the aggregates, but Algorithm 13 differs from the original expectations since some nodes strongly connected to a root node can be associated with a different root node in the second ring if they have stronger connections. This makes sense if we only prioritize the strong connections between nodes, but this goes against the essence of the standard aggregation schema.

To illustrate this unexpected behavior in `libparanumal`, consider the next example.

Example 3.3. *Let us apply Algorithms 12 and 13 with* $\varepsilon = 0.5$ *to a circular mesh* (`CircleH05`). *Figure 3.3 shows the mesh and strong connections, while Figure 3.4 shows the obtained root nodes and aggregates.*

Note the following aspects of Figure 3.4 (bottom).

- There are some aggregates formed by only one node, such as aggregates 4, 13, and 18.
- There are some aggregates with only two nodes, while other have more than 10 nodes.
- There are some aggregates with nodes that are not strongly connected, such as aggregates 3, 5 and 7.
- In particular, the root nodes 4 and 7 seem to be too close: since those two nodes are vertices of the same triangle, the physical distance between them is 1, but the edge connecting those vertices is not a strong connection, so the distance between them is 2; this shows that ε and the metric used to define the strong connections is important.

Now let us look at the strong connectivity matrix corresponding to the strong connections shown in Figure 3.3 (bottom). Here we have 194 degrees of freedom after applying the boundary conditions. The nonzero pattern of the strong connectivity matrix $\mathbf{C} \in \mathbb{R}^{194\times 194}$ is shown in Figure 3.5(a).

Indeed the construction of the strong conenectivity matrix $\mathbf{C}$ in `libparanumal` inherits the lack of symmetry from Definition 3.4, as seen in Figure 3.5(b). On the other hand, Definition 3.2 would yield a symmetric matrix. The lack of symmetry is not the real problem, but since the aggregation approaches are closely related to graph theory, we would prefer symmetric strong connectivity matrices.

If we change the metric used in `libparanumal` to the one given in Definition 3.2, will the problem be fixed?

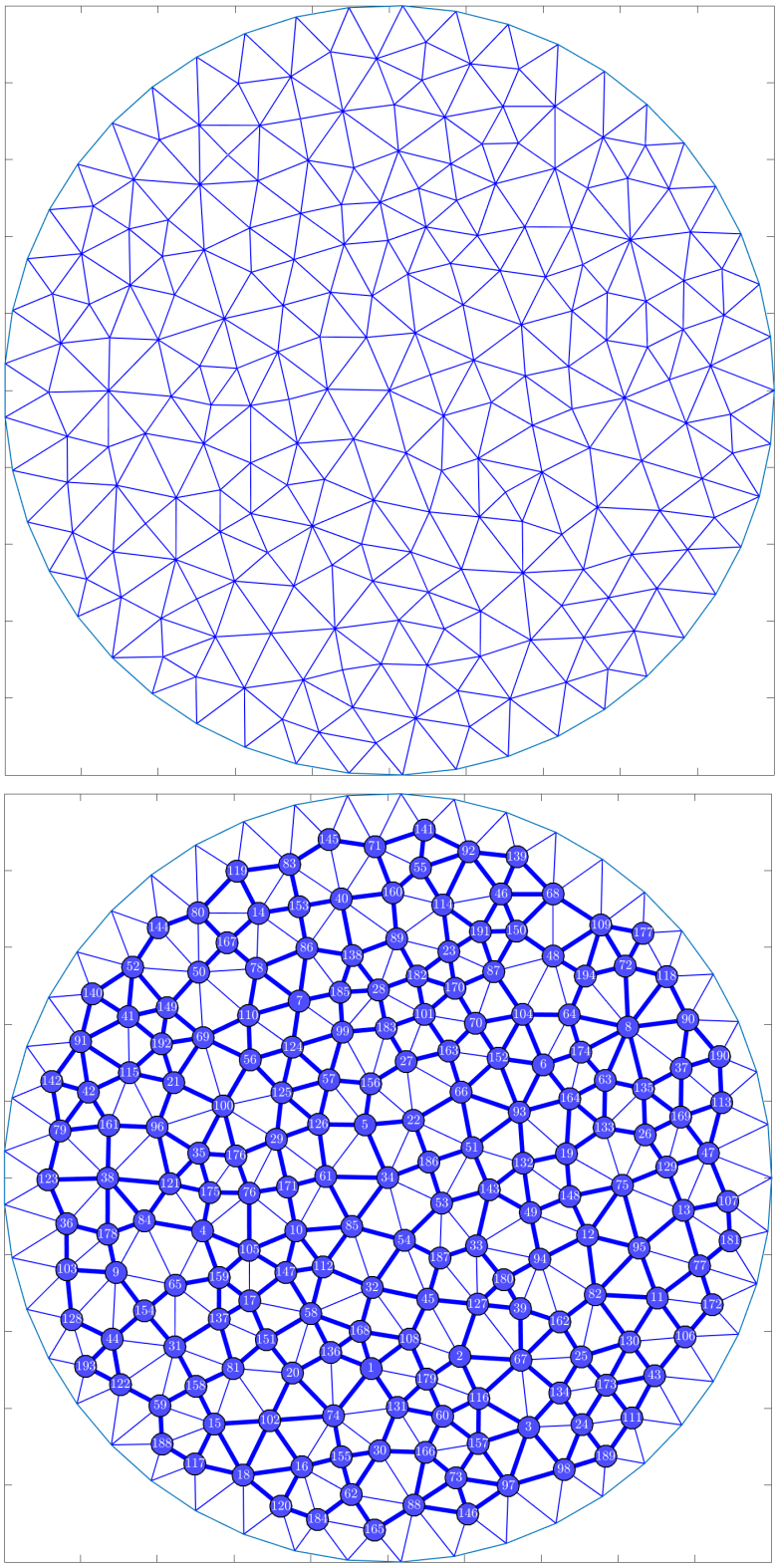


Figure 3.3: Example 3.3: (Top) mesh: CircleH05.msh; (Bottom) nodes and strong connections in thick edges.

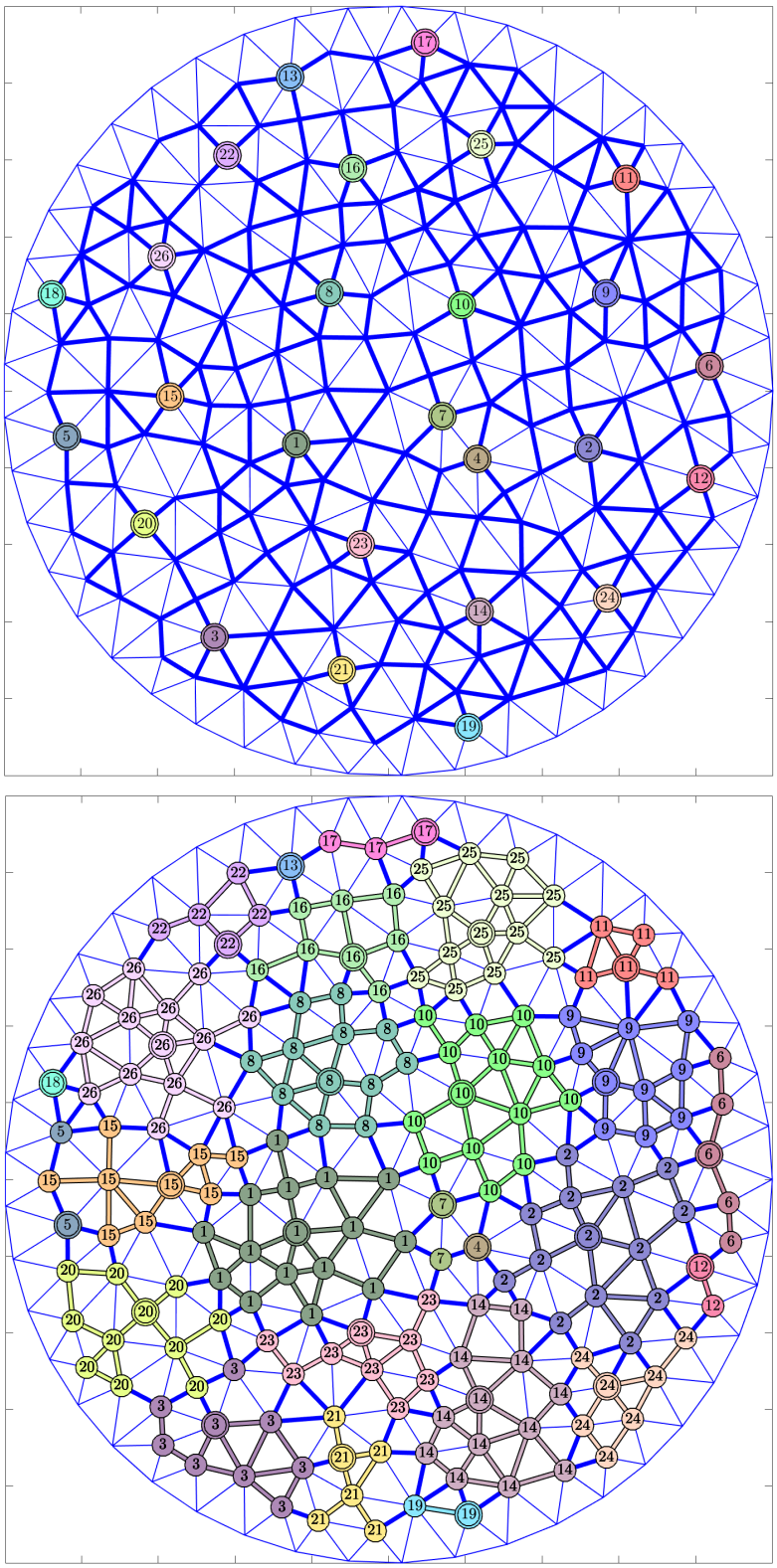
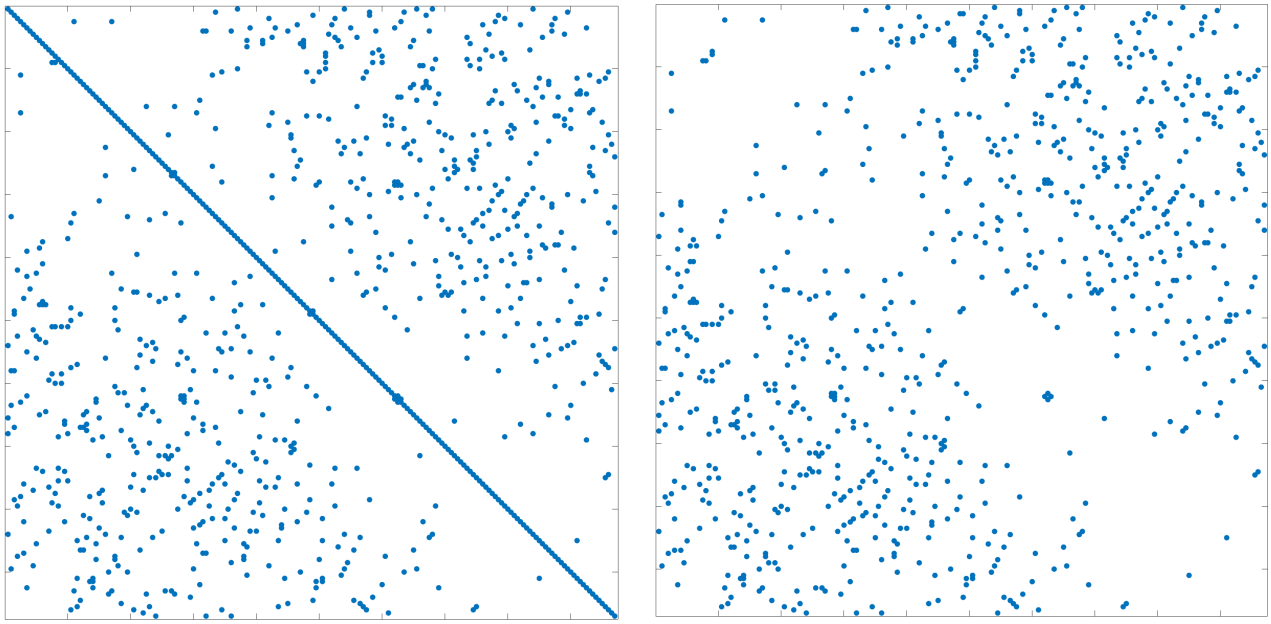


Figure 3.4: MIS(2): (Top) root nodes, double circles, generated by Algorithm 12; (Bottom) 26 aggregates, shown by color and number, generated by Algorithm 13.



(*a*) $\texttt{spy}(\mathbf{C})$ (*b*) $\texttt{spy}(\mathbf{C} - \mathbf{C}^T)$

Figure 3.5: Sparsity pattern: (*a*) strong connectivity matrix $\mathbf{C}$ used to construct the aggregates in Figure 3.4; (*b*) $\mathbf{C} - \mathbf{C}^T$, illustrating the lack of symmetry in $\mathbf{C}$.

Figure 3.6 shows that combining the standard metric, Definition 3.2, with Algorithm 13 is not a good idea. Using $\varepsilon = 0.5$, we obtained 30 aggregates, with more aggregates consisting of only one node. We know that the standard metric is usually more strict, in the sense that there are fewer strong connections for ε insufficiently small, but $\varepsilon = 0.25$ produces 18 aggregates. This shows that decreasing ε decreases the number of aggregates, since there are more strong connections, but rather than solving the issue, this makes it worse: it leads to a sparser strong connectivity matrix $\mathbf{C}$, and so slower convergence.

Since the aggregates shown in Figures 3.4 and 3.6 do not look adequate, a natural question is, what do the aggregates produced by Algorithm 10, the standard aggregation approach, look like?

Figure 3.7 shows the aggregates generated by Algorithm 10. We note that $\varepsilon = 0.5$ is too strict a threshold parameter, since it produces isolated nodes (no strong connections) that are aggregated at the third phase, looking at the weak connections, yielding 31 aggregates. On the other hand, $\varepsilon = 0.25$ produces 25 aggregates with no isolated nodes.

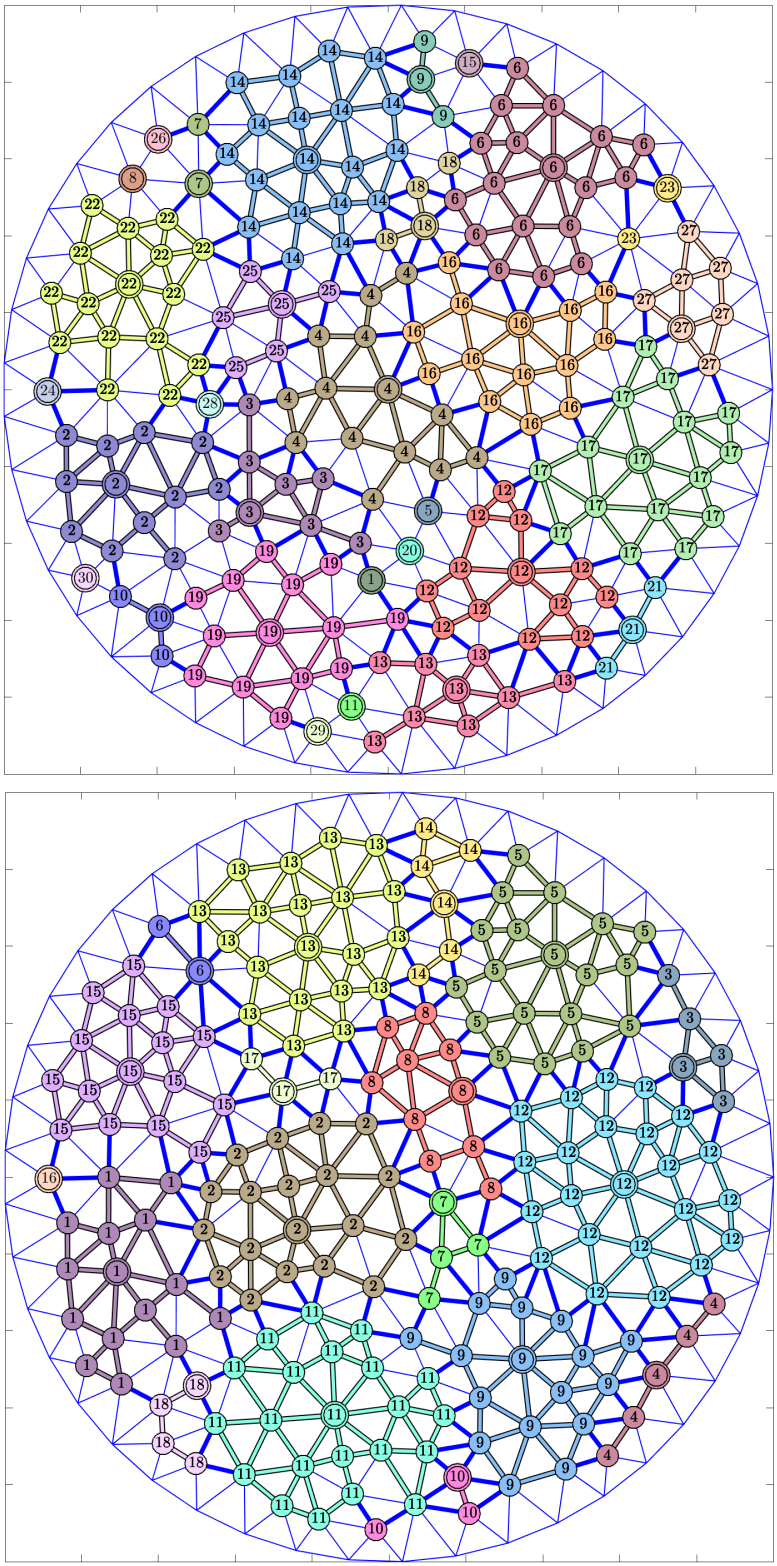


Figure 3.6: Aggregates generated by MIS(2) with Definition 3.2: (Top) $\varepsilon = 0.5$; (Bottom) $\varepsilon = 0.25$.

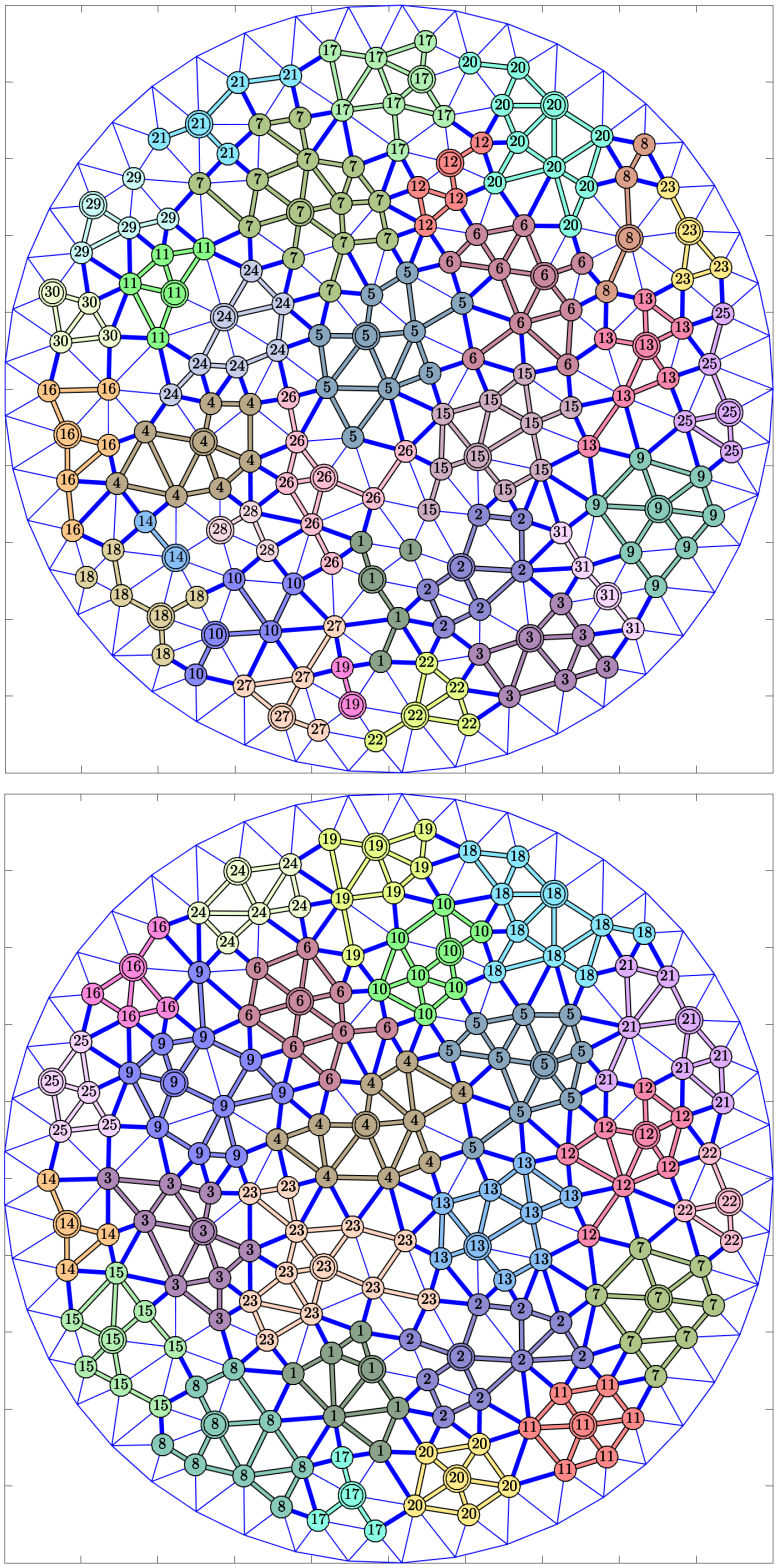


Figure 3.7: Aggregates generated by Algorithm 10: (Top) $\varepsilon = 0.5$; (Bottom) $\varepsilon = 0.25$.

We would like to generate aggregates similar to the ones shown in Figure 3.7 (bottom), but without relying on the greedy sequential algorithm. Thus we seek to keep the best ideas of each approach.

- Since the ultimate goal is run in parallel, MIS(2) seems to be the clear choice for the first step towards the construction of aggregates in parallel.
- Each rank constructs its own aggregates, but rather than doing so node-by-node, do it neighborhood-by-neighborhood, using the root nodes from MIS(2).
- Since there could be neighborhoods that are scattered in more than one rank, we can gather-scatter information between ranks, or just construct the aggregates locally.
- The first phase will construct aggregates around the root nodes. Since the minimal strong distance between two root nodes is 2, there could be overlapping strong neighborhoods for two root nodes, thus we should be consistent with the order in which the root nodes are selected.
- In a second phase, if there are missing nodes that have strong connections, they are added to an aggregate that has more strong connections. If there is a tie, then choose the aggregate with fewer nodes (in an attempt to keep the aggregate size homogeneous).
- Isolated nodes, i.e., nodes with no strong connections to other nodes, will be aggregated based on weak connections.

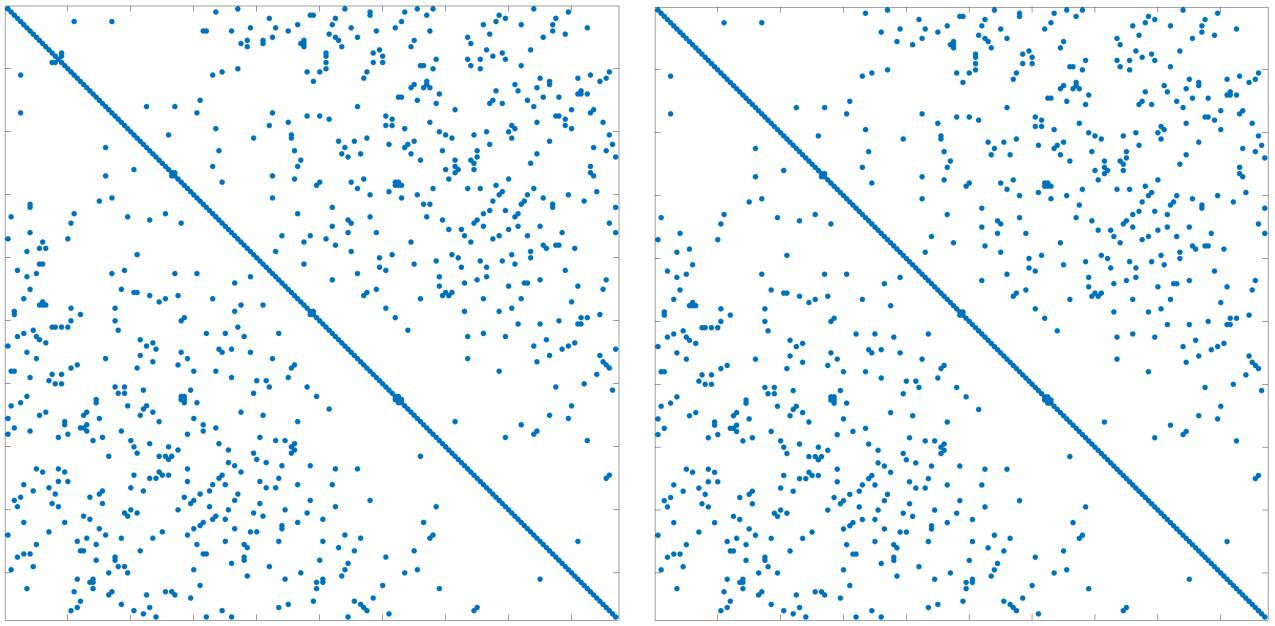
On the physical level (original mesh) it is quite unusual to get isolated nodes unless we are using a bad combination of ε and metric, as in Figure 3.7 (top). Working locally in parallel and at coarser mesh levels, this is likely to happen.

Another characteristic that can be improved is the metric. The standard metric could be too strict, and the one used in `libparanumal` is not symmetric. Thus we modify the metric in `libparanumal` a little bit to make it symmetric.

Definition 3.5. *The node i is strongly connected to j, if*

$$\frac{|\mathbf{A}_{i,j}|}{\sqrt{|\mathbf{A}_{i,i}\mathbf{A}_{j,j}|}} \geq \frac{\varepsilon}{2}\left(\max_k\left\{\frac{|\mathbf{A}_{i,k}|}{\sqrt{|\mathbf{A}_{i,i}\mathbf{A}_{k,k}|}}\right\}+\max_k\left\{\frac{|\mathbf{A}_{j,k}|}{\sqrt{|\mathbf{A}_{j,j}\mathbf{A}_{k,k}|}}\right\}\right)$$

Let $\widehat{\mathbf{C}}$ be the strong connectivity matrix generated by Definition 3.5. Why should we use $\widehat{\mathbf{C}}$, rather than symmetrize $\mathbf{C}$ at the matrix level by $\widetilde{\mathbf{C}} = \frac{1}{2}(\mathbf{C}+\mathbf{C}^T)$?



(*a*) spy($\widetilde{\mathbf{C}}$) (*b*) spy($\widehat{\mathbf{C}}$)

Figure 3.8: Sparsity pattern: (*a*) $\widetilde{\mathbf{C}}$ with 932 nonzero entries; (*b*) $\widehat{\mathbf{C}}$ with 972 nonzero entries.

Figure 3.8 compares the sparsity patterns of $\widetilde{\mathbf{C}}$ and $\widehat{\mathbf{C}}$ for Example 3.3. Here we note that $\widehat{\mathbf{C}}$ has 932 nonzero entries, while $\widetilde{\mathbf{C}}$ has 972 nonzero entries; this difference in the number of strong connections affects the final number of aggregates produced by each approach. In particular, $\widehat{\mathbf{C}}$ will produce roughly the same number of aggregates as $\mathbf{C}$ from `libparanumal` (26 aggregates), while $\widetilde{\mathbf{C}}$ is more aggressive, producing fewer aggregates (25 aggregates) and thus yielding, in general, a slower convergence. This difference might seem small for this small example, but one observes a much larger difference for the size of example we are most interested in solving.

Figure 3.9 shows the aggregates obtained using $\widetilde{\mathbf{C}}$ (top) and $\widehat{\mathbf{C}}$ (bottom). In the aggregates generated by both connectivity matrices, we still can see that there are aggregates consisting of only one node. At this point we should notice that this problem cannot be solved by modifying the metric used to define strong connections. This is the reason why we decided to use a new approach to assemble the aggregates. As we said before, to prevent aggregates consisting of only one node, we should preserve the strong neighborhood around the root node. When we are aggregating nodes that are not in the strong neighborhood of a root node, we will primarily use the number of strong connections. To break possible ties, we use the size of the aggregate; secondly, we use the number of weak connections, if there are no strong connections with any aggregate.

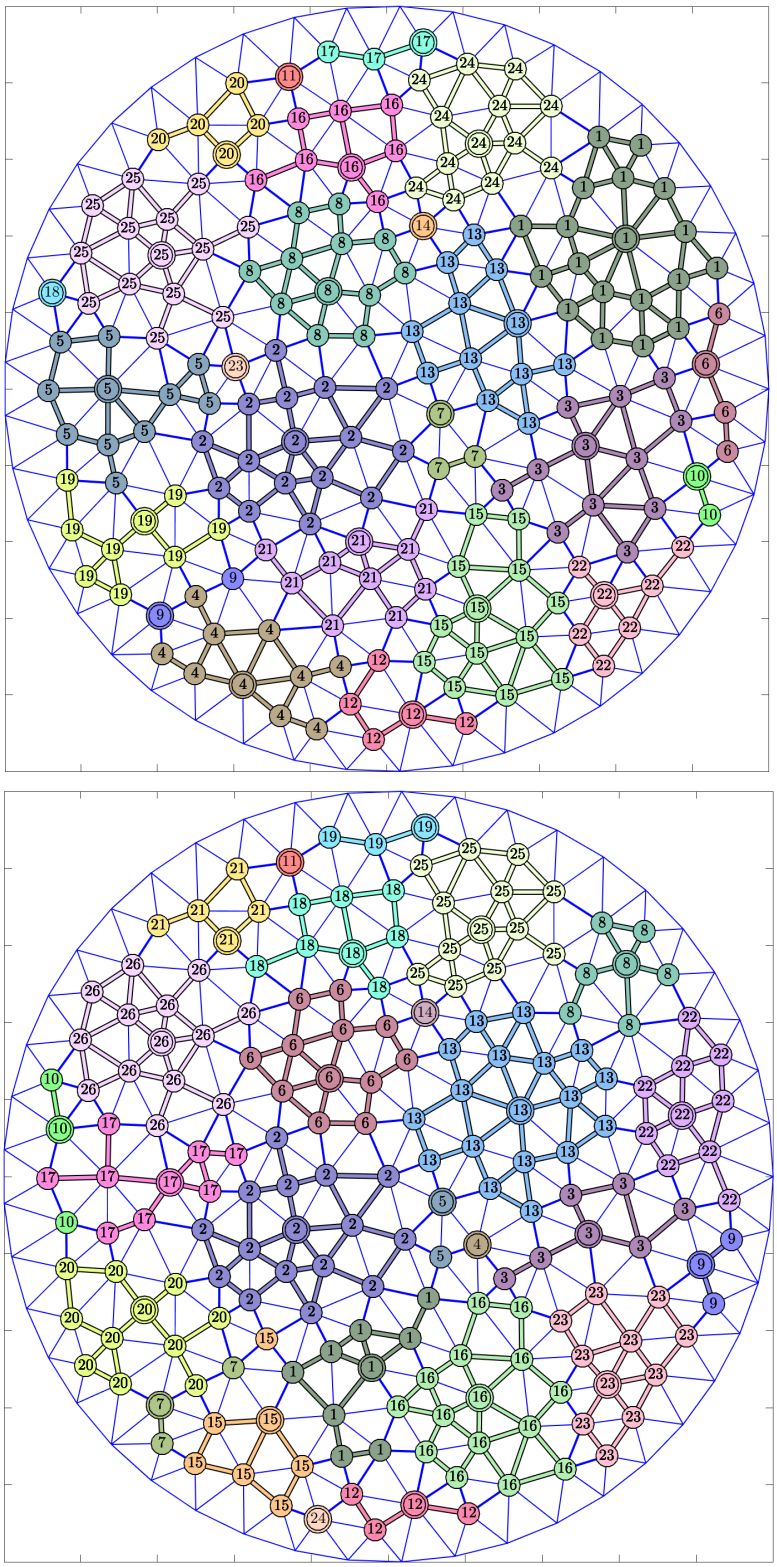


Figure 3.9: Aggregates generated by Algorithm 10: (Top) using $\widetilde{\mathbf{C}}$; (Bottom) using $\widehat{\mathbf{C}}$.

Since the MIS(2) provides a good choice of root nodes whose strong neighborhoods hardly ever overlap, we keep using Algorithm 12 to generate the root nodes. Once the MIS(2) is constructed using the metric given by Definition 3.5, we are ready to assemble the aggregate, and so we apply Algorithm 15 to generate the final aggregates.

Algorithm 15: Locally partial strong connected nodes (`LPSCN`)

input : $\mathbf{s}$ (MIS(2))
output: $FineToCoarse$

$FineToCoarse \leftarrow \{-1, -1, \ldots, -1\}$;
$numAgg \leftarrow 0$;
for $i \leftarrow 1$ **to** N **do** `// Label root nodes`
  if $\mathbf{s}(i) = 1$ **then**
    $FineToCoarse(i) \leftarrow numAgg + 1$;
    $numAgg \leftarrow numAgg + 1$;
$\mathbf{R} \leftarrow$ `find`$(\mathbf{s} > 0)$; `// Get root nodes`
for $i \leftarrow 1$ **to** `length`$(\mathbf{R})$ **do** `// First phase`
  for $j \in$ `StrongNeighborhood`$(\mathbf{R}(i))$ **do** `// Get the strong neighborhood`
    if $\mathbf{R}(i) = j$ **or** $\mathbf{s}(\mathbf{R}(i)) = 1$ **then**
      continue;
    else
      $FineToCoarse(j) \leftarrow FineToCoarse(\mathbf{R}(i))$; `// Add` j `to the aggregate`
$\mathbf{R} \leftarrow$ `find`$(FineToCoarse < 0)$; `// Get unaggregated nodes`
for $i \leftarrow 1$ **to** `length(R)` **do** `// Second & third phase (missing nodes)`
  if *i-Node has more than 1 neighbor* **then**
    i-Node is added to the aggregate most strongly connected;
    if there is a tie, i-Node is added to the aggregate with fewer nodes;
  else
    i-Node is added to the aggregate most weakly connected;
    if there is a tie, i-Node is added to the aggregate with fewer nodes;
Share information with the rest of ranks;

Note that Algorithm 15 is parallel, as desired; each rank can execute and generate its own aggregates, then share the information between ranks.

Finally, it is time to see the new aggregates. Figure 3.10 shows the aggregates obtained by Algorithm 15 with the metric given in Definition 3.5. Using $\varepsilon = 0.5$, we obtain 26 aggregates, the same number as Algorithm 13 but with better properties, which are similar to the aggregates obtained by Algorithm 10 with $\varepsilon = 0.25$.

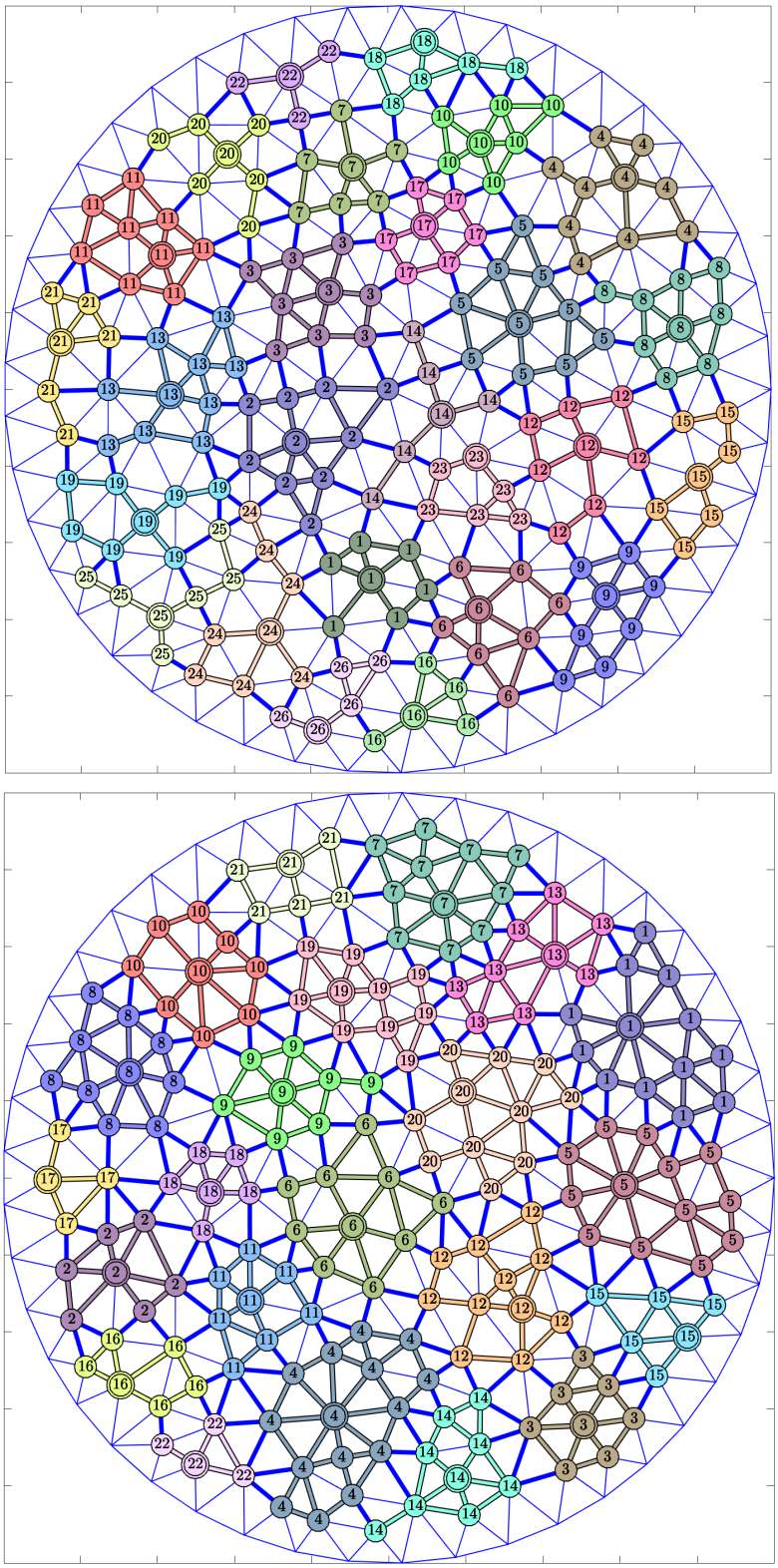


Figure 3.10: Aggregates generated by Algorithm 15 with metric (3.5): (Top) $\varepsilon = 0.5$; (Bottom) $\varepsilon = 0.25$.

3.3.1 The Prolongation Operator

As we said in Section 3.2.1, we will explain why the aggregates generated by Algorithm 13 can lead to a slow convergence rate when the prolongation operator is constructed using Algorithm 14. Let us start discussing the classical rough aggregation prolongation operator, i.e., the entries of $\mathbf{P}$ are only 1 and 0, and so are the entries of $\mathbf{R}$. This choice often yields slow convergence for the multigrid process due to the generation of high-energy modes by such low-order transfer operators. More details and examples about this can be found in references like [20].

Now, consider the smooth aggregation schema, where a smoothing step is applied to $\mathbf{P}$, as shown in Example 3.2. The drawback is that dense prolongation operators $\mathbf{P}$ yield expensive computational operations. Thus, updates to the classical rough prolongation operator were made to make $\mathbf{P}$ as sparse as the classical one, but with a rate of convergence as fast as the smooth one. One possible update to the rough aggregation is the use of $\mathbf{B}_k$, the so-called *near null space* of $\mathbf{A}_k$, as applied in Algorithm 14; this approach works fairly well in `libparanumal`, but it could be better.

To explain how aggregates consisting of only one node yield exactly the same prolongation pattern as in the classical rough aggregation, consider the following example.

Example 3.4. *Suppose we have the nodes 1 to 4 at the level k, the aggregates $\mathcal{C}[1] = \{1\}$, $\mathcal{C}[2] = \{2, 3\}$, and $\mathcal{C}[3] = \{4\}$, and a random vector $\mathbf{B}_k^T = [\ 0.1 \quad 0.4 \quad 0.3 \quad 0.2\ ]$; then the classical prolongation matrix $\mathbf{P}_k$ is given by*

$$\mathbf{P}_k^T = \begin{bmatrix} 1 & & & \\ & 1 & 1 & \\ & & & 1 \end{bmatrix}.$$

The first loop in Algorithm 14 copies the entries of $\mathbf{B}_k$ corresponding to the aggregated nodes. Thus $\mathbf{P}_k$ is updated to

$$\mathbf{P}_k^T = \begin{bmatrix} 0.1 & & & \\ & 0.4 & 0.3 & \\ & & & 0.2 \end{bmatrix}.$$

The second loop constructs the next near null space vector by taking the Euclidean norm of each column of $\mathbf{P}_k$, thus $\mathbf{B}_{k+1}^T = [\ 0.1 \quad 0.5 \quad 0.2\ ]$. The third loop divides the columns of $\mathbf{P}_k$ by $\mathbf{B}_{k+1}$, giving

$$\mathbf{P}_k^T = \begin{bmatrix} 1 & & & \\ & 0.8 & 0.6 & \\ & & & 1 \end{bmatrix}.$$

Thus the columns of $\mathbf{P}_k$ corresponding to aggregates of only one node are as bad as in the classical rough aggregation.

The issue shown in Example 3.4 happens often in `libparanumal`, as we saw in Example 3.3 and Figure 3.4 (bottom). In general, let $Agg(i) = \{j\}$ be the i-th aggregate consisting of only node j. Then the prolongation operator is constructed by Algorithm 14 as follows.

i. $I \leftarrow j$.

ii. $\mathbf{P}_k(I, i) \leftarrow \mathbf{B}_k(I)$, since $I = j$ from step (i). This is equivalent to copying the j-th entry of $\mathbf{B}_k$, as the only nonzero entry of the j-th row of $\mathbf{P}_k$.

iii. $\mathbf{B}_{k+1}(i) \leftarrow \|\mathbf{B}_k(I)\|_2$, when the next near null space vector is updated, its i-th entry is given by $\mathbf{B}_{k+1}(i) = |\mathbf{B}_k(j)|$.

iv. $\mathbf{P}_k(I, i) \leftarrow \mathbf{P}_k(I, i)/\mathbf{B}_{k+1}(i)$, when the column corresponding to the i-th aggregate is normalized; basically, we get $\mathbf{B}_k(j)/|\mathbf{B}_k(j)|$, giving ± 1.

Thus the row corresponding to the i-th aggregate will have ± 1 at the aggregate position and 0 otherwise, i.e., this is the same as the classical rough aggregation schema, producing high-energy modes that drag down the rate of convergence. Thus we can use the aggregates generated by Algorithm 15 and keep using Algorithm 14 to construct the prolongation operator.

In the smooth aggregation scenario, we still have the same idea: the tentative prolongation operator $\widehat{\mathbf{P}}$ is constructed, and then one smooth step is applied to it, as in Example 3.2.

With this new aggregation schema, we expect better performance in the `libparanumal` solver, but before investigating that, we would like to explore the `MG`γ cycle as a solver and preconditioner in the `PCG` solver. In the next chapter we will explore how the different algorithms presented in this chapter perform in the smooth and rough aggregation schema.

Chapter 4

Computational Experiments

At this point we have all the tools needed for implementing a solver for the linear system arising from the Poison equation. In Chapter 2 we explained how we can solve the linear system using multigrid algorithms, or using multigrid as a preconditioner for the conjugate gradient method. Here we will show experimentally which solver is faster, given the algorithmic variants discussed in the last chapter.

4.1 Setup Phase

So far we have discussed how to use the different spaces Ω_i with their respective operators $\mathbf{P}_i$ and $\mathbf{R}_i$, but one of the crucial steps in AMG algorithms is the setup phase, where the operators $\mathbf{P}_i$, $\mathbf{R}_i$, and the restricted representation $\mathbf{A}_i$ are built at each level in the multigrid hierarchy, as outlined in Algorithm 17. Here we built each level using Algorithm 16.

Algorithm 16: AMG level (`AMGlevel`)

input : $\mathbf{A}$, ε, $\mathbf{B}$ (optional)
output: **AMG** (data structure)

$\mathbf{C} \leftarrow$ `StrongGraph`$(\mathbf{A}, \varepsilon)$; `// Strong connectivity matrix`
$FineToCoarse \leftarrow$ `Coarsener`$(\mathbf{A}, \varepsilon)$; `// Get the aggregates`
$[\mathbf{P}, \mathbf{B}] \leftarrow$ `Prolong`$(FineToCoarse, \mathbf{B})$; `// Form the prolongation operator`
$\mathbf{AMG}.P \leftarrow \mathbf{P}$;
$\mathbf{AMG}.B \leftarrow \mathbf{B}$;
$\mathbf{AMG}.R \leftarrow \mathbf{P}^T$; `// Galerkin property`
$\mathbf{AMG}.A \leftarrow \mathbf{RAP}$;
$\mathbf{AMG}.n \leftarrow$ `size`$(\mathbf{A})$; `// Number of DOF`
$\mathbf{AMG}.\omega \leftarrow$ `OptimalOmega`(A) ; `// Optimal relaxation parameter`

Algorithm 16 allows an optional input **B**, the near-null space of **A**. This input is optional because it is only required for the rough aggregation coarsener, but not for the smooth aggregation coarsener. The general routines `StongGraph, FormAggregates`, and `Prolong` are different, depending on the aggregation approach used.

Algorithm 17: AMG setup (`AMGsetup`)

input : **A**, ε, **B** (optional), TS (target size)
output: **AMGlvl**

```
lvl(1).P ← [ ] ;                                          // No P on level 1
lvl(1).R ← [ ] ;                                          // No R on level 1
lvl(1).A ← A ;
lvl(1).B ← B ;
lvl(1).n ← size(A) ;                                      // Number of DOF
lvl(1).ω ← OptimalOmega(A) ;                  // Optimal relaxation parameter
Reduce ← 0 ;
α ← ε ;
AS ← AMG.n ;                                              // Actual size
i ← 1 ;                              // Actual level on the multigrid hierachy
while AS > TS do
    if Reduce = 0 then
        α ← ε ;                                  // Reset α to default value
    lvlt ←AMGlevel(lvl(i).A, α, lvl(i).B) ;          // Build a temporal level
    if AS < 2 * lvlt.n then
        α ← ε/2 ;                                 // Relax the threshold α
        Reduce ← 1 ;
    else
        Reduce ← 0 ;
        i ← i + 1 ;
        lvl(i) ← lvlt ;                          // Save the temporal level
        AS ← lvl(i).n ;                           // Update the actual size
AMGlvl.n ← i ;                                    // Total number of levels
AMGlvl.lvl ← lvl ;                                   // Multigrid hierarchy
```

Algorithm 17 assembles **AMGlvl**.n levels in the multigrid hierarchy, and uses a bootstrap step when there is insufficient reduction in the number of DOF. (This usually happens when there are not enough strong connections, so we need to relax the coarsener threshold.) The number of levels in this multigrid hierarchy depends on TS (target size); let us say, we will keep coarsening the problem until the last level has fewer than 1000 DOF (since this is small enough to solve directly). Also note that Algorithm 17 ensures that each level in the multigrid hierarchy has at least half the number of DOF as the previous level.

4.2 Multigrid as a solver

From all the different algorithms discussed in Chapter 3, we will focus our attention on the combinations shown in Table 4.1.

Table 4.1: Setups for testing the multigrid solver.

Name	Metric	Coarsener	
libparanumal	(3.1)	Algorithm 12	Algorithm 13
sym+libparanumal	(3.5)	Algorithm 12	Algorithm 13
classical+MIS2/LPSCN	(3.2)	Algorithm 12	Algorithm 15
sym+MIS2/LPSCN	(3.5)	Algorithm 12	Algorithm 15

Since the multigrid hierarchy depends on the size of the problem (number of DOF), for small meshes we should not expect more than two levels (1 and 2); on the other hand, for large meshes we expect more levels, to make the coarsest level small enough to solve directly. In the following experiments the damped Jacobi iteration is the smoother with optimal relaxation parameter approximated using 3 steps of the Arnoldi iteration from [19].

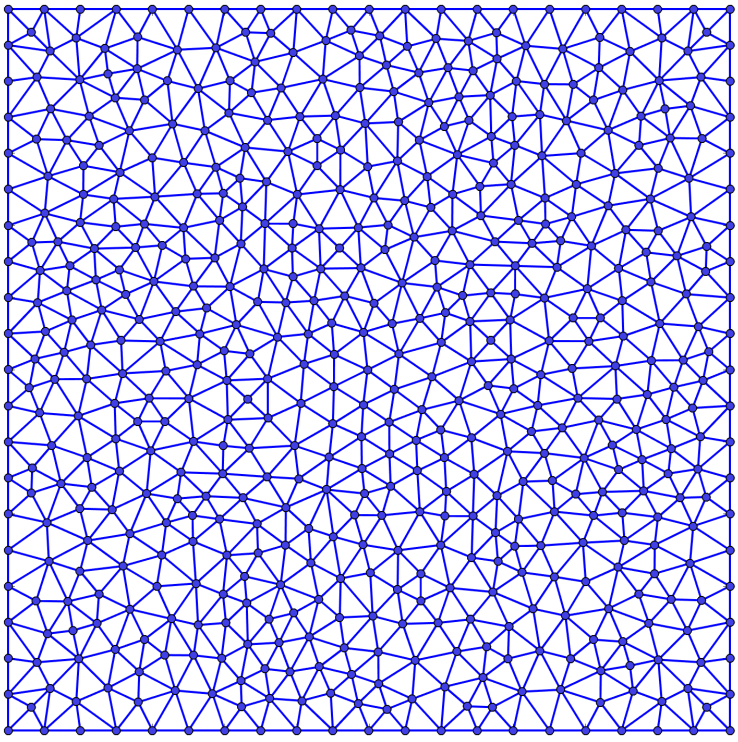


Figure 4.1: Mesh `cavityH01.msh`

Figure 4.1 shows our initial mesh for testing the multigrid solver by iterating Algorithm 7 (`MG`γ). The mesh `cavityH01` consists of 1046 elements, so using linear finite elements gives 564 DOF. Since this is a small problem, we set $TS = 100$ (target size) in the AMG setup phase; thus the multigrid hierarchy will only consist of two levels (levels 1 and 2) for all the different setups in Table 4.1. Since the multigrid hierarchy consists only of the fine and coarse level, the `MG`γ cycle with $\gamma = 1$ is equivalent to the `Two-Grid` cycle.

Since we use homogeneous boundary conditions, we do not count the DOF on the boundary since their values are zero. Table 4.2 shows the multigrid hierarchies obtained by the four different setups.

Table 4.2: DOF in the multigrid hierarchy for the mesh cavityH01.

Setup	DOF	
	lvl 1	lvl 2
libparanumal	484	62
sym+libparanumal	484	60
classical+MIS2/LPSCN	484	64
sym+MIS2/LPSCN	484	60

Figure 4.2 shows that the third setup (classical+MIS2/LPSCN) is fastest in terms of iterations, followed by the fourth setup (sym+MIS2/LPSCN)[1]. In general, we can see that the number of iterations reduces according to the number of smoothing steps applied. A higher number of smoothing steps seems to be effective at reducing the number of *iterations*, but what happens to the elapsed time? We can also see in Figure 4.2 (bottom), that after 3 smoothing steps there is not enough speedup (as from 1 to 3 smoothing steps); moreover as the number of smoothing steps increases, the elapsed times seem to increase as well. Since the linear system arising from the mesh `cavityH01` is a small problem, we would like more evidence about this behavior for larger problems.

Now consider the mesh `cavityH005` shown in Figure 4.3, which consists of 4184 elements; linear finite elements give 2173 DOF. (Recall that, since we are using homogeneous boundary conditions, the DOF located on the boundary are removed.) Thus this is also a relatively small problem, so as before we keep $TS = 100$; each multigrid hierarchy will then have 3 levels, but since we are using different coarseners we expect these levels to have slightly different sizes, as shown in Table 4.3.

[1] In Figure 4.2 and all subsequent plots, Elapsed Time is shown in seconds.

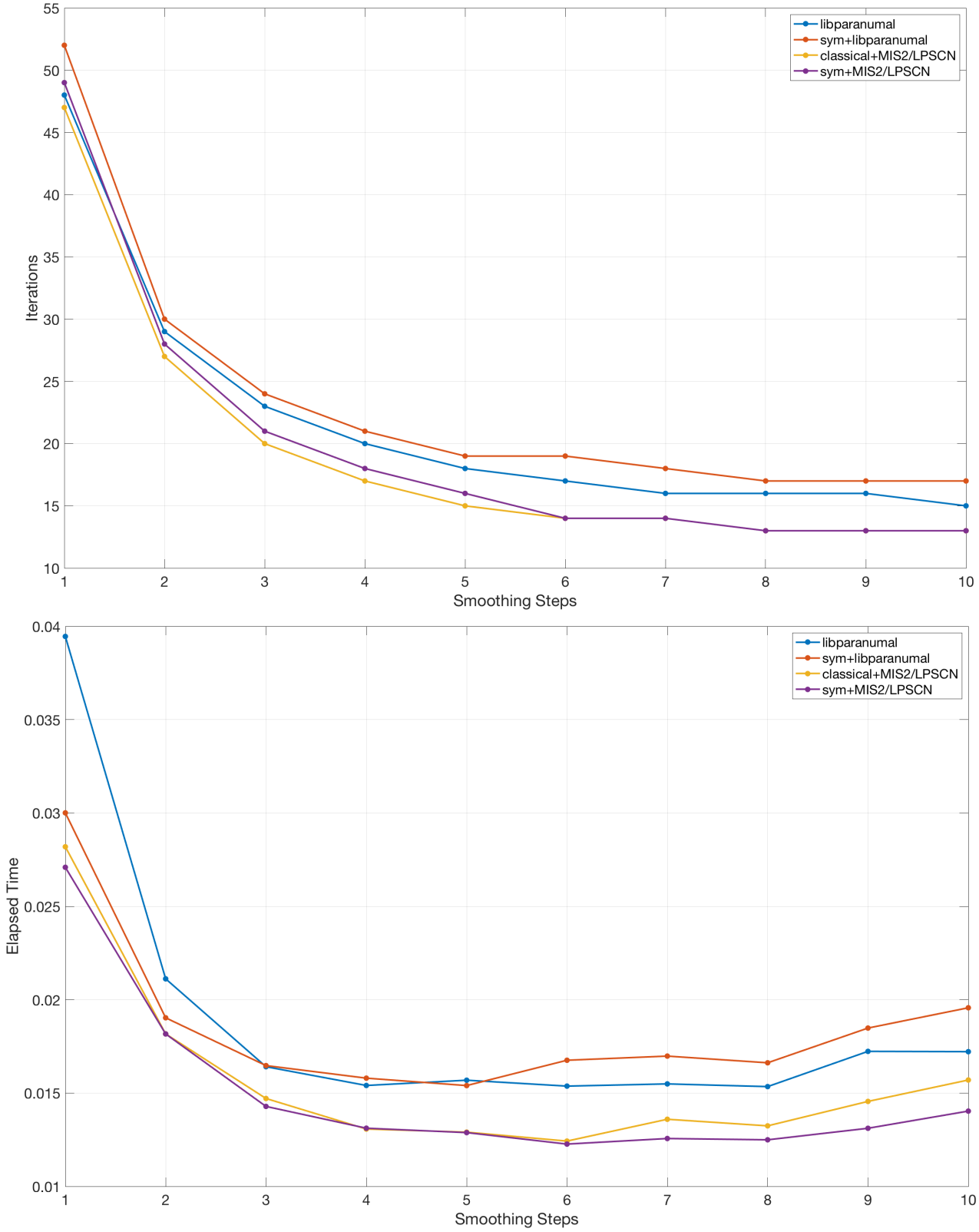


Figure 4.2: Experiment 01: (Top) number of iterations; (Bottom) elapsed time using the `Two-Grid` cycle as solver.

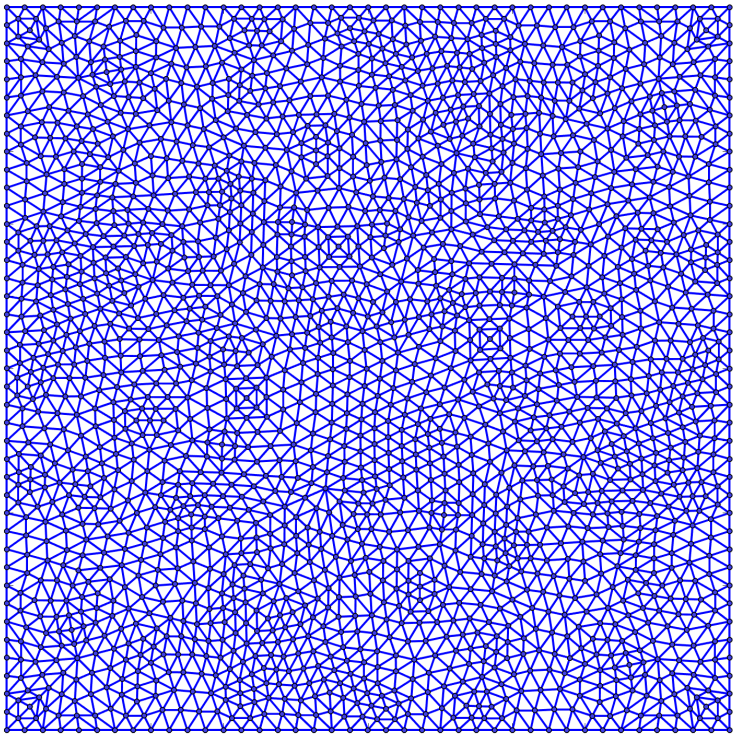


Figure 4.3: Mesh `cavityH005.msh`

In Table 4.3 we can see that the second and fourth setups seem to be more aggressive in the first coarsening, but in the second coarsening the third setup is the most aggressive coarsener. Note that being too aggressive could lead to a slower convergence rate, since too much information is lost in the inter-grid operations.

Table 4.3: DOF in the multigrid hierarchy for the mesh cavityH005.

Setup	DOF		
	lvl 1	lvl 2	lvl 3
libparanumal	2013	270	29
sym+libparanumal	2013	260	35
classical+MIS2/LPSCN	2013	268	22
sym+MIS2/LPSCN	2013	260	24

Figure 4.4 (top) shows that the the third setup converges slower than the other setups, while the fourth setup seems to be working better than the other setups in terms of number of iterations and elapsed time, as shown in Figure 4.4 (bottom).

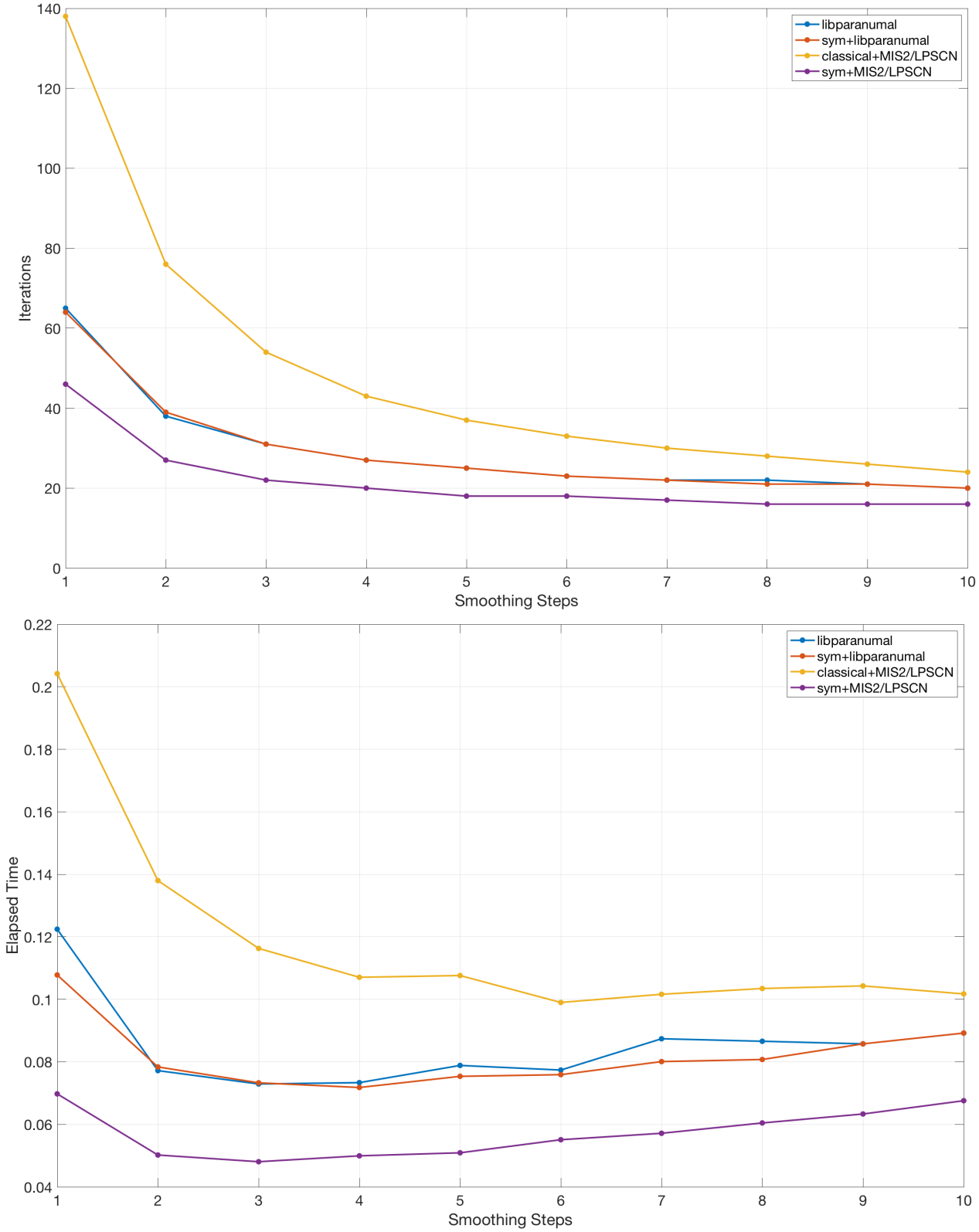


Figure 4.4: Experiment 02: (Top) number of iterations; (Bottom) elapsed time using the MGγ cycle as solver.

Let us keep increasing the size of the problem: now we consider the mesh `cavityH0025` shown in Figure 4.5, which consists of 16736 elements. Using linear finite elements, we get 8529 DOF; after applying the boundary conditions, this discretization gives the multigrid hierarchy shown in Table 4.4.

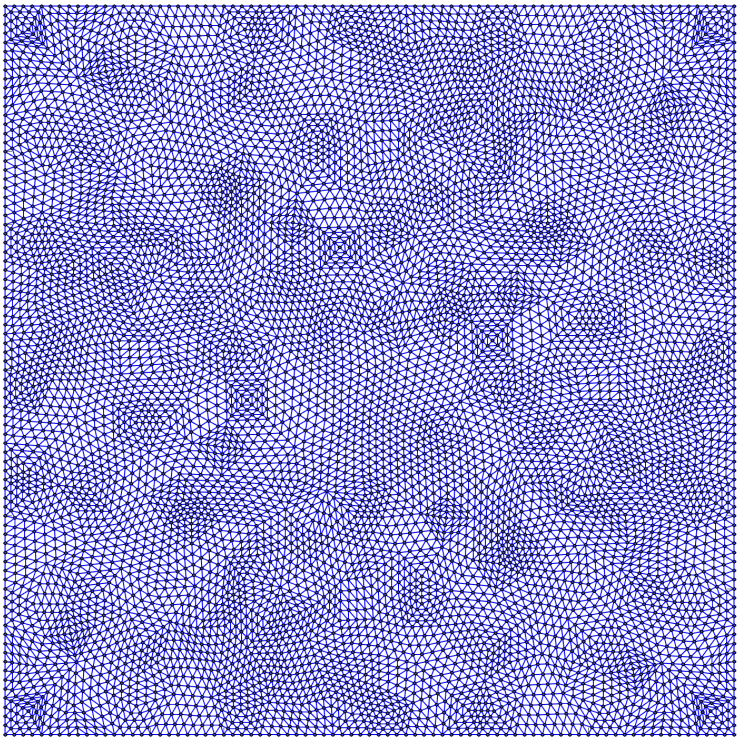


Figure 4.5: Mesh cavityH0025.msh

In Table 4.4 we can see that the third setup is the most aggressive coarsener, decreasing quickly the number of DOF, while the fourth setup assembles multigrid levels with similar numbers of DOF as the first setup, but the aggregates have better features (as discussed in Chapter 3), which should yield a better convergence rate.

Table 4.4: DOF in the multigrid hierarchy for the mesh cavityH0025.

Setup	DOF			
	lvl 1	lvl 2	lvl 3	lvl 4
libparanumal	8209	1154	116	21
sym+libparanumal	8209	1153	131	30
classical+MIS2/LPSCN	8209	1118	78	—
sym+MIS2/LPSCN	8209	1153	118	16

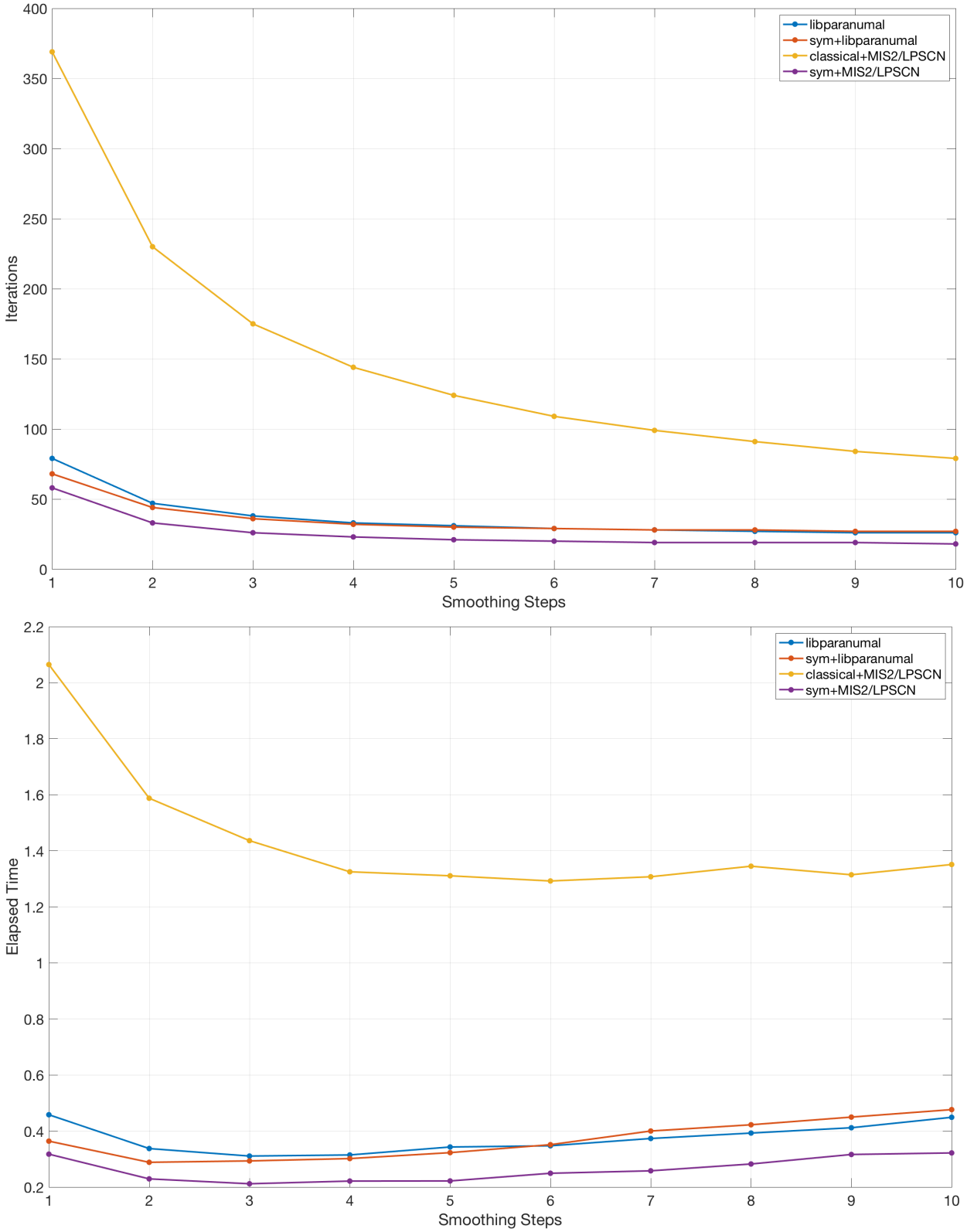


Figure 4.6: Experiment 03a: (Top) number of iterations; (Bottom) elapsed time using the MGγ cycle as solver.

Figure 4.6 (top) shows that the most aggressive coarsener (third setup) requires the most iterations; as we said before, being too aggressive will yield a poor rate of convergence, because a lot of information is lost in the inter-grid operations. On the other hand, the fourth setup is still performing better in terms of number of iterations as well as elapsed time, as shown in Figure 4.6 (bottom).

Note that in Table 4.4, there is not much gain in constructing level 3 in the multigrid hierarchy, since the number of DOF is small enough in level 2. How different are the results if we only use three levels rather than four? Note that changing the target size to $TS = 1000$ (for example) will cut the multigrid hierarchy from Table 4.4 at level 3, i.e., the `MG`γ cycle will be a `V`-cycle through level 1 to level 3. As we can see in Figure 4.7, using $TS = 1000$ more or less produces the same results. This is expected, since level 3 is small enough to solve the coarsened problem directly.

In general, since we use one level less, the computational cost should be less than using four levels; indeed this reduces the elapsed time, as shown in Figure 4.7 (bottom); however, the price is paid in the number of iterations, which increases a little bit, as shown in Figure 4.7 (top). We observe this behavior in the first, second, and fourth setup, while the third setup seems to get worse in iterations and elapsed time. We will discuss this behavior in more detail at the end of the section.

Selecting a convenient target size, TS, in the AMG setup is important. For large problems $TS = 1000$ seems to be a good selection; it is the default size in `libparanumal`, so we will use it also. Moreover, from Tables 4.2, 4.3, and 4.4, we note that our setups are aggressive, since the number of DOF are reduced by 80% to 90% from level to level. Take, for example, the first setup reduction shown in Table 4.5.

Table 4.5: Reduction in the number of DOF in the multigrid hierarchy by the first setup.

Mesh	DOF		
	lvl 1 to lvl 2	lvl 2 to lvl 3	lvl 3 to lvl 4
cavityH01	87%	—	—
cavityH005	86%	90%	—
cavityH0025	86%	90%	82%

Since our setups aggressively reduce the number of DOF, a slow rate of convergence is likely, when the setup is not adequate. As we can see in Figures 4.6 and 4.7, the third setup converges slowly as the size of the size of the problem increases. Thus, to avoid having the solver run for long time, we will use $N_{max} = 1000$ (maximum number of iterations) to end the `MG`γ cycle iteration.

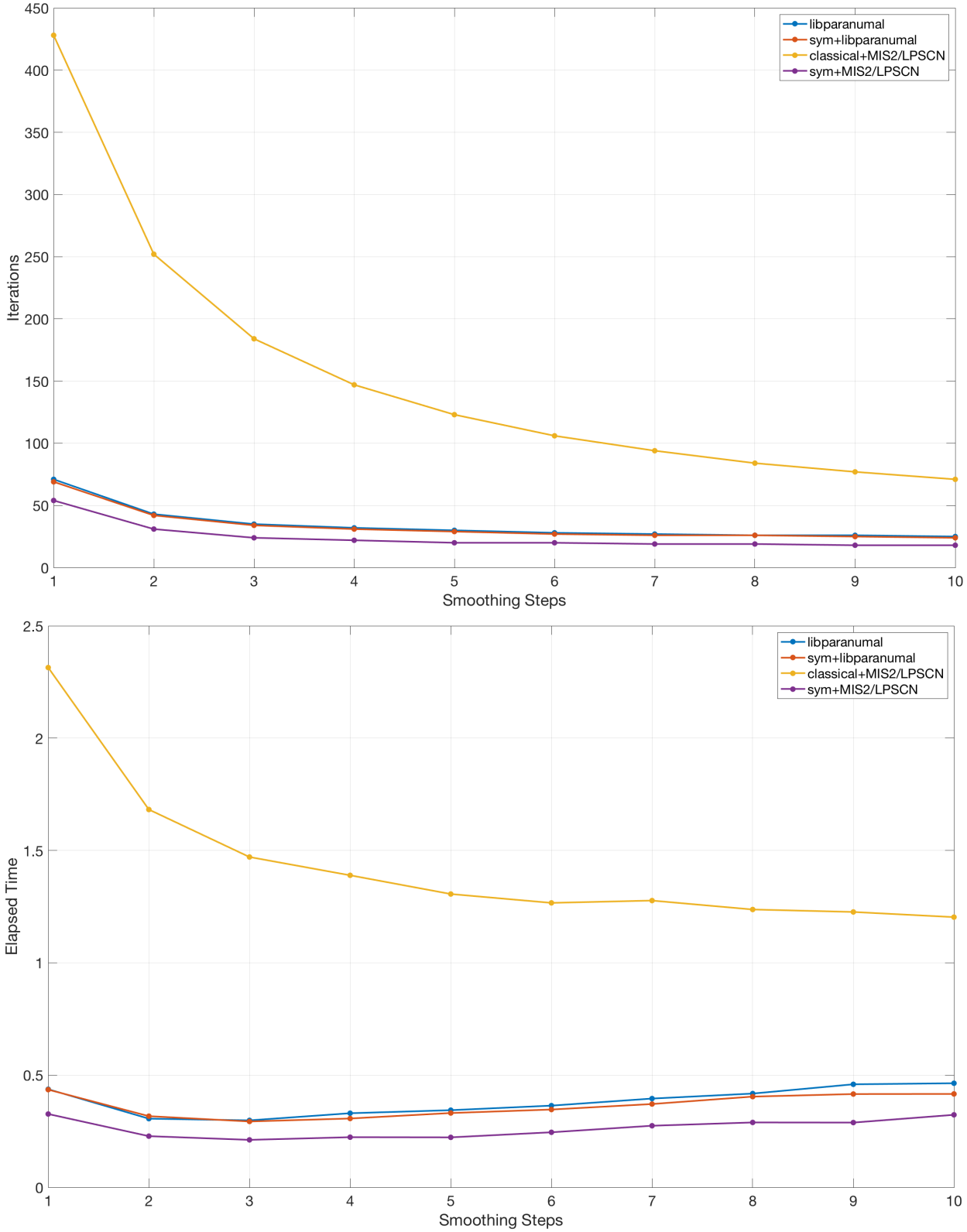


Figure 4.7: Experiment 03b: (Top) number of iterations; (Bottom) elapsed time using the MGγ cycle as solver.

Consider for example, the mesh `cavityH00125` shown in Figure 4.8, which consists of 66944 elements. Using linear finite elements, we get 33793 DOF. After applying the boundary conditions, this discretization gives the multigrid hierarchy shown in Table 4.6.

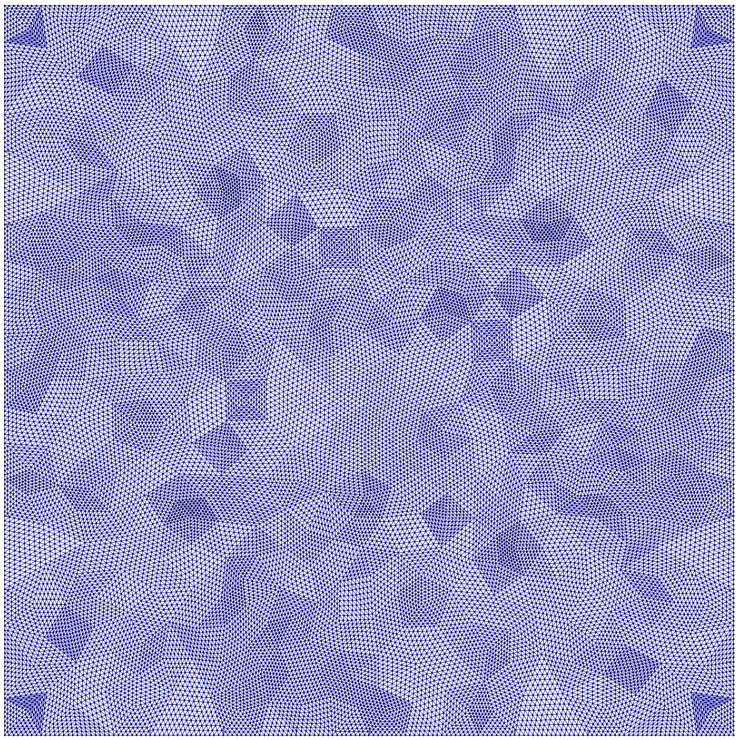


Figure 4.8: Mesh cavityH00125.msh

Tabla 4.6 shows, one more time, that the third setup is the most aggressive coarsener; in contrast, the second setup is the least aggressive.

Table 4.6: DOF in the multigrid hierarchy for the mesh cavityH00125.

Setup	DOF		
	lvl 1	lvl 2	lvl 3
libparanumal	33153	4856	465
sym+libparanumal	33153	4803	527
classical+MIS2/LPSCN	33153	4475	321
sym+MIS2/LPSCN	33153	4803	470

Note that the third setup is too aggressive, yielding slow convergence for large problems. In this experiment, the number of iterations for the third setup gets close to N_{max}. Figure 4.9 shows clearly that the fourth setup performs better, and using 3 smoothing steps usually yields the fastest convergence rate.

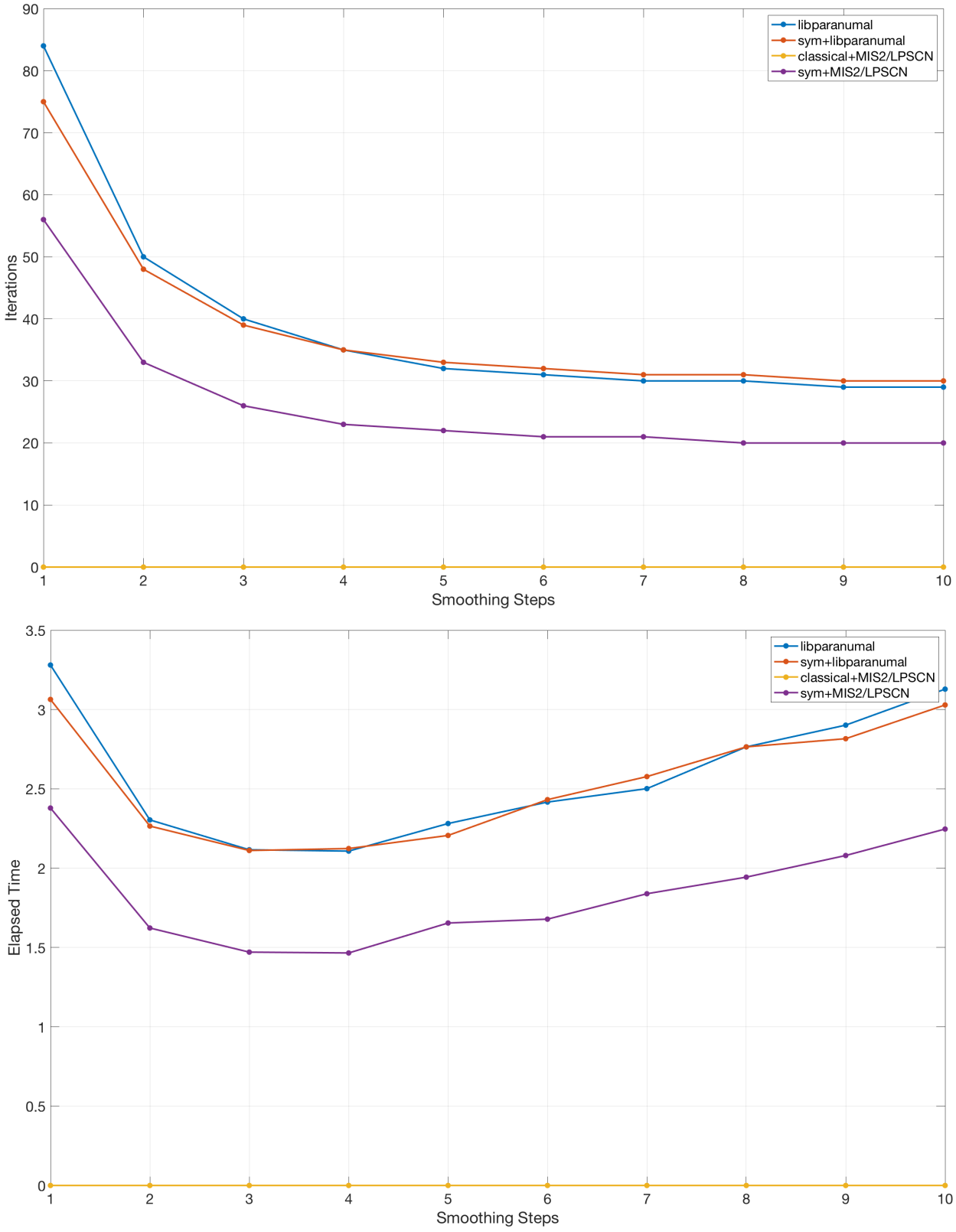


Figure 4.9: Experiment 04: (Top) number of iterations; (Bottom) elapsed time using the MGγ cycle as solver.

Let us go back to Experiment 03 using mesh `cavityH0025`. We stated that many levels are not always better in a multigrid hierarchy. To illustrate this point, we will use the `MG`γ cycle, but rather than go through all the levels, we will iterate with a fixed number of levels each time. Since the fourth setup performs better than the other setups, we will use this setup to show how a fixed number of cycles affects the performance of the `MG`γ cycle with $\gamma = 1$. In general, the same qualitative behavior is observed with the other three setups.

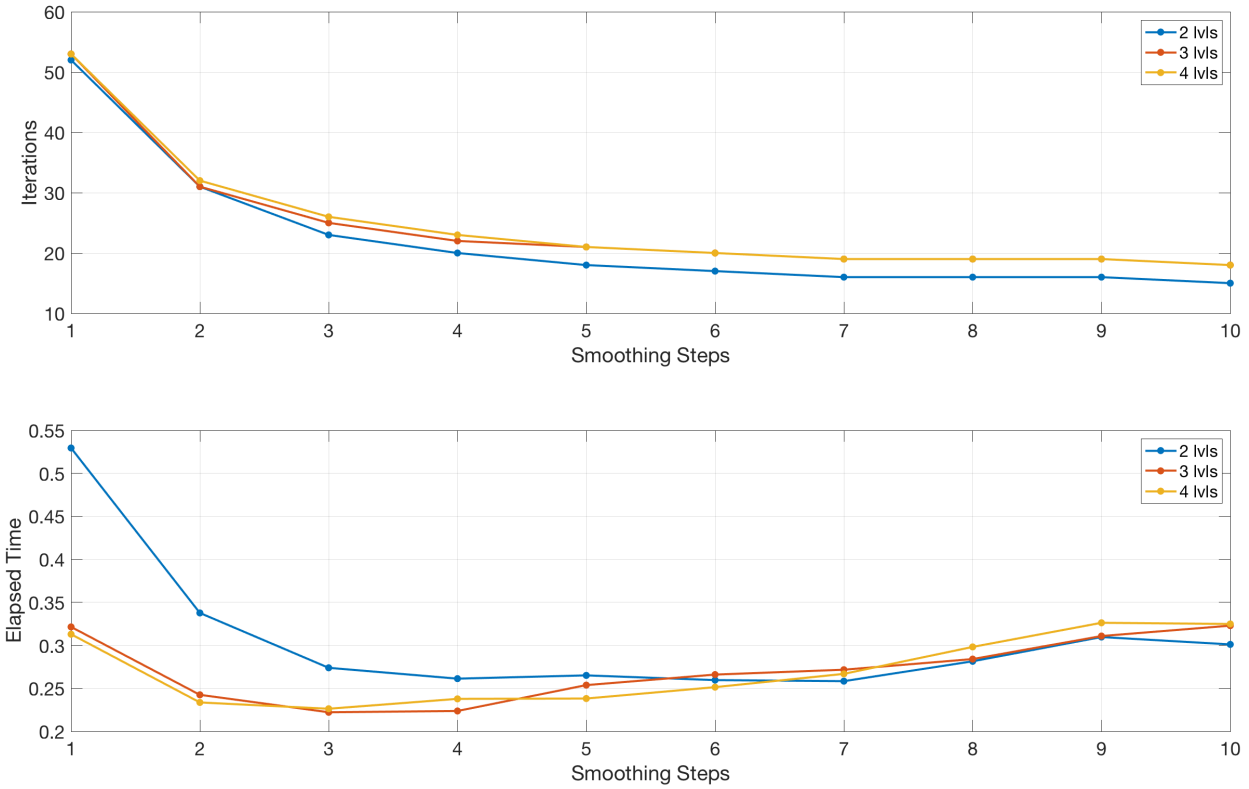


Figure 4.10: Experiment 03c: (Top) number of iterations; (Bottom) elapsed time using a fixed number of levels in the `MG`γ cycle as solver.

Figure 4.10 (top) shows that, in terms of the number of iterations, using the `Two-Grid` cycle (2 levels) is faster than the `V`-cycle (3 and 4 levels); moreover, when we use more levels, the number of iterations increases (2 to 3 smoothing steps).

On the contrary, Figure 4.10 (bottom) shows that the `V`-cycle (3 levels) is almost as fast as the `V`-cycle (4 levels), while the `Two-grid` is slower, when we use from 1 to 5 smoothing steps. In contrast, one can expect that 4 levels should perform better than 3 levels, but, as we can see in Table 4.4, from level 3 (118 DOF) to level 4 (16 DOF) there is not enough gain in the coarsest level, in the sense that at level 3 the problem is small enough to be solved directly; so rather than speed up the solver, we could make it slower when we add one more

level.

To illustrate the last statement, consider the mesh `cavityH00125`, with $TS = 100$, rather than $TS = 1000$; thus, the multigrid hierarchy will have four levels (33153/4803/470/48).

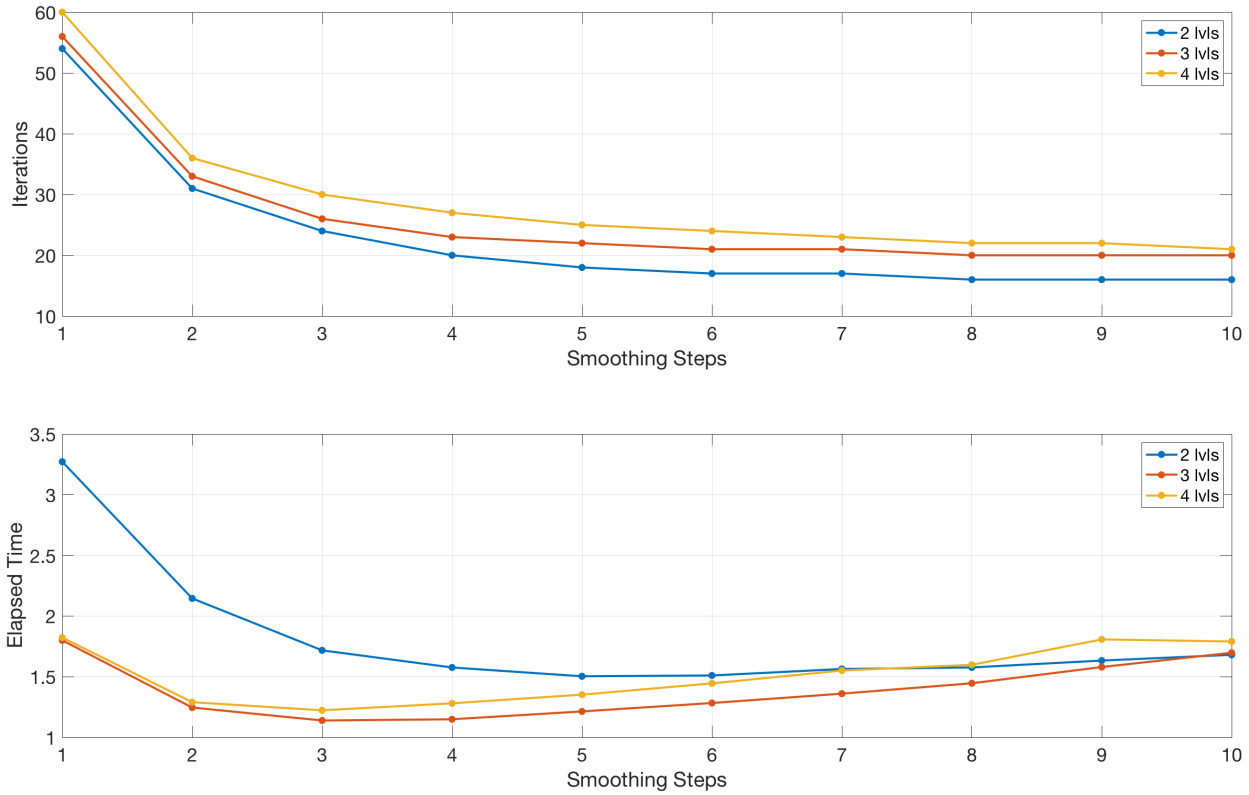


Figure 4.11: Experiment 04b: (Top) number of iterations; (Bottom) elapsed time using a fixed number of levels in the `MG`γ cycle as solver.

Figure 4.11 (top) shows clearly that using more levels in the `MG`γ cycle with $\gamma = 1$ increases the number of iterations, since it involves more inter-grid operations, but can benefit the elapsed time, as shown in Figure 4.11 (bottom). Note also that the solver using 3 levels was faster than the one using 4 levels; this happens because between level 3 (470 DOF) and level 4 (48 DOF) there is not much gain, in the sense that level 3 is small enough to solve the problem exactly.

Thus, more levels in the multigrid hierarchy are not always better; but usually more levels, with enough reduction in the number of DOF, will lead to faster convergence even though the number of iterations could increase.

4.3 Multigrid as preconditioner

In the previous experiments iterating the MGγ cycle, we used $\gamma = 1$, which corresponds to the V-cycle; using $\gamma > 1$ usually yields faster convergence in terms of the number of iterations, but the elapsed time is usually slower.

Now, rather than use the MGγ cycle as a solver, let us use it as a preconditioner for the PCG method (Algorithm 3). The following experiments will compare the V-cycle and W-cycle as solvers and preconditioners. Let us start with the mesh `cavityH01` (Figure 4.1).

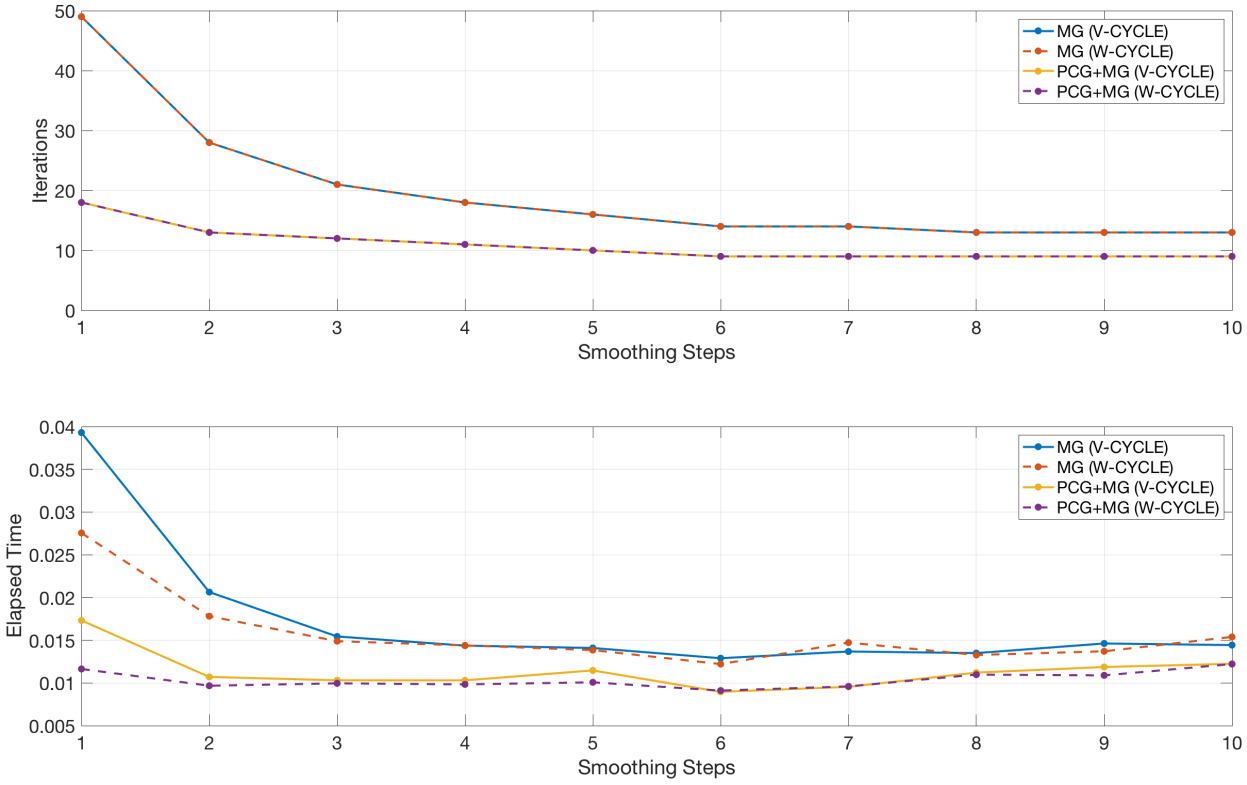


Figure 4.12: Experiment 05: (Top) number of iterations; (Bottom) elapsed time using the MGγ cycle as preconditioner in the PCG solver.

Figure 4.12 shows that the MGγ cycle with $\gamma = 1$ and $\gamma = 2$ converges in the same number of iterations, because the multigrid hierarchy consists only of two levels; since we are using $TS = 100$, the V-cycle and W-cycle produce almost the same output, with a slight difference in the elapsed time. This behavior is observed in small problems, but we do not expect the same for larger problems.

As we observed previously 3 smoothing steps seem to be optimal, but in this experiment there is no clear choice between $\gamma = 1$ and $\gamma = 2$.

Now let us use the mesh `cavityH005` (Figure 4.3). Here we keep $TS = 100$, so again the multigrid hierarchy has 3 levels (Table 4.3); we expect different results than in the previous experiment.

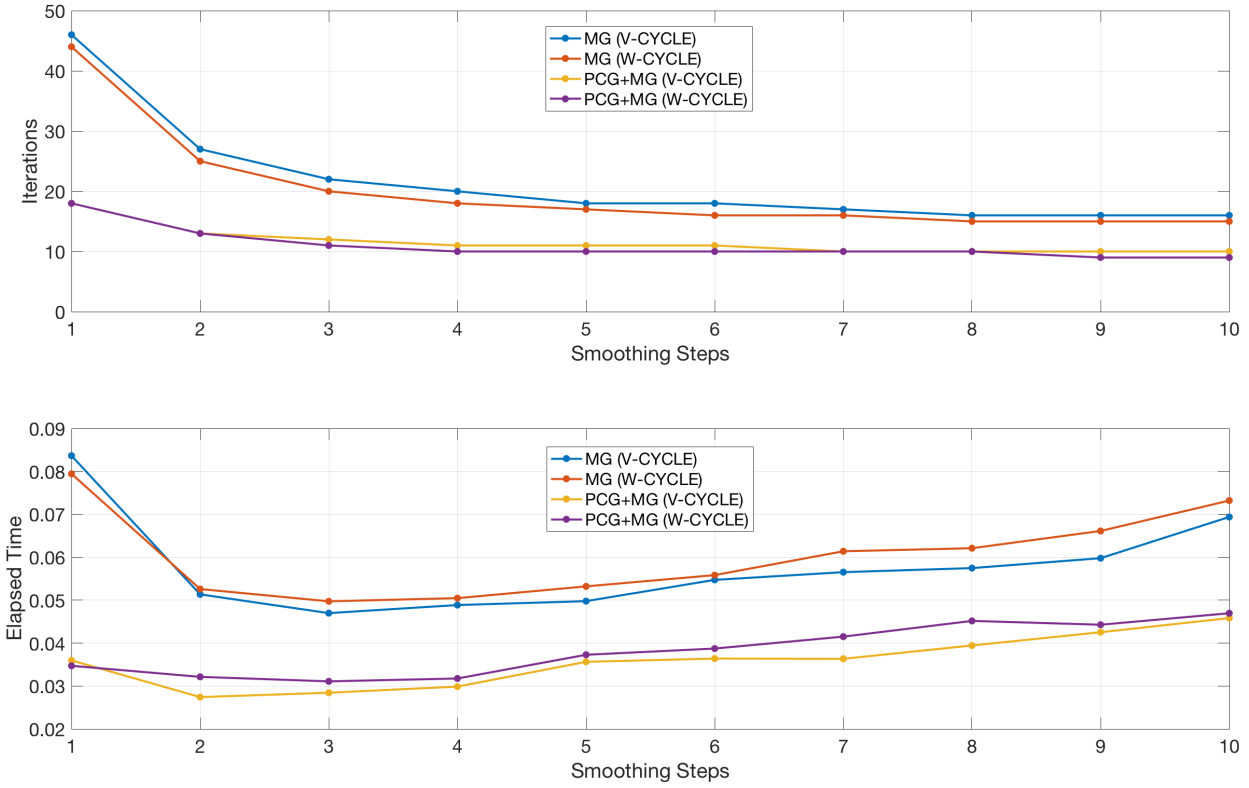


Figure 4.13: Experiment 06: (Top) number of iterations; (Bottom) elapsed time using the `MG`γ cycle as a preconditioner in the `PCG` solver.

Figure 4.13 shows that the `W`-cycle is faster than the `V`-cycle in terms of number of iterations, but again when we use the multigrid cycles as preconditioners in PCG, we get almost the same number of iterations; with respect to elapsed time, the `V`-cycle seems to perform slightly better.

Since the `V`-cycle requires fewer inter-level transfer operations, it is natural to expect it to require less elapsed time than the `W`-cycle as a solver and preconditioner. Based on the experiments, if we have to choose the optimal preconditioner and number of smoothing steps, we recommend the `V`-cycle or the `MG`γ cycle with $\gamma = 1$ and 3 smoothing steps.

Note that more smoothing steps indeed lead to a faster convergence rate, in terms of number of iterations, but the price is paid in the elapsed time. Since our goal is to minimize the elapsed time, not the iteration count, we should use between 2 and 4 smoothing steps.

Does the same hold for large problems? Let us consider the mesh `cavityH0025` (Figure 4.5). Recall, that we here use $TS = 1000$, so the multigrid has 3 levels again (Table 4.4).

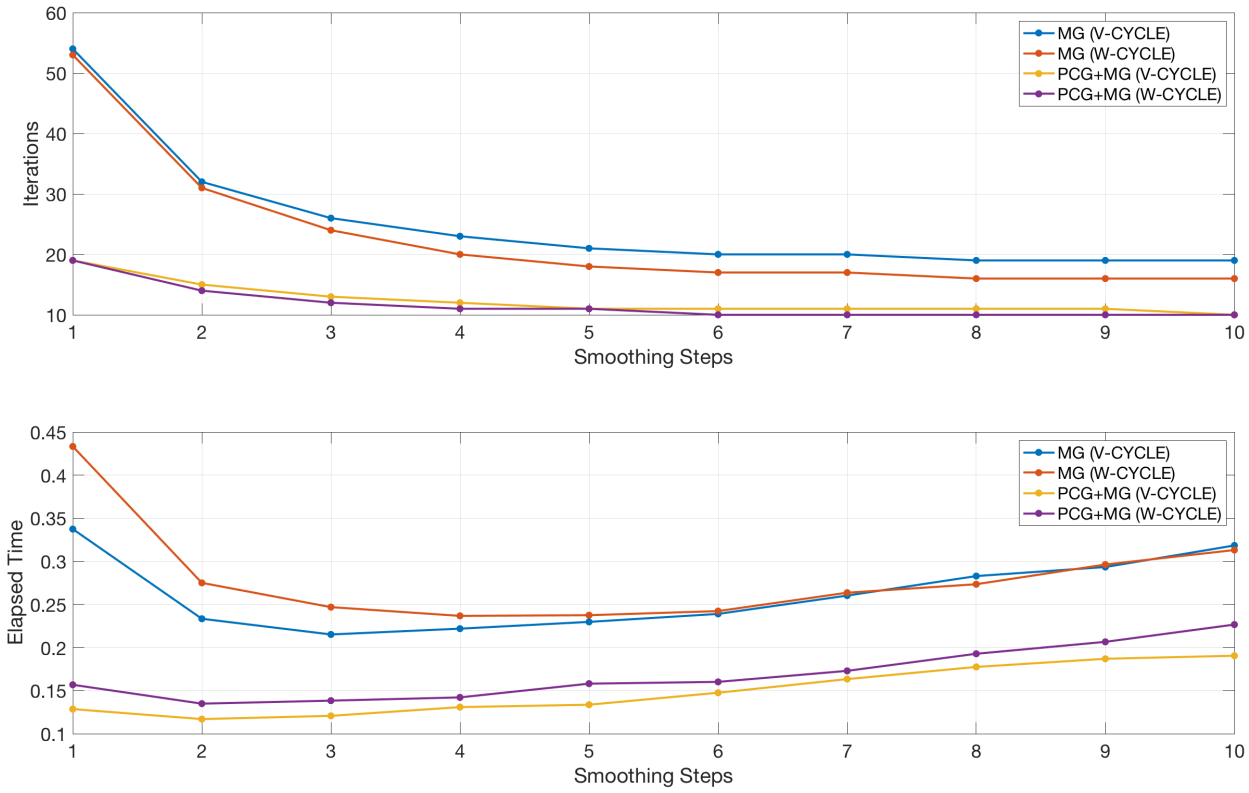


Figure 4.14: Experiment 07: (Top) number of iterations; (Bottom) elapsed time using the `MG`γ cycle as preconditioner in the `PCG` solver.

Figure 4.14 shows that for a small number of smoothing steps (from 1 to 4), the `V`-cycle is faster than the `W`-cycle in terms of elapsed time, but slower in iteration count. After 4 smoothing steps, both cycles require almost the same time, but the `W`-cycle requires fewer iterations than the `V`-cycle.

On the other hand, when we use the `MG`γ cycles as a preconditioner in the PCG solver, we get almost the same number of iterations, but in terms of elapsed time the `V`-cycle performs better than the `W`-cycle.

Here the clear choice is to use the V-cycle with between 2 to 4 smoothing steps, which performs better in terms of elapsed time and converges in a small number of iterations.

Finally, let us use the mesh `cavityH00125` (Figure 4.8). Recall that we here keep using $TS = 1000$, so the multigrid hierarchy again has 3 levels (Table 4.6). In this problem, we have 33153 DOF, so we would like to see if the setup that we chose before performs better as well.

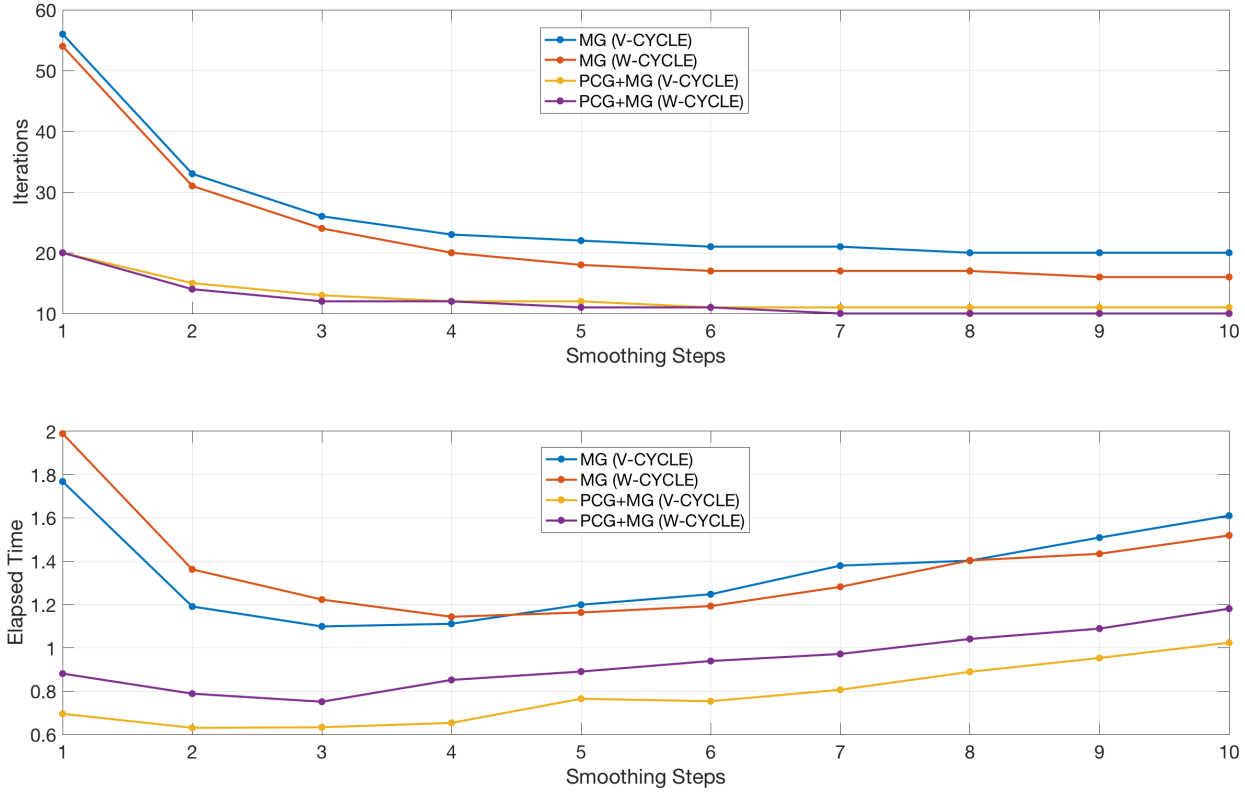


Figure 4.15: Experiment 08: (Top) number of iterations; (Bottom) elapsed time using the MGγ cycle as preconditioner in the PCG solver.

In Figure 4.15 we can see that for standard multigrid, the W-cycle performs better than the V-cycle in iteration count, but in elapsed time the V-cycle is faster, with a small number of smoothing steps, being the fastest with 3 smoothing steps.

When we use the MGγ cycle as a preconditioner in the PCG solver, we observe that the W-cycle is slightly faster than the V-cycle, in terms of number of iterations, but in terms of elapsed time the V-cycle is still faster than the W-cycle. Thus, again the V-cycle with 2 to 4 smoothing steps performs better. Thus we recommend taking 3 smoothing steps by default.

4.4 Parallel Simulations

In the previous sections we analyzed the MGγ cycle as a solver and preconditioner for the PCG method, using smooth aggregation with damped Jacobi as smoother in serial. Now let us look at the rough aggregation accelerated by using the near null spaces (as discussed at Section 3.3) running in parallel.

Since we aim to run the simulation with a high volume of data, let us start by introducing high order finite elements. Consider the mesh `cavityLshape.msh` shown in Figure 4.16; this is a small mesh consisting of 32 triangular elements and 25 nodes.

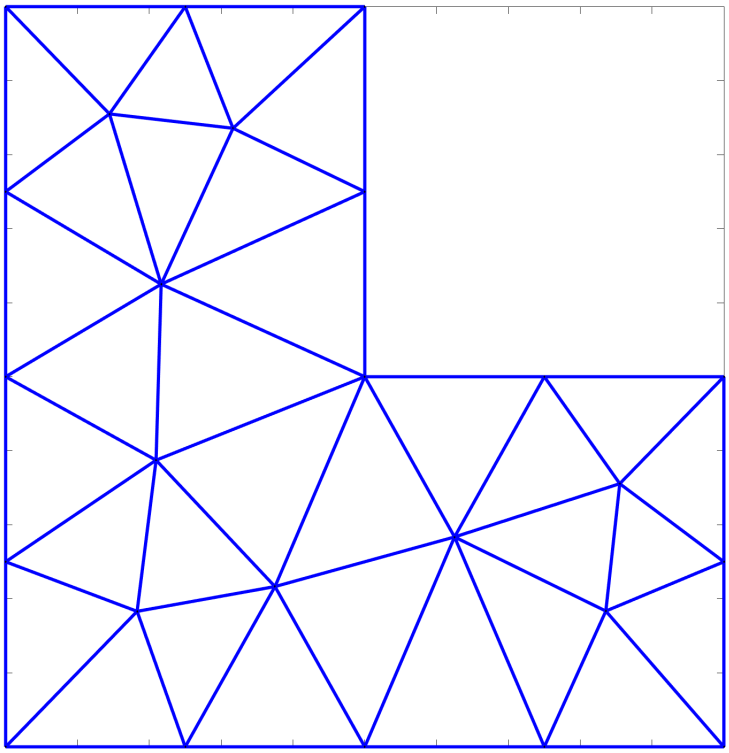


Figure 4.16: Mesh `cavityLshape.msh`

In the previous sections we increase the size of the problem by refining the mesh; now we will do it by increasing the order of the finite elements. Let us start by considering fifth order finite elements in the mesh `cavityLshape`. To do this we use "warp & blend" nodes as in [12] and [22]. Note that we will need 21 nodes per element.

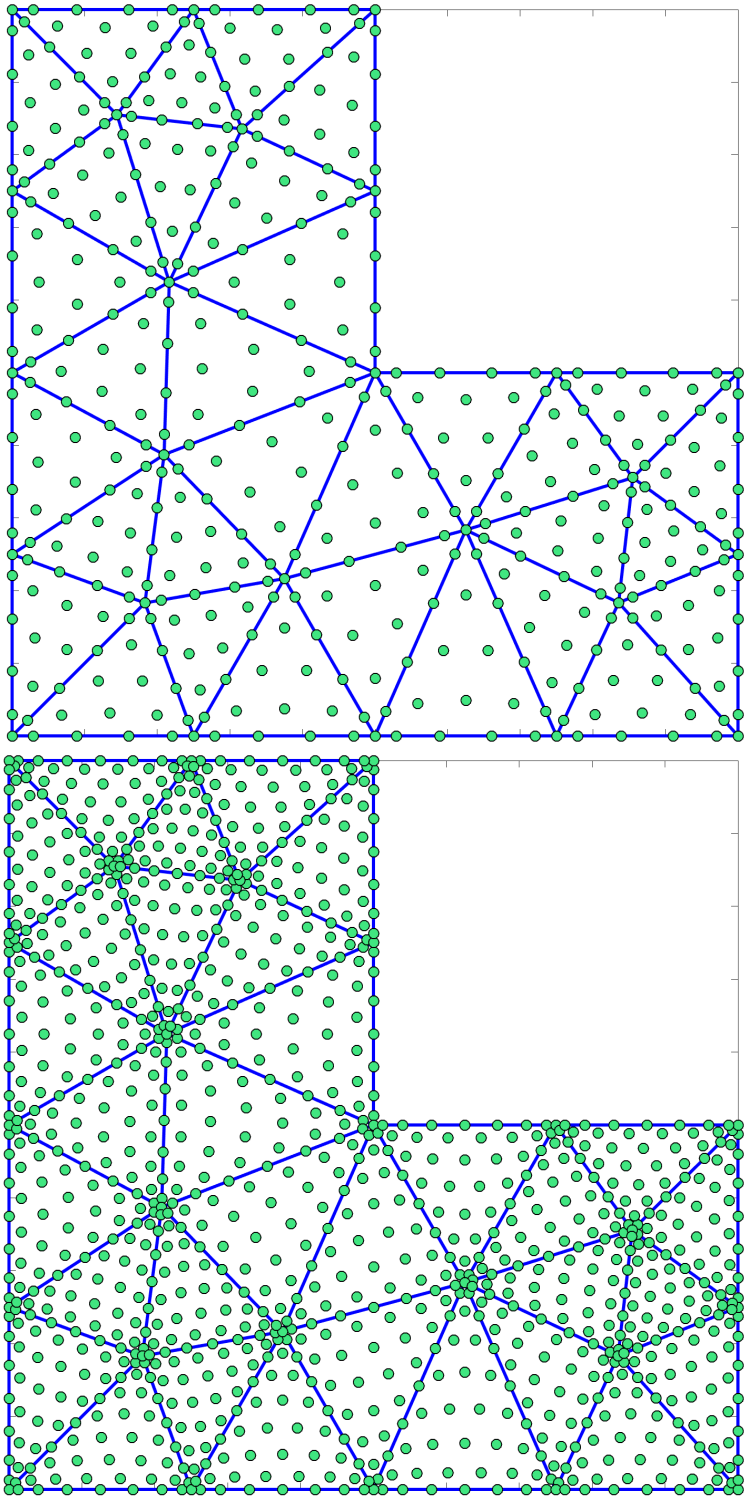


Figure 4.17: High order finite elements: (Top) fifth order; (Bottom) eighth order in the mesh `cavityLshape.msh`

Figure 4.17 (top) shows the extra nodes computed over the physical mesh `cavityLshape`; here we have the same 32 elements, but now 444 physical nodes, since each element has 21 nodes. Note that we have fewer nodes than $32 \times 21 = 672$, since the nodes along the edges of the triangular elements are the same for adjacent elements; but when we work in parallel, the mesh is split by elements between ranks, and each rank has its own copy of nodes (duplicating nodes in the edges), so we end up manipulating 672 nodes. Similarly, if we use eighth order finite elements in the mesh `cavityLshape`, we need 45 nodes per element, yielding 1092 physical nodes, as shown in Figure 4.17 (bottom).

Now, rather than using high order finite elements on a coarse mesh, let us use them on a fine mesh like `cavityH0025`. We compare results between the default algorithms in `libparanumal` and our proposed algorithms in Section 3.3.

When we use fifth order finite elements, we end up with 351456 DOF, and the results are shown in Table 4.7. Similarly, we get 753120 DOF for the eighth order finite elements, and the results are shown in Table 4.8.

Table 4.7: Comparison between `libparanumal` and `sym+MIS2/LPSCN` in `cavityH0025` with fifth order finite elements.

Ranks	libparanumal		sym+MIS2/LPSCN	
	Iterations	Elapsed time(s)	Iterations	Elapsed time(s)
1	109	0.4952	90	0.3168
2	166	0.7109	129	0.5334
3	166	1.7291	129	1.1163
4	169	1.0403	127	0.6607

Table 4.8: Comparison between `libparanumal` and `sym+MIS2/LPSCN` in `cavityH0025` with eighth order finite elements.

Ranks	libparanumal		sym+MIS2/LPSCN	
	Iterations	Elapsed time(s)	Iterations	Elapsed time(s)
1	238	2.6008	173	1.7298
2	230	3.2561	184	1.8848
3	237	8.1849	178	4.5393
4	239	4.5084	177	2.2033

Our next question is, how far can we go? Do we observe the same results with fine meshes or a large number of DOF?

The next experiments will be done for the mesh `cavityH00125`. Let us start using sixth order finite elements, this discretization gives 1874432 DOF, and the comparison between results are shown in Table 4.9.

Table 4.9: Comparison between `libparanumal` and `sym+MIS2/LPSCN` in `cavityH00125` with sixth order finite elements.

Ranks	libparanumal		sym+MIS2/LPSCN	
	Iterations	Elapsed time(s)	Iterations	Elapsed time(s)
1	208	3.43205	154	2.31248
2	309	4.87735	237	3.47039
3	208	10.41594	156	6.90600
4	215	5.58381	159	3.84596

Now, let us use seventh order finite elements; this discretization gives 2409984 DOF, and the results are shown in Table 4.10.

Table 4.10: Comparison between `libparanumal` and `sym+MIS2/LPSCN` in `cavityH00125` with seventh order finite elements.

Ranks	libparanumal		sym+MIS2/LPSCN	
	Iterations	Elapsed time(s)	Iterations	Elapsed time(s)
1	331	7.97031	167	4.16788
2	364	9.59701	264	6.08656
3	228	18.43983	170	12.53917
4	227	9.39497	171	6.52101

Since similar results are observed in our large-scale problems with high order finite elements, we thus conclude our experiments with satisfaction, because the modifications we proposed in Chapter 3 have yielded a considerable speedup.

A peculiar observation in all the parallel results is that the elapsed time with 1 rank is always less than any other elapsed times. The parallel experiments were done on the *pascal* cluster (VT Math Department), where each rank has its own GPU (GeForce GTX 1080); thus for the data volume used, the GPU solver is good enough to perform well. For extreme-scale problems, one rank easily can run out of memory, being unable to handle the data volume, showing the importance of a parallel solver.

Chapter 5

Conclusions

In this thesis, we have studied some algebraic multigrid (AMG) techniques to solve the Poisson equation. In Chapter 2 we talked about the foundations of AMG, describing the classical multigrid cycles, from the `Two-Grid` cycle to the `MG`γ cycle; recall that in this chapter we assumed the existence of inter-grid operators (prolongation and restriction), giving a generic explanation of AMG. In Chapter 3 we addressed the inter-grid transfer operators; among all the different techniques for constructing these operators, we used aggregation. Here, we also proposed the `sym+MIS2/LPSCN` setup, which combines (in our opinion) the best ideas of the MIS(2) aggregation used in `libparanumal` and the standard smooth aggregation. Finally, in Chapter 4 we tested the `MG`γ cycle as a solver and preconditioner in the `PCG` method. Here, we obtained considerable speedup over the `libparanumal` codes in both serial and parallel tests.

Now, let us summarize some of the main conclusions of this thesis.

- Algebraic Multigrid (AMG) is highly dependent on the effectiveness of the inter-grid transfer operators; thus, it is natural to expect better results when we can define better prolongation/restriction operators.

- For large-scale problems, smooth aggregation is a hard task to perform efficiently in parallel. On the other hand, rough aggregation is easily parallelizable; using the near null space technique, it can be fast enough to solve large-scale problems. Note that this technique is used to define better inter-grid transfer operators, but, as we showed in Section 3.3, depending on the geometry of the aggregates, some aggregation algorithms produce the same patterns as the plain rough aggregation. Thus we need to pay attention to some geometric qualities to improve the aggregation performance.

- The aggregation is usually a hard task in parallel, since the classical method is a greedy sequential algorithm. Thus some graph techniques have been used to mimic the aggregates generated by classical algorithm. Here we used aggregation around the root nodes generated by MIS(2), considering the idea of strong neighbors as in the classical algorithm, letting each rank generate its own aggregates and communicate the information to the other ranks.

- Since we construct the aggregates locally by rank, we are expecting good mesh partitions, and so the local aggregation is good enough to give a fast convergence rate. In particular, `liparanumal` uses the Morton space filling curve to split the mesh between ranks. In general the results are good, but infrequently the algorithm generates irregular partitions, leading to slower convergence compared with different partitions of the same mesh. This behavior can be seen from Tables 4.7 to 4.10.

- A drawback in our approach is that we ignore the strong connections between nodes in different ranks. Thus we rely a lot on the quality of the mesh's splitting between ranks. Also, we are designing methods for large-scale problems proceeding from high order finite elements, where the number of DOF in the interior of the portion of the mesh hosted by any rank is much larger than the number of DOF in its corresponding boundary; thus a local aggregation approach produces good results.

- Since we were aiming for a parallel aggregation algorithm, working locally seems plausible. However, note that the MIS(2) is constructed globally, with many synchronizations (gather/scatter operations) between ranks. After the MIS(2) is constructed, our algorithm does not require any synchronization between ranks, as the `libparanumal` algorithm does; our method merely communicates the final aggregates between ranks at the end of the calculation.

- We even got a speedup using multigrid as preconditioner for the `PCG` solver. It is very unlikely to use pure multigrid for large-scale problems, since we will end up assembling extremely large matrices. A more common approach is use p-multigrid instead, where rather than assembling the extremely large matrices, the problem is reduced algebraically until it is small enough to apply multigrid. Thus we also expect a speedup in the p-multigrid approach.

References

[1] Bell, N., Dalton, S., and Olson, L. Exposing fine-grained parallelism in algebraic multigrid methods. *SIAM Journal on Scientific Computing 34*, 4 (2012), C123–C152.

[2] Brenner, S., and Scott, L. *The Mathematical Theory of Finite Element Methods*. Springer New York, 2002.

[3] Briggs, W., Henson, V., and McCormick, S. *A Multigrid Tutorial: Second Edition*. Society for Industrial and Applied Mathematics, 2000.

[4] Burden, R. L., and Faires, J. D. *Numerical Analysis*, ninth ed. CENGAGE Learning. Brooks/Cole, 2011.

[5] Cerwinsky, D. C. *The Theory and Practice of Algebraic Multigrid Methods*. PhD thesis, University of Wyoming, Laramie, Wyoming, November 2012.

[6] Elman, H., Silvester, D., and Wathen, A. *Finite Elements and Fast Iterative Solvers: with Applications in Incompressible Fluid Dynamics*. Oxford University Press, 2005.

[7] Ern, A., and Guermond, J. *Theory and Practice of Finite Elements*. Springer New York, 2004.

[8] Evans, L. C. *Partial Differential Equations*. American Mathematical Society, 1998.

[9] Gandham, R. *High Performance High-Order Numerical Methods: Applications in Ocean Modeling*. PhD thesis, Rice University, Houston, Texas, May 2015.

[10] Gandham, R., Esler, K., and Zhang, Y. A GPU accelerated aggregation algebraic multigrid method. *Computers & Mathematics with Applications 68*, 10 (2014), 1151 – 1160.

[11] Hackbusch, W. *Multi-Grid Methods and Applications*. Springer-Verlag, Berlin, 1985.

[12] Hesthaven, J. S., and Warburton, T. *Nodal Discontinuous Galerkin Methods: Algorithms, Analysis, and Applications*. Springer, 2010.

[13] Larson, M., and Bengzon, F. *The Finite Element Method: Theory, Implementation, and Applications*, vol. 10. Springer-Verlag, Berlin, 11 2010.

[14] McCormick, S. *Multigrid Methods*. Society for Industrial and Applied Mathematics, 1987.

[15] Notay, Y. Aggregation-based algebraic multilevel preconditioning. *SIAM Journal on Matrix Analysis and Applications 27*, 4 (2006), 998–1018.

[16] Pflaum, C. Lecture Notes: Multigrid Methods. Department of Computer Science in Friederich-Alexander Universität, 2015. URL: https://www.cs10.tf.fau.de/files/2018/06/pflaum-script-mg.pdf. Last visited on 2018/12/07.

[17] Quarteroni, A., Saleri, F., and Gervasio, P. *Scientific Computing with MATLAB and Octave*, fourth ed., vol. 2. Springer, 2014.

[18] Saad, Y. *Iterative Methods for Sparse Linear Systems*, second ed. Society for Industrial and Applied Mathematics, 2003.

[19] Trefethen, L. N., and Bau, D. *Numerical Linear Algebra*. Society for Industrial and Applied Mathematics, 1997.

[20] Treister, E. *Aggregation-based Adaptive Algebraic Multigrid for Sparse Linear Systems*. PhD thesis, Technion Israel Institute of Technology, September 2014.

[21] Vaněk, P., Mandel, J., and Brezina, M. Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems. *Computing 56* (1995), 179–196.

[22] Warburton, T., and Chan, J. Lecture notes: High-order finite elements: Parallel algorithms for extreme-scale simulation. MATH 5414 Virginia Tech, 2018.