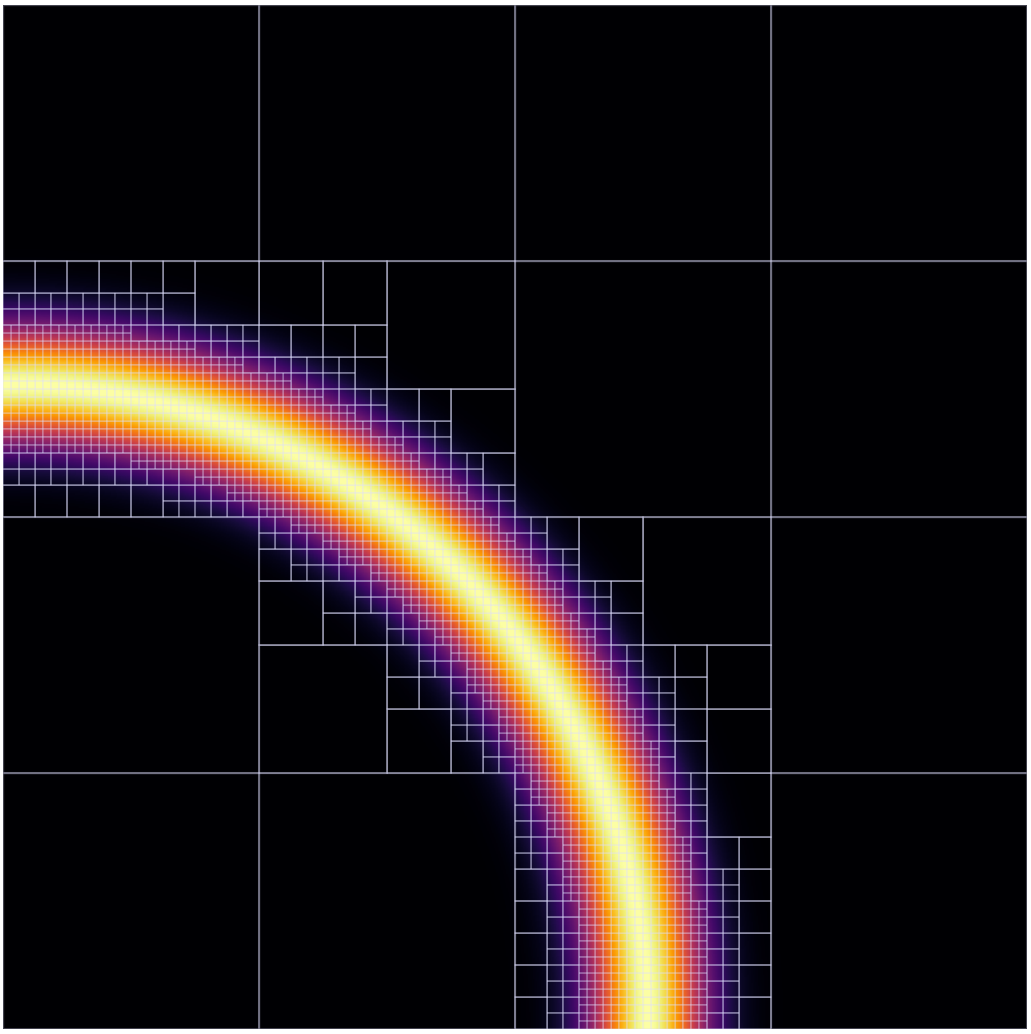
CHALMERS
UNIVERSITY OF TECHNOLOGY



Reusing preconditioners after grid refinement

Thesis for the degree of Master of Science in Physics

ANDREAS KAJLER

IGNITION COMPUTING & DEPARTMENT OF PHYSICS
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2026
www.chalmers.se

Master's thesis 2026

Reusing preconditioners after grid refinement

ANDREAS KAJLER

CHALMERS
UNIVERSITY OF TECHNOLOGY

Ignition Computing
Department of Physics
Chalmers University of Technology
Gothenburg, Sweden 2026

Reusing preconditioners after grid refinement
ANDREAS KAJLER

Supervisor: Alan Cotino, Ignition Computing
Examiner: Christophe Demazière, Department of Physics

Master's Thesis 2026
Ignition Computing
Department of Physics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Adaptive mesh refinement (AMR) grid overlaid on a curved shock front, visualizing the block-structured AMR approach discussed in this thesis.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2026

Reusing preconditioners after grid refinement
ANDREAS KAJLER
Department of Physics
Chalmers University of Technology

Abstract

FLASH, a radiation-MHD simulation code, uses adaptive mesh refinement to concentrate computational effort near features of interest, solving a linear system at each time step using a preconditioned iterative method. After each refinement step the system is rediscretized on the new grid, rendering the old preconditioner incompatible and forcing a rebuild from scratch. This thesis investigates whether the preconditioner can instead be reused via mapping operators derived purely from the old and new grid geometries.

Applied without modification, the mapped preconditioner introduces high-frequency interpolation errors that stall convergence. A composite preconditioner resolves this by surrounding the mapped application with smoothing sweeps that damp these errors.

The method is first developed for a one-dimensional steady-state diffusion model problem. Extensive numerical experiments demonstrate that the composite preconditioner produces grid-independent convergence across diverse physical parameters and grid sizes. The results are robust to changes in the preconditioner, the solver, and the structure of the linear system.

Applied to FLASH, a generalized transition operator handles the block-structured adaptive grid and mixed steps where refinement and derefinement occur simultaneously. An adaptive criterion triggers a rebuild only when convergence degrades past a threshold. In a 2000-step heat diffusion simulation, 1801 steps avoided a rebuild entirely, reducing wall-clock time by 4.4%. Since ILU(0) on a one-dimensional tridiagonal is among the cheapest preconditioners available, the savings are expected to grow substantially for higher-dimensional problems with more expensive builds.

Keywords: adaptive mesh refinement, preconditioner reuse, iterative methods, FLASH, finite volume method, PARAMESH, linear systems.

Acknowledgements

I would like to express my gratitude to my supervisor, Alan Cotino. His invaluable guidance was essential to the completion of this thesis, and his support extended far beyond the project's technical scope.

My sincere thanks go to my examiner, Christophe Demazière, for making this project possible by overseeing my work and providing the crucial academic connection to Chalmers.

A special thank you goes to Daan van Vugt for the opportunity to contribute to this project. His generous support, along with the insight and inspiration sparked by our discussions, has been truly appreciated.

To my colleagues at the office, thank you for giving me an immediate sense of community. Your welcoming company was just as important as any technical assistance, and you made my transition to a new country much smoother.

Lastly, I extend a heartfelt thank you to my family and friends. Your visits and unwavering support from afar provided the exact encouragement I needed to keep my spirits high throughout this journey.

Andreas Kajler, Gothenburg, June 2026

Nomenclature

Below is the nomenclature of symbols and operators used throughout this thesis.

Latin letters

A	System matrix
D	Diagonal part of the matrix splitting $A = D - E - F$
$D(x)$	Diffusion coefficient
E	Strictly lower triangular part of $A = D - E - F$
F	Strictly upper triangular part of $A = D - E - F$
f	Right-hand side vector
G	Iteration matrix
G_{comp}	Iteration matrix of the composite preconditioner
$\bar{H}_k$	Extended upper Hessenberg matrix
I	Identity matrix
$\mathcal{K}_k$	Krylov subspace
L	Lower triangular matrix
$\tilde{L}$	Lower triangular ILU factor
M	Preconditioner matrix
M_{comp}	Composite preconditioner
M_{map}	Mapped preconditioner
M_{old}	Preconditioner on the previous grid
M_{pre}	Pre-smoother matrix
M_{post}	Post-smoother matrix
N_c	Number of coarse-grid cells
N_f	Number of fine-grid cells
n_{cells}	Number of cells per block
n_{root}	Number of root blocks

P	Prolongation operator
R	Restriction operator
r	Residual vector
T	Transition operator
T_k	Symmetric tridiagonal matrix
u	Solution vector
$\tilde{U}$	Upper triangular ILU factor
V_k	Krylov basis matrix
h_i	Distance between adjacent cell centers
$h_{i+1/2}$	Cell width
h_ℓ	Uniform grid spacing at refinement level ℓ

Greek letters

α	FVM stencil subdiagonal coefficient
β	FVM stencil diagonal coefficient
γ	FVM stencil superdiagonal coefficient
δ	Krylov correction vector
$\kappa(A)$	Condition number of A
λ_i	i-th eigenvalue
$\Lambda(A)$	Spectrum of A
ν	Number of smoother sweeps
$\rho(A)$	Spectral radius of A
φ_i	i-th eigenvector
ω	Relaxation parameter

Subscripts and superscripts

$(\cdot)^*$	Exact solution
$(\cdot)^{(k)}$	Quantity at iteration k
$(\cdot)_k$	Krylov subspace dimension
$(\cdot)_c$	Coarse-grid quantity
$(\cdot)_f$	Fine-grid quantity
$(\cdot)_i$	Component at cell index i

$(\cdot)_\ell$	Quantity at refinement level ℓ
$(\cdot)_{\text{int}}$	Intensive variant of a mapping operator
$(\cdot)_{\text{ext}}$	Extensive variant of a mapping operator

Notation conventions

Discrete quantities carry a subscript index, e.g. $u_{i+1/2}$, $f_{i+1/2}$, while continuous quantities are written as explicit functions of x, e.g. $u(x)$, $f(x)$.

Contents

1

Introduction

The global energy transition has elevated nuclear fusion to one of the most compelling scientific challenges of this century. Fusion promises a carbon-free, high-energy-density power source drawing on widely available fuel, and progress toward a working reactor depends on our ability to understand and predict plasma behavior at the extreme temperatures these reactions require. In the *magnetohydrodynamics* (MHD) description of these plasmas, the governing equations form a coupled nonlinear system that is not tractable by analytical methods alone. Numerical simulation is therefore essential for predicting plasma behavior, identifying magnetic instabilities, and guiding reactor design.

FLASH is an open-source code for radiation MHD simulations, used to model complex plasma environments in astrophysical and fusion research contexts [1, p. 25]. The complexity of these environments demands resolution at vastly different length scales simultaneously, and resolving this range on a uniform grid would require a prohibitive number of computational cells. *Adaptive mesh refinement* (AMR) addresses this by concentrating resolution where the physics demands it and derefining elsewhere, dynamically adjusting the grid as the simulation evolves to track features such as shock fronts and thermal gradients.

While fusion simulation provides the immediate motivation for this work, the bottleneck it addresses is not specific to plasma physics. Every AMR step rediscretizes the governing equations on the updated grid, producing a new linear system $Au = f$ whose dimensions differ from the previous one. Solving this system efficiently requires a *preconditioner* M to approximate the system matrix A, and constructing M from scratch for each new system is computationally expensive [2].

The natural question is whether M from the previous step can simply be reused. In principle, it cannot, because after refinement the system dimensions change and the old M does not act on the new solution space at all. Rebuilding from scratch discards all geometric structure encoded in the old preconditioner, even in regions the refinement did not touch. This work instead constructs purely geometric mapping operators from the old and new grid geometries and uses them to map M onto the new grid, retaining what remains valid and adapting what does not. The mapping operators are the standard grid transfer operators from multigrid theory. Prolongation P interpolates a vector from the coarse grid to the fine grid, and restriction R transfers in the opposite direction. Multigrid methods use these operators to build a hierarchy of grids and cycle residuals and corrections across levels, achieving fast convergence through a combination of local smoothing and coarse-grid correction [5]. The present work borrows the same operators for a different purpose, using them to carry M across a single refinement step rather than to construct or traverse a grid hierarchy.

Several approaches to preconditioner reuse have been studied for sequences of linear systems of fixed dimension. Carr, de Sturler, and Gugercin [2] find a sparse approximate map such that the old and new system matrices are nearly equivalent through the existing preconditioner, allowing it to be updated algebraically rather than rebuilt. Singh and Ahuja [3] apply the same algebraic framework to sequences arising in model-order-reduction algorithms. Neither approach extends to AMR, where the system dimensions change at every refinement step. Preconditioning in the pres-

ence of local grid refinement was studied by Bramble, Ewing, Pasciak, and Schatz [4], who construct a domain-decomposition preconditioner for elliptic problems on composite grids and show that the coarse-grid component remains invariant when local refinements are dynamically added or removed. Their method builds a preconditioner suited to the composite structure from scratch at each configuration rather than mapping an existing one to the new grid. Maes and Oswald [5] develop multilevel preconditioners for sequences of non-nested finite element spaces by constructing prolongation operators between adjacent refinement levels, establishing uniformly bounded condition numbers. The present work applies grid transfer operators of the same kind to transport a single preconditioner across one AMR step, rather than to build a multilevel hierarchy.

Chapter 2 develops the theoretical background, covering iterative solvers, preconditioning, and AMR. Chapter 3 develops and validates the mapping approach on a one-dimensional model problem. Chapter 4 transfers the validated approach to FLASH and demonstrates it on more complex configurations. Chapter 5 draws conclusions and identifies directions for future work.

2

Theory

Discretizing a partial differential equation on a computational grid, or linearizing a nonlinear one, produces a sparse linear system whose spectral properties depend on both the physics and the grid. Solving such systems iteratively is practical because iterative methods respect sparsity, but their convergence rate depends on the eigenvalue distribution of A, which deteriorates with grid size. Preconditioning transforms the system to improve the eigenvalue distribution and allows the solver to converge efficiently. When the grid is refined adaptively, the system changes between solves, and the preconditioner must adapt accordingly. Understanding iterative methods, preconditioning, and adaptive mesh refinement is therefore the prerequisite for the analysis in Chapters 3 and 4.

Section 2.1 collects the linear algebra definitions used throughout the thesis. Section 2.2 introduces stationary and Krylov subspace methods. Section 2.3 develops preconditioning, explaining via a polynomial framework why eigenvalue clustering accelerates convergence and which constructions are practical. Section 2.4 describes the block-structured AMR hierarchy and the grid transfer operations that move field values between refinement levels. Readers already familiar with iterative methods and preconditioning may proceed directly to Section 2.4.

2.1 Linear algebra

This section collects the linear algebra definitions and results used throughout the thesis. For a thorough treatment of these topics, see [6].

Eigenvalues and spectrum. A square matrix $A \in \mathbb{R}^{n\times n}$, where n is a positive integer, has a scalar $\lambda \in \mathbb{C}$ as an *eigenvalue* if there exists a nonzero vector $\phi \in \mathbb{C}^n$ satisfying $A\phi = \lambda\phi$, and ϕ is called an *eigenvector* of A associated with λ. The set of all eigenvalues of A is the *spectrum* of A, denoted $\Lambda(A)$, and the *spectral radius* is $\rho(A) \coloneqq \max_{\lambda\in\Lambda(A)} |\lambda|$.

Nonsingular matrices. A square matrix A is *nonsingular*, or *invertible*, if there exists a unique matrix A^{-1} satisfying $AA^{-1} = A^{-1}A = I$. Equivalently, A is nonsingular if and only if $0 \notin \Lambda(A)$.

Norms. A *norm* on $\mathbb{R}^n$ assigns a nonnegative real length to each vector, satisfying positivity ($\|x\| = 0$ if and only if $x = 0$), homogeneity ($\|\alpha x\| = |\alpha|\|x\|$ for any scalar $\alpha \in \mathbb{R}$), and the triangle inequality ($\|x + y\| \le \|x\| + \|y\|$). The two norms used most frequently in this thesis are the *2-norm* $\|x\|_2 \coloneqq \sqrt{\sum_{i=1}^n x_i^2}$ and the *infinity norm* $\|x\|_\infty \coloneqq \max_{1\le i\le n} |x_i|$.

Inner products and orthonormality. The *inner product* of two vectors $x, y \in \mathbb{R}^n$ is $\langle x, y\rangle \coloneqq x^T y = \sum_{i=1}^n x_i y_i$, and satisfies $\|x\|_2 = \sqrt{\langle x, x\rangle}$. Two vectors are *orthogonal* if $\langle x, y\rangle = 0$. A set

of vectors $\{v_1, \ldots, v_k\}$ is *orthonormal* if $\langle v_i, v_j \rangle = \delta_{ij}$ for all i, j, where δ_{ij} is the Kronecker delta, equal to 1 if $i = j$ and 0 otherwise. Storing these vectors as columns of a matrix $V \in \mathbb{R}^{n \times k}$, the orthonormality condition reads $V^T V = I_k$.

Symmetric positive definite matrices. A real matrix A is *symmetric* if $A = A^T$. A symmetric matrix is *symmetric positive definite* (SPD) if $x^T A x > 0$ for all nonzero $x \in \mathbb{R}^n$. The eigenvalues of a symmetric matrix are real, and an SPD matrix has strictly positive eigenvalues. A symmetric matrix B is *positive semidefinite* if $y^T B y \geq 0$ for all $y \in \mathbb{R}^n$.

Range and null space. The *range* of a matrix $A \in \mathbb{R}^{m \times n}$, written $\text{range}(A) \coloneqq \{Ax \mid x \in \mathbb{R}^n\}$, is the set of all vectors reachable by A. The *null space* of A, written $\text{null}(A) \coloneqq \{x \in \mathbb{R}^n \mid Ax = 0\}$, is the set of all vectors mapped to zero by A.

Projection operators. A square matrix P is a *projection* if it is *idempotent*, meaning $P^2 = P$. Any vector v can be decomposed as $v = Pv + (I - P)v$, where $Pv \in \text{range}(P)$ and $(I - P)v \in \text{null}(P)$. This decomposition is unique and is called a *direct sum*, written $\mathbb{R}^n = \text{range}(P) \oplus \text{null}(P)$. A consequence of the idempotency is that $\text{null}(P) = \text{range}(I - P)$. If additionally $P = P^T$, the projection is *orthogonal* and its range and null space are orthogonal.

Triangular matrices. A square matrix $L \in \mathbb{R}^{n \times n}$ is *lower triangular* if all entries above the main diagonal are zero, that is, $L_{ij} = 0$ for $j > i$. Similarly, $U \in \mathbb{R}^{n \times n}$ is *upper triangular* if $U_{ij} = 0$ for $i > j$.

Hessenberg matrices. A square matrix $H \in \mathbb{R}^{n \times n}$ is *upper Hessenberg* if all entries below the first subdiagonal are zero, that is, $H_{ij} = 0$ for $i > j + 1$. An *extended* upper Hessenberg matrix $\bar{H} \in \mathbb{R}^{(n+1) \times n}$ has the same sparsity pattern with one additional row appended at the bottom.

Condition number. The *condition number* of a nonsingular matrix A is $\kappa(A) \coloneqq \|A\|_2 \|A^{-1}\|_2$ and quantifies how sensitively the solution u of $Au = f$ responds to perturbations in f, as a large $\kappa(A)$ means small changes in f can produce large changes in u. For a symmetric matrix the 2-norm equals the spectral radius, $\|A\|_2 = \rho(A) = \max_i |\lambda_i|$, so $\kappa(A) = |\lambda_{\max}|/|\lambda_{\min}|$. For an SPD matrix the eigenvalues are strictly positive, so the absolute values can be dropped, giving $\kappa(A) = \lambda_{\max}/\lambda_{\min}$. As discussed in subsequent sections, $\kappa(A)$ is one measure of how the eigenvalue distribution affects the convergence speed of iterative solvers.

2.2 Iterative methods

Solving linear systems of the form $Au = f$ is a central task in numerical physics. Small systems can be solved efficiently using direct methods, such as Gaussian elimination or LU factorization, which find the exact solution in a finite number of steps. However, the computational work for direct factorization of a general dense system scales as $\mathcal{O}(N^3)$ [6, Eq. 20.8]. Even for sparse matrices, direct solvers create new nonzero values in positions that were originally empty. This effect, known as *fill-in*, significantly increases the storage cost of the factorization [7, p. 96]. Iterative methods avoid this by using matrix-vector products that leave the system matrix A unchanged. This preserves the original sparsity and keeps memory demands low.

Instead of a single exact calculation, these methods evolve an initial guess $u^{(0)}$ through a sequence of approximations $u^{(k)}$ toward the exact solution u^*. Each step improves the result by reducing the residual $r = f - Au$, which measures how far the current iterate is from satisfying $Au = f$.

The price for this memory efficiency is a loss of predictability. Iterative solvers do not guarantee an exact solution in a fixed number of steps. The convergence rate depends on the eigenvalue distribution of A, as made precise in Section 2.3.1. When the distribution is widely dispersed, reflected in a large $\kappa(A)$, convergence can be prohibitively slow. To fix this, we later introduce preconditioning to improve the eigenvalue distribution of the transformed system without changing the final answer. Section 2.2.1 covers stationary methods derived from matrix splittings, and Section 2.2.2 turns to Krylov subspace methods. Readers familiar with iterative solvers may proceed directly to Section 2.3.

2.2.1 Stationary methods

While direct solvers attempt to satisfy the entire system $Au = f$ simultaneously, stationary iterative methods employ a constant transition rule that reduces the residual $r = f - Au$ by updating the approximate solution vector u component-wise [7, p. 105]. The following subsections introduce the Jacobi, Gauss-Seidel, successive over-relaxation (SOR), and symmetric successive over-relaxation (SSOR) methods, each extending the previous to improve convergence speed. Readers already familiar with these methods need only the matrix splitting framework (2.2) and the error propagation result (2.18).

Jacobi method

The Jacobi method updates all components of u simultaneously, using only information from the previous iterate $u^{(k)}$. Since no component influences another within the same step, the trajectory in residual space consists of a single leap from the old residual state to a new one, as illustrated in Figure 2.1.

To derive its vector form, we decompose the matrix as $A = D - E - F$, where D contains the nonzero diagonal entries, $-E$ is the strictly lower triangular part, and $-F$ is the strictly upper triangular part [7, Eq. 4.2]. We start from the i-th component of the residual

$$r_i = f_i - \sum_{j=1}^{n} A_{ij} u_j = 0.$$

Isolating u_i directly would create a circular dependency, since computing u_i requires knowing every other u_j. Instead, we treat the update as a sequence of approximations, using the previous iterate $u^{(k)}$ for all off-diagonal terms and solving for the new component $u_i^{(k+1)}$,

$$\begin{aligned} 0 &= f_i - \sum_{j=1}^{i-1} A_{ij} u_j^{(k)} - A_{ii} u_i^{(k+1)} - \sum_{j=i+1}^{n} A_{ij} u_j^{(k)} \\ &= f_i + \sum_{j=1}^{n} E_{ij} u_j^{(k)} - D_{ii} u_i^{(k+1)} + \sum_{j=1}^{n} F_{ij} u_j^{(k)}. \end{aligned}$$

Stacking these n rows back into a single system gives the vector update rule [7, Eq. 4.5]

$$u^{(k+1)} = D^{-1}(E + F)u^{(k)} + D^{-1} f. \tag{2.1}$$

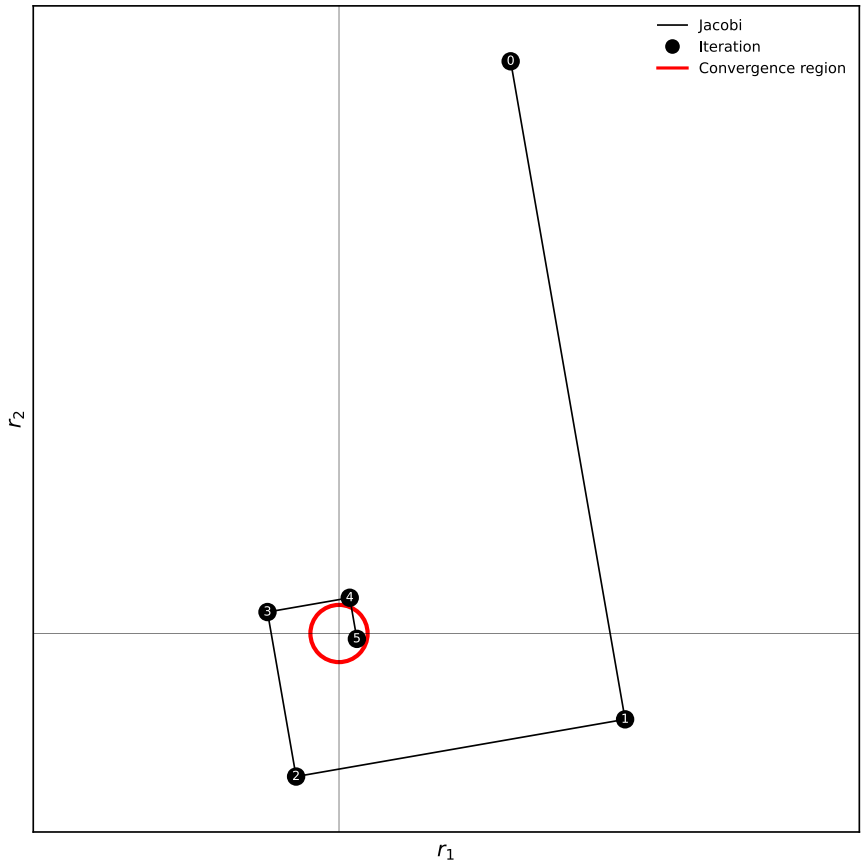


Figure 2.1: Iterative progress by the Jacobi method in residual space for a 2×2 model system defined in Appendix B.

This confirms its classification as a stationary method since the transition is governed by the constant iteration matrix $D^{-1}(E + F)$, applied identically at every step.

The Jacobi iteration is the first instance of the *matrix splitting framework* [7, Eq. 4.11]. Any decomposition $A = M - N$ defines a stationary iteration

$$u^{(k+1)} = M^{-1}Nu^{(k)} + M^{-1}f, \tag{2.2}$$

and Jacobi corresponds to the choice $M = D$. Such a method converges to u^* if and only if the spectral radius satisfies $\rho(M^{-1}N) < 1$ [7, Thm. 4.1].

We can extend this approach by introducing a relaxation parameter ω to control the step size, resulting in the *weighted Jacobi method*. Denoting the standard Jacobi iterate from (2.1) as u^J, the weighted variant takes a combination of the previous iterate and the Jacobi step,

$$u^{(k+1)} = \omega u^J + (1 - \omega)u^{(k)}. \tag{2.3}$$

Substituting $u^J = D^{-1}(E + F)u^{(k)} + D^{-1}f$ gives the vector update rule [9, p. 9],

$$u^{(k+1)} = \left[(1 - \omega)I + \omega D^{-1}(E + F)\right] u^{(k)} + \omega D^{-1}f. \tag{2.4}$$

Within the splitting framework this corresponds to $M = \frac{1}{\omega}D$. Setting $\omega = 1$ recovers the standard Jacobi method, while $\omega < 1$ takes a more conservative partial step.

Gauss-Seidel method

The Gauss-Seidel method is built on the same goal of annihilating the residual component-wise, but it incorporates newly computed values immediately rather than waiting until the full sweep is complete. Specifically, the method uses $u_j^{(k+1)}$ for all $j < i$ already computed in the current sweep, while the remaining components $u_j^{(k)}$ $(j > i)$ are still taken from the previous iterate. Since each update drives the i-th residual component to zero, every intermediate point in the trajectory lands on a residual axis, as shown in Figure 2.2.

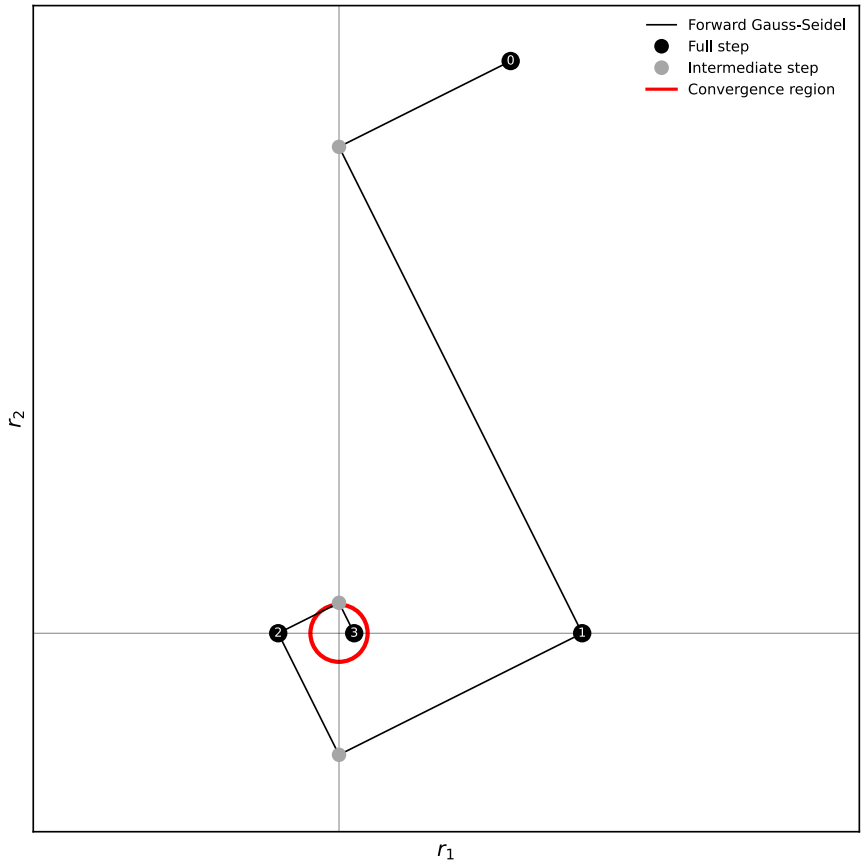


Figure 2.2: Iterative progress by the forward Gauss-Seidel method in residual space for the 2×2 model system defined in Appendix B.

Algebraically, this alters the i-th component of the residual

$$\begin{aligned} 0 &= f_i - \sum_{j=1}^{i-1} A_{ij} u_j^{(k+1)} - A_{ii} u_i^{(k+1)} - \sum_{j=i+1}^{n} A_{ij} u_j^{(k)} \\ &= f_i + \sum_{j=1}^{n} E_{ij} u_j^{(k+1)} - D_{ii} u_i^{(k+1)} + \sum_{j=1}^{n} F_{ij} u_j^{(k)}. \end{aligned}$$

Collecting all $u^{(k+1)}$ terms on the left gives the vector update rule [7, Eqs. 4.8–4.9]

$$u^{(k+1)} = (D - E)^{-1} F u^{(k)} + (D - E)^{-1} f. \tag{2.5}$$

This is the *forward Gauss-Seidel* iteration, where the unknowns are updated from $1, \ldots, n$. Reversing this order defines the *backward Gauss-Seidel* iteration,

$$u^{(k+1)} = (D - F)^{-1} E u^{(k)} + (D - F)^{-1} f. \tag{2.6}$$

The vector forms above are primarily analytical tools for studying convergence. In practice, computing an explicit inverse such as $(D - E)^{-1}$ would produce a dense matrix and eliminate the sparsity savings of the iterative approach. Instead, the method applies the component-wise summation formula directly, which is algorithmically equivalent to forward substitution.

Within the splitting framework, forward Gauss-Seidel corresponds to $M = D - E$ and backward Gauss-Seidel to $M = D - F$.

Successive over-relaxation

While the Gauss-Seidel method improves upon Jacobi by utilizing updated components immediately, its convergence can still be slow. The successive over-relaxation (SOR) method seeks to accelerate this convergence by introducing a relaxation parameter ω that controls how large a step is taken in the Gauss-Seidel direction. Rather than simply accepting the Gauss-Seidel update u_i^{GS}, the SOR method computes the new component $u_i^{(k+1)}$ as a weighted combination of the previous iterate $u_i^{(k)}$ and the Gauss-Seidel value,

$$u_i^{(k+1)} = \omega u_i^{GS} + (1 - \omega) u_i^{(k)}. \tag{2.7}$$

Choosing $\omega > 1$ extrapolates beyond the Gauss-Seidel step, while $\omega < 1$ takes a more conservative partial step. Figure 2.3 shows example residual trajectories for $\omega = 0.9$ and $\omega = 1.1$ applied to the same test problem as in Figures 2.1 and 2.2. In this particular instance the under-relaxed value $\omega = 0.9$ reaches the solution in fewer steps than the over-relaxed $\omega = 1.1$, which may seem counterintuitive given the method's name. The choice of relaxation parameter is highly system-dependent and will be revisited when analyzing numerical results.

To derive the vector update rule, we recall the component-wise Gauss-Seidel formula

$$D_{ii} u_i^{GS} = f_i + \sum_{j=1}^{n} E_{ij} u_j^{(k+1)} + \sum_{j=1}^{n} F_{ij} u_j^{(k)}. \tag{2.8}$$

Substituting into the weighted average and multiplying both sides by D_{ii} gives

$$D_{ii} u_i^{(k+1)} = \omega \left(f_i + \sum_{j=1}^{n} E_{ij} u_j^{(k+1)} + \sum_{j=1}^{n} F_{ij} u_j^{(k)} \right) + (1 - \omega) D_{ii} u_i^{(k)}. \tag{2.9}$$

Stacking these n equations and collecting all $u^{(k+1)}$ terms on the left yields

$$(D - \omega E) u^{(k+1)} = (\omega F + (1 - \omega) D) u^{(k)} + \omega f. \tag{2.10}$$

Solving for $u^{(k+1)}$ gives the vector update rule for the *forward SOR* iteration [7, Eq. 4.12],

$$u^{(k+1)} = (D - \omega E)^{-1} (\omega F + (1 - \omega) D) u^{(k)} + \omega (D - \omega E)^{-1} f. \tag{2.11}$$

Reversing the sweep order from $i = n$ down to 1 yields the *backward SOR* iteration,

$$u^{(k+1)} = (D - \omega F)^{-1} (\omega E + (1 - \omega) D) u^{(k)} + \omega (D - \omega F)^{-1} f. \tag{2.12}$$

Within the splitting framework, the forward SOR corresponds to $M = \frac{1}{\omega}(D - \omega E)$ and the backward SOR to $M = \frac{1}{\omega}(D - \omega F)$. Setting $\omega = 1$ recovers the respective Gauss-Seidel iterations exactly.

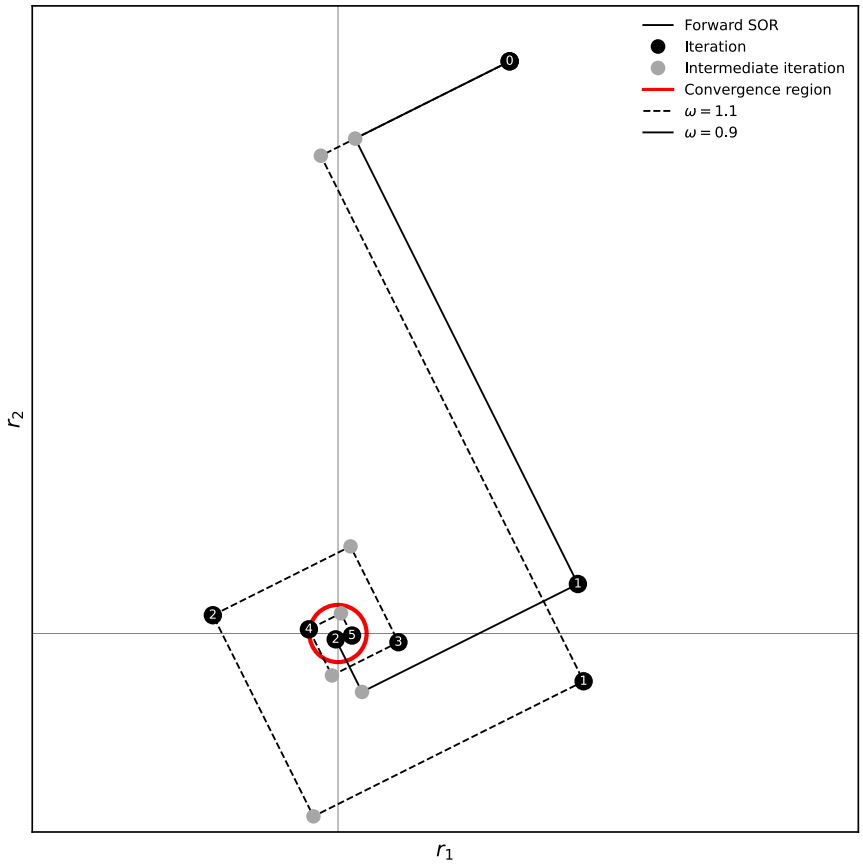


Figure 2.3: Iterative progress by the forward SOR method in residual space for $\omega = 0.9$ (under-relaxation) and $\omega = 1.1$ (over-relaxation), applied to the 2×2 model system defined in Appendix B.

Symmetric successive over-relaxation

While the standard SOR method successfully accelerates convergence, for symmetric systems its reliance on a specific substitution direction breaks the symmetry of the iteration matrix. The symmetric successive over-relaxation (SSOR) method restores this symmetry by pairing a forward SOR sweep with a backward SOR sweep. Each full iteration consists of two half-steps. The first half-step computes an intermediate vector $u^{(k+1/2)}$ using the forward SOR update, sweeping from $i = 1$ to n,

$$(D - \omega E)u^{(k+1/2)} = (\omega F + (1 - \omega)D)u^{(k)} + \omega f, \tag{2.13}$$

and the second applies the backward SOR update, sweeping from $i = n$ down to 1,

$$(D - \omega F)u^{(k+1)} = (\omega E + (1 - \omega)D)u^{(k+1/2)} + \omega f. \tag{2.14}$$

Substituting the first equation into the second and collecting terms gives the one-step vector update rule

$$\begin{aligned} u^{(k+1)} = {} & (D - \omega F)^{-1}(\omega E + (1 - \omega)D)(D - \omega E)^{-1}(\omega F + (1 - \omega)D)u^{(k)} \\ & + \omega(2 - \omega)(D - \omega F)^{-1}D(D - \omega E)^{-1}f. \end{aligned} \tag{2.15}$$

Reading off the splitting matrix gives [7, Eq. 4.27]

$$M = \frac{1}{\omega(2 - \omega)}(D - \omega E)D^{-1}(D - \omega F), \tag{2.16}$$

provided $0 < \omega < 2$ so that the method is well-defined.

Residual and error propagation

With splitting matrices now established for all four methods, we can characterize their shared convergence behavior. Rewriting the splitting iteration (2.2) using the residual $r^{(k)} = f - Au^{(k)}$ gives

$$u^{(k+1)} = u^{(k)} + M^{-1}r^{(k)}.$$

Multiplying the update rule by A and subtracting from f shows how the residual evolves,

$$r^{(k+1)} = (I - AM^{-1})r^{(k)}.$$

Unfolding this recurrence back to the initial state gives

$$r^{(k)} = (I - AM^{-1})^k r^{(0)}. \tag{2.17}$$

Each stationary method therefore constructs a polynomial in AM^{-1} applied to the initial residual $r^{(0)}$, with coefficients fixed by the binomial expansion rather than optimized for the specific system. The same correction step also determines how the error $e^{(k)} = u^* - u^{(k)}$ propagates. Since $r^{(k)} = f - Au^{(k)} = Ae^{(k)}$, replacing $r^{(k)}$ in the update and subtracting from u^* gives

$$e^{(k+1)} = (I - M^{-1}A)\, e^{(k)}, \tag{2.18}$$

so the error is contracted at each step by the *iteration matrix* $G \coloneqq I - M^{-1}A$.

2.2.2 Krylov subspace methods

The previous section established that stationary methods construct a polynomial in AM^{-1} acting on the initial residual $r^{(0)}$. We can view this polynomial as a linear combination of basis vectors formed by applying ascending powers of a matrix to $r^{(0)}$. Stationary methods effectively build a vector space but restrict themselves to one predetermined point within it. Krylov subspace methods improve upon this by searching that entire vector space for the $u^{(k)}$ that best approximates u^*, in a sense made precise below. Readers already familiar with GMRES and CG may focus on the minimization problem (2.23) and the Galerkin condition (2.25), which recur in later sections.

To understand how to search this space, we first need to define the space itself. By repeatedly applying A to the initial residual $r^{(0)}$, we generate a sequence of basis vectors. The span of these vectors forms the Krylov subspace [7, §6.2]

$$\mathcal{K}_k(A, r^{(0)}) \coloneqq \operatorname{span}\{r^{(0)}, Ar^{(0)}, A^2r^{(0)}, \ldots, A^{k-1}r^{(0)}\}. \tag{2.19}$$

Building this subspace requires only sequential matrix-vector multiplications, which preserves the sparse structure of A and avoids the memory fill-in characteristic of direct methods. Furthermore, unlike a generic basis that is independent of the physics of the problem, the Krylov subspace intrinsically incorporates it. Repeatedly multiplying by A amplifies the residual's components along the matrix's dominant eigenmodes. By retaining this entire sequence of vectors as a search space, the solver reduces the residual contributions along the dominant eigenvectors first, achieving convergence within a smaller subspace.

Generalized minimal residual

Having established the Krylov subspace $\mathcal{K}_k(A, r^{(0)})$ as our search space, the next challenge is determining which vector within it best approximates u^*. The generalized minimal residual (GMRES) method resolves this by framing the selection process as a minimization problem [8, §3.1]. It seeks the approximation $u^{(k)} \in u^{(0)} + \mathcal{K}_k$ that minimizes the 2-norm of the residual, $\|r^{(k)}\|_2$. Letting $\delta^{(k)} \in \mathcal{K}_k$ denote the correction vector, we can write the residual

$$r^{(k)} = f - Au^{(k)} = f - A(u^{(0)} + \delta^{(k)}) = r^{(0)} - A\delta^{(k)}. \tag{2.20}$$

Because the search space restricts $\delta^{(k)}$ to $\mathcal{K}_k$, the resulting vector $A\delta^{(k)}$ is confined to the transformed space $A\mathcal{K}_k$. Geometrically, minimizing this residual distance requires $r^{(k)} \perp A\mathcal{K}_k$, which is called the *Petrov-Galerkin condition* [7, Prop. 5.3]. GMRES thus reduces the task of solving a large linear system down to a smaller, k-dimensional least-squares problem over the Krylov subspace.

While the basis provided by $\mathcal{K}_k$ technically defines the search space and targets the largest errors, using it directly leads to numerical instability. As established, the successive vectors tend to align with the dominant eigenvector, causing them to eventually become nearly linearly dependent, which is problematic when they are represented in finite precision arithmetic. An orthonormal basis spanning the same subspace would prevent rounding errors from corrupting the solution by ensuring that each vector captures a distinct, independent direction [6, p. 253].

GMRES systematically enforces this orthogonality using a *Gram-Schmidt* (GS) process. It starts by normalizing the first vector in the Krylov subspace $v_1 = r^{(0)}/\|r^{(0)}\|_2$. For the next vector, it projects the new direction Av_1 onto v_1, subtracts this overlapping component to isolate the orthogonal direction $w_2 = Av_1 - \langle Av_1, v_1\rangle v_1$, and normalizes the result to yield $v_2 = w_2/\|w_2\|_2$ (Figure 2.4). For any step i, this orthogonalization of the newly generated vector Av_i against all previously computed basis vectors is given by [7, Algorithm 1.1]

$$w_{i+1} \coloneqq Av_i - \sum_{j=1}^{i} \langle Av_i, v_j\rangle v_j. \tag{2.21}$$

The new orthonormal basis vector is obtained by normalizing this remainder $v_{i+1} = w_{i+1}/\|w_{i+1}\|_2$.

In practice, GMRES employs the *Modified Gram-Schmidt* (MGS) algorithm, which subtracts each projection sequentially to reduce the accumulation of numerical errors [7, Algorithm 1.2].

If we denote the inner product by $h_{j,i} \coloneqq \langle Av_i, v_j\rangle$ and define the length $h_{i+1,i} \coloneqq \|w_{i+1}\|_2$, we can write equation (2.21) as

$$h_{i+1,i}v_{i+1} = Av_i - \sum_{j=1}^{i} h_{j,i}v_j.$$

Collecting the terms under a single sum gives us the vector equation

$$Av_i = \sum_{j=1}^{i+1} h_{j,i}v_j.$$

By executing this process for $i = 1, \ldots, k$ steps and storing the orthonormal basis vectors v_i as columns in a matrix V_k, we arrive at the *Arnoldi relation* [7, Prop. 6.5, Eq. 6.7]

$$AV_k = V_{k+1}\bar{H}_k. \tag{2.22}$$

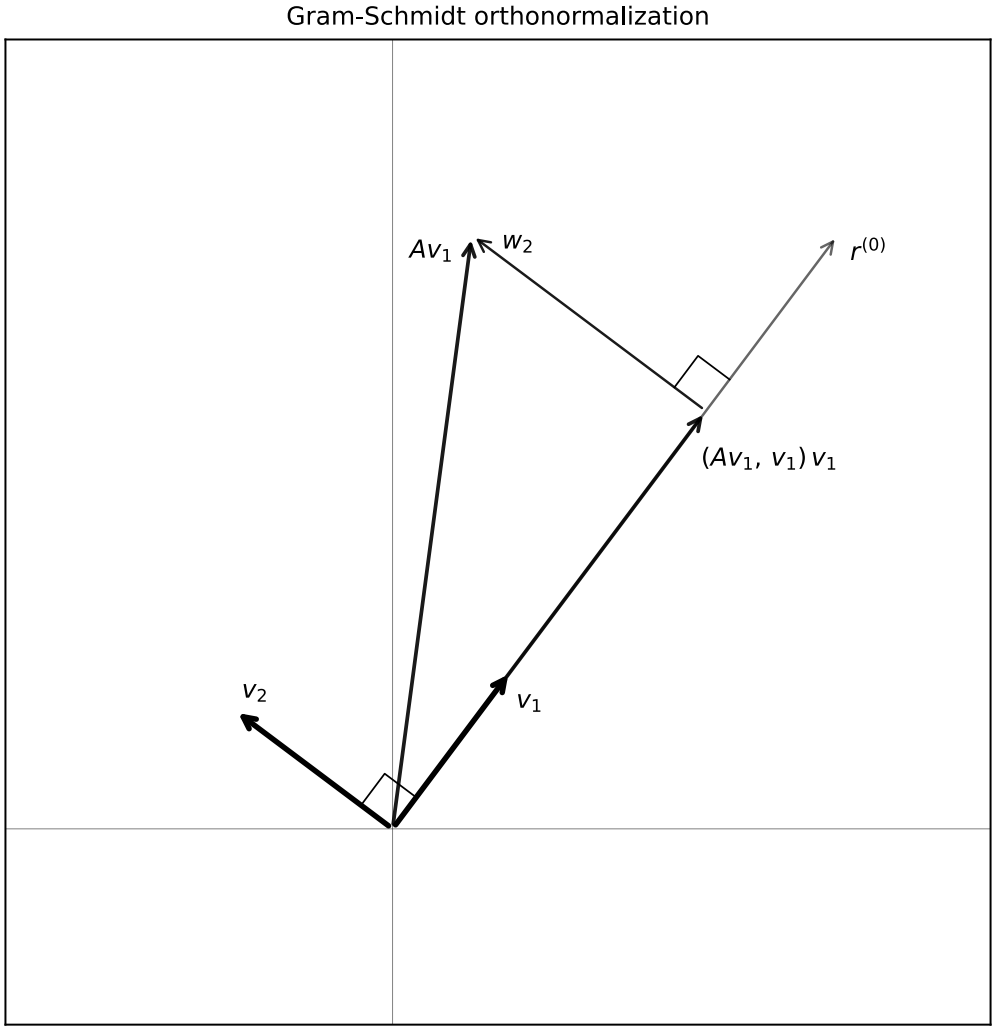


Figure 2.4: Schematic of the Gram-Schmidt orthogonalization process. The algorithm constructs a custom orthonormal basis (v_1, v_2) for the Krylov subspace by normalizing the orthogonal component (w_2) of the new search direction (Av_1).

Here, $\bar{H}_k$ is a $(k+1)\times k$ matrix whose upper triangular part contains the inner products $h_{j,i}$, while the lower subdiagonal contains the lengths $h_{i+1,i}$. This structure is called an *extended upper Hessenberg matrix*.

Since the Krylov subspace $\mathcal{K}_k$ is spanned by the basis V_k, we can write the correction vector that lives in it as $\delta^{(k)} = V_k y$, where y is a k-dimensional vector of coefficients. Continuing on equation (2.20) by using this and equation (2.22) gives us

$$r^{(k)} = r^{(0)} - AV_k y = r^{(0)} - V_{k+1}\bar{H}_k y.$$

Recall that the very first basis vector is defined by normalizing the initial residual, $r^{(0)} = \|r^{(0)}\|_2 v_1$. To factor out V_{k+1}, we express this first basis vector as $v_1 = V_{k+1}e_1$, where $e_1 = (1, 0, \ldots, 0)^T$ is the first standard basis vector in $\mathbb{R}^{k+1}$. Factoring out the basis matrix yields

$$r^{(k)} = V_{k+1}(\|r^{(0)}\|_2 e_1 - \bar{H}_k y).$$

The objective of GMRES is to find the vector y that minimizes the 2-norm of this residual $\|r^{(k)}\|_2$. Because multiplying a vector by a matrix with orthonormal columns (like V_{k+1}) does not change its 2-norm, the minimization problem simplifies to

$$\min_y \|r^{(k)}\|_2 = \min_y \left\| \|r^{(0)}\|_2 e_1 - \bar{H}_k y \right\|_2. \tag{2.23}$$

The initially large linear system $Au = f$ has thus been reduced to a small $(k+1) \times k$ least-squares problem. By finding the coefficients y that minimize the residual norm, the resulting residual $r^{(k)}$ will be orthogonal to AV_k. Since AV_k spans $A\mathcal{K}_k$, this directly satisfies the Petrov-Galerkin condition $(r^{(k)} \perp A\mathcal{K}_k)$.

The basis matrix V_{k+1}, however, acquires one new column at every iteration, so both memory and the cost of the least-squares solve grow with k. In practice, GMRES is therefore restarted every m iterations, discarding the accumulated subspace and resuming from the current iterate to keep memory bounded [7, §6.5.5].

Ultimately, the specific quantity we choose to minimize dictates the required orthogonality condition. While GMRES minimizes the residual, shifting the objective to minimize the error $e^{(k)} = u^* - u^{(k)}$ results in the standard Galerkin condition $(r^{(k)} \perp \mathcal{K}_k)$ instead. The following section explores the mathematical advantages and practical trade-offs of this approach.

Conjugate gradient

The preceding section concluded by shifting the minimization objective from the residual to the error $e^{(k)} = u^* - u^{(k)}$. The difficulty is that evaluating $e^{(k)}$ directly requires the unknown u^*. We resolve this by measuring the error in the A-norm, defined as

$$\|e\|_A \coloneqq \sqrt{e^T A e}. \tag{2.24}$$

Multiplying the error by A allows us to substitute the unknown Au^* with the known right-hand side f. For $\|e\|_A$ to be a valid norm, A must be symmetric ($A = A^T$) and positive definite ($e^T A e > 0$ for all $e \neq 0$). Consequently, CG is limited to symmetric positive definite (SPD) systems.

We seek the iterate $u^{(k)} \in u^{(0)} + \mathcal{K}_k$ that minimizes $\|e^{(k)}\|_A$. Writing the correction as $\delta^{(k)} \in \mathcal{K}_k$ gives $e^{(k)} = e^{(0)} - \delta^{(k)}$. For the error to be minimized over the subspace, its derivative in any allowed search direction $w \in \mathcal{K}_k$ must vanish, yielding the condition $w^T A e^{(k)} = 0$.

Here the computational payoff of the A-norm is realized. Since $Ae^{(k)} = Au^* - Au^{(k)} = f - Au^{(k)} = r^{(k)}$, the unknown error is replaced by the computable residual. The condition $w^T A e^{(k)} = 0$ thus becomes $w^T r^{(k)} = 0$ for all $w \in \mathcal{K}_k$, meaning the residual must be orthogonal to the entire search space [7, §6.4],

$$r^{(k)} \perp \mathcal{K}_k. \tag{2.25}$$

Whereas GMRES enforced orthogonality of the residual against the transformed space $A\mathcal{K}_k$, CG enforces it against the subspace itself. This condition will reappear in the analysis of the preconditioned system.

The symmetry that SPD demands yields a dramatic reduction in the orthogonalization cost of building the Krylov basis. Recall that each new vector Av_i in the Arnoldi process contributes a full column of inner products $h_{j,i} = \langle Av_i, v_j \rangle$ to the Hessenberg matrix. Rewriting these as $h_{j,i} = v_j^T A v_i = (Av_j)^T v_i$ and noting that $Av_j \in \operatorname{span}\{v_1, \ldots, v_{j+1}\}$, orthogonality of the basis gives $h_{j,i} = 0$ for $i > j + 1$. All but two inner products per step vanish, collapsing the Hessenberg matrix to a symmetric tridiagonal one. This short recurrence is the *Lanczos process* [7, Thm. 6.19], the direct

algebraic payoff of requiring A to be SPD. The Arnoldi relation $AV_k = V_{k+1}\bar{H}_k$ therefore simplifies to the *Lanczos relation*

$$AV_k = V_{k+1}\bar{T}_k, \tag{2.26}$$

where $\bar{T}_k$ is a $(k+1) \times k$ matrix with nonzero entries only on the main diagonal and its two adjacent off-diagonals. Multiplying (2.26) by V_k^T from the left and using $V_k^T V_{k+1} = [I_k \ 0]$, which holds because the columns of V_{k+1} are orthonormal, yields $V_k^T A V_k = T_k$, where T_k is a $k \times k$ symmetric tridiagonal matrix.

Writing the correction as $\delta^{(k)} = V_k y$, the residual becomes $r^{(k)} = r^{(0)} - AV_k y$. Enforcing the Galerkin condition (2.25), and using $V_k^T A V_k = T_k$ and $V_k^T r^{(0)} = \|r^{(0)}\|_2 e_1$, reduces the problem to the $k \times k$ symmetric tridiagonal system

$$T_k y = \|r^{(0)}\|_2 e_1. \tag{2.27}$$

Storing the full basis V_k and solving this system at the end would require memory growing with k. CG instead solves on the fly via a bidiagonal factorization of T_k. Because the factorization is bidiagonal, each step draws only on the current and previous Lanczos vectors, giving both a short recurrence and bounded storage. Denoting the vectors produced by this factorization as $\{p^{(k)}\}$, they are mutually A-orthogonal [7, Prop. 6.20],

$$\langle p^{(i)}, Ap^{(j)} \rangle = 0, \quad i \neq j, \tag{2.28}$$

a property known as *conjugacy* that ensures each one-dimensional minimization is independent of all previous ones. This decoupling reduces the algorithm to two scalar inner products per iteration. One computes the step length that minimizes $\|e\|_A$ along the current direction, and the other propagates A-orthogonality to the next, together replacing the growing least-squares solve of GMRES. The Petrov-Galerkin condition for GMRES and the Galerkin condition for CG both reappear when the system is preconditioned, which is the subject of the next section.

2.3 Preconditioning

The convergence rate of iterative solvers is governed by the eigenvalue distribution of the matrix A. When this spectrum is tightly clustered, the solver can reduce the residual rapidly, whereas a wide and dispersed spectrum degrades its performance. To address this, we introduce a preconditioner M, an operator that transforms the linear system into an equivalent one with a more tightly clustered eigenvalue distribution [6, p. 314]. Section 2.3.1 formalizes why this clustering accelerates convergence, Section 2.3.2 contrasts left and right preconditioning strategies, and Section 2.3.3 surveys common preconditioner constructions and their computational trade-offs.

2.3.1 The polynomial framework

To establish exactly why the eigenvalue distribution affects convergence, we will use a polynomial framework. As discussed in previous sections, iterative methods operate by constructing a matrix polynomial that acts on the initial residual. While stationary methods apply a fixed polynomial dictated by their matrix splitting $A = M - N$, Krylov methods dynamically construct an optimal polynomial to minimize a specific error metric [7, p. 158, p. 171]. The Krylov correction vector $\delta^{(k)}$,

which belongs to $\mathcal{K}_k(A, r^{(0)}) = \text{span}\{r^{(0)}, Ar^{(0)}, \ldots, A^{k-1}r^{(0)}\}$, is a linear combination of its basis vectors. This means there exist coefficients $C_0, \ldots, C_{k-1}$ such that

$$\delta^{(k)} = \Big(C_0 I + C_1 A + \cdots + C_{k-1}A^{k-1}\Big)r^{(0)} = q_{k-1}(A)\, r^{(0)},$$

where q_{k-1} is a polynomial of degree at most $k-1$. Substituting the updated solution $u^{(k)} = u^{(0)} + \delta^{(k)}$ into the residual expression in equation (2.20) yields

$$r^{(k)} = r^{(0)} - Aq_{k-1}(A)\, r^{(0)} = p_k(A)\, r^{(0)}. \tag{2.29}$$

We can define a new matrix polynomial $p_k(A) \coloneqq I - Aq_{k-1}(A)$. To understand how this matrix polynomial behaves, we can evaluate its effect on the individual eigenvectors of A. Assuming A has a complete set of linearly independent eigenvectors $\phi_1, \ldots, \phi_n$ with corresponding eigenvalues $\lambda_1, \ldots, \lambda_n$, we can express the initial residual as a linear combination of these distinct directions

$$r^{(0)} = \sum_{i=1}^{n} c_i \phi_i,$$

where c_i represents the projection of the initial residual onto each eigenvector. Substituting this expansion into equation (2.29) gives

$$r^{(k)} = p_k(A) \sum_{i=1}^{n} c_i \phi_i = \sum_{i=1}^{n} c_i p_k(A) \phi_i.$$

The eigenvalue relation $A\phi_i = \lambda_i \phi_i$ dictates that applying the matrix A repeatedly simply scales the eigenvector by powers of its eigenvalue, meaning $A^j \phi_i = \lambda_i^j \phi_i$. Consequently, any matrix polynomial acting on an eigenvector simplifies to a scalar polynomial evaluated at the corresponding eigenvalue

$$p_k(A)\phi_i = p_k(\lambda_i)\phi_i.$$

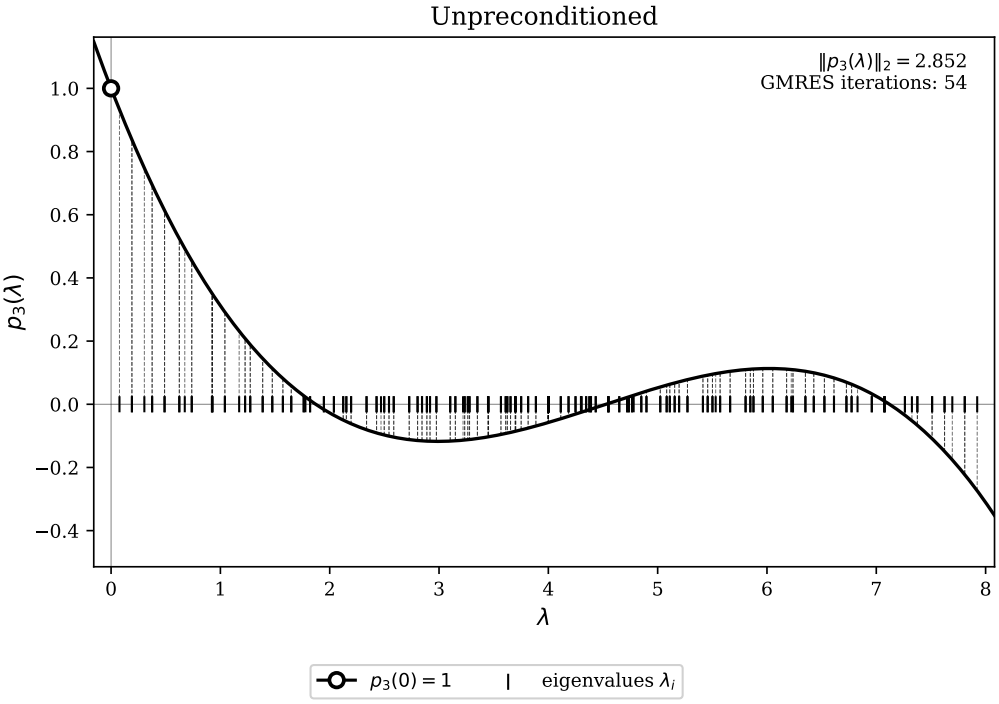
Applying this property yields the final representation of the residual

$$r^{(k)} = \sum_{i=1}^{n} c_i p_k(\lambda_i) \phi_i. \tag{2.30}$$

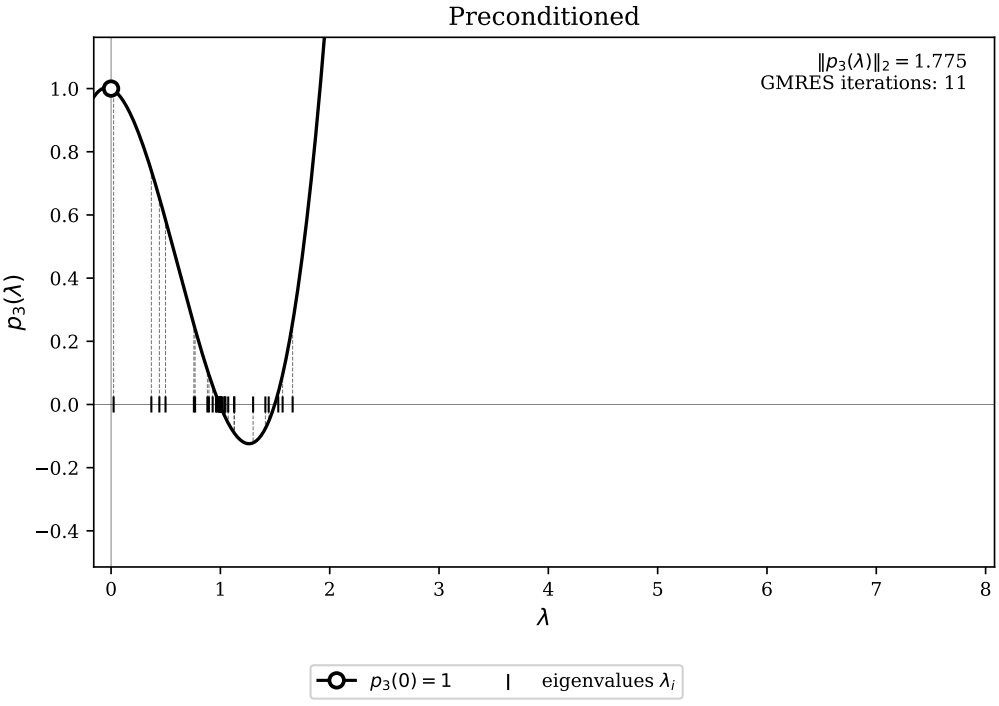
Minimizing the residual $r^{(k)}$ is thus equivalent to minimizing a scalar polynomial p_k at the eigenvalue locations λ_i, subject to the constraint $p_k(0) = 1$. For a symmetric positive definite A, the condition number $\kappa(A) = \lambda_{\max}/\lambda_{\min}$ measures the spread of the spectrum [7, §6.11], and a widely dispersed spectrum forces the polynomial to stay near zero over a larger interval. Achieving this requires a high-degree polynomial, which means more solver iterations.

To reduce this cost, we introduce a nonsingular preconditioner matrix M that approximates A, transforming $Au = f$ into an equivalent system with a more clustered eigenvalue spectrum [7, §9.1]. A polynomial of the same degree can then place its roots within these clusters and suppress p_k near every eigenvalue simultaneously.

This behavior is demonstrated in Figure 2.5, which plots the optimal residual polynomial of degree $k = 3$ for a 2D Poisson problem discretized on a uniform grid. The two-dimensional problem is chosen here for visual illustration, as the same qualitative behavior holds in one dimension.



(a) Unpreconditioned system. Dispersed eigenvalues force p_3 to remain large across the spectrum, giving a high residual norm $\|p_3(\lambda)\|_2$.



(b) Preconditioned system. Clustered eigenvalues allow p_3 to place its roots within each group, achieving a much lower residual norm.

Figure 2.5: Optimal residual polynomial of degree three for a 2D Poisson system, with the polynomial 2-norm $\|p_3(\lambda)\|_2$ and GMRES iteration count annotated. See Appendix B.2 for numerical parameters.

The left panel shows that dispersed eigenvalues force the third-degree polynomial to produce a larger residual error. Conversely, the preconditioned system in the right panel groups the eigenvalues into clusters, allowing the same polynomial to align its roots with these groups and achieve a much lower error, so fewer iterations are needed to reach the same convergence tolerance. An effective preconditioner must therefore group the spectrum tightly enough to minimize the required polynomial degree while remaining inexpensive to compute.

2.3.2 Applying preconditioners

Once a preconditioner M has been chosen, it must be incorporated into the iterative solver. The placement of M^{-1} relative to A is more than an algebraic choice, as it dictates which residual norm the method minimizes and defines what the stopping criterion actually measures [7, §9.3.4]. The primary approaches are left and right preconditioning, with a third form, split preconditioning, arising for the special case of symmetric systems.

As established previously, we want to transform the original problem using M. The most straightforward approach multiplies the system $Au = f$ by M^{-1} from the left, yielding $M^{-1}Au = M^{-1}f$ [7, Eq. 9.1]. This approach is known as *left preconditioning*. Multiplying through by M^{-1} replaces the operator A with $M^{-1}A$, creating a new system with a spectrum clustered near unity when M approximates A well. In a left preconditioned Krylov method, the search space becomes

$$\mathcal{K}_k(M^{-1}A,\, M^{-1}r^{(0)}),$$

and the method minimizes the norm of the preconditioned residual $\|M^{-1}r^{(k)}\|_2$. Because this equals the true residual $\|r^{(k)}\|_2$ only when $M = I$, the convergence criterion reported by the solver can misrepresent the actual solution quality [7, §9.3.4].

Right preconditioning avoids this discrepancy by introducing the change of variables $\hat{u} = Mu$ and rewriting the original system as [7, Eq. 9.2]

$$AM^{-1}\hat{u} = f.$$

The solver recovers the final solution afterwards using $u = M^{-1}\hat{u}$. The Krylov subspace is now built from the unpreconditioned residual

$$\mathcal{K}_k(AM^{-1},\, r^{(0)}),$$

and GMRES minimizes the true residual $\|r^{(k)}\|_2$. Consequently, right preconditioning is the more reliable choice when accurate residual monitoring matters.

Neither left nor right preconditioning preserves the symmetry of the preconditioned operator, even when both A and M are symmetric, because matrix products generally do not commute. The conjugate gradient method requires the system matrix to be SPD, but neither $M^{-1}A$ nor AM^{-1} is symmetric in general, making CG incompatible with either form. *Split preconditioning* recovers symmetry by factoring $M = LL^T$, where L is lower triangular, and forming the equivalent system [7, Eq. 9.4]

$$L^{-1}AL^{-T}\hat{u} = L^{-1}f,$$

where $\hat{u} = L^T u$ is recovered afterwards as $u = L^{-T}\hat{u}$. The preconditioned operator $L^{-1}AL^{-T}$ is SPD whenever A is SPD and $M = LL^T$, so CG applies directly.

In practice, the *preconditioned conjugate gradient* (PCG) algorithm never forms this split [7, Algorithm 9.1]. At each iteration it computes $z^{(k)} = M^{-1}r^{(k)}$ and replaces the two scalar inner products

per step with M-weighted ones, using $(r^{(k)})^T z^{(k)}$ in place of $(r^{(k)})^T r^{(k)}$. This is mathematically equivalent to running CG on $L^{-1}AL^{-T}$, but requires only that M be SPD and that M^{-1} can be applied as an operator to a vector, without ever factoring M explicitly. Provided the preconditioner is SPD, PCG applies it directly at each iteration, which is why this requirement is central to the remainder of the thesis.

2.3.3 Defining preconditioners

Section 2.3.1 established that an effective preconditioner must cluster the eigenvalues of the preconditioned operator while remaining inexpensive to construct and apply. This section explores the main strategies for achieving that balance, beginning with the simplest choices and progressively addressing their limitations.

The matrix splitting $A = M - N$ introduced in Section 2.2.1 reveals a deeper connection between stationary iterations and preconditioning. At convergence, iteration (2.2) satisfies $u = M^{-1}Nu + M^{-1}f$. Substituting $N = M - A$ gives

$$M^{-1}Au = M^{-1}f,$$

which is precisely the left-preconditioned system. Every stationary iteration is therefore a converged staionary method on a preconditioned system, and its splitting matrix M is the preconditioner. For Jacobi, $M = D$, and for Gauss-Seidel, $M = D - E$. These are inexpensive to apply, but the diagonal and triangular approximations neglect too much of the coupling in A to cluster eigenvalues effectively for strongly coupled systems.

Retaining more of this off-diagonal coupling requires a closer approximation of A. *Incomplete LU factorization* (ILU) provides this by computing an approximate factorization $A \approx \tilde{L}\tilde{U}$, where $\tilde{L}$ is lower triangular and $\tilde{U}$ is upper triangular, while discarding fill-in outside a chosen sparsity pattern, keeping the factors sparse. The ILU(p) family controls the fill-in budget by allowing up to p additional levels beyond the sparsity pattern of A, trading memory and setup cost for improved eigenvalue clustering [7, §10.3.3]. The simplest member, ILU(0), allows no fill-in, so $\tilde{L}$ and $\tilde{U}$ are constrained to the sparsity pattern of A [7, §10.3.2].

The preceding discussion expressed the preconditioned system in terms of M^{-1}. In practice, this inverse is never formed explicitly, since even when M is sparse, M^{-1} is in general dense, destroying the memory savings that motivate iterative methods. Applying the preconditioner to a vector r therefore means solving $Mw = r$ for w.

Using a preconditioner introduces a computational trade-off with two distinct cost components. The *build phase* constructs the preconditioner from A and is paid once before the solver starts. The *solve phase* covers the Krylov iteration and grows with the number of iterations required. A cheap preconditioner such as Jacobi has negligible build cost but clusters eigenvalues poorly, leaving the solve phase expensive. ILU(0) incurs a higher build cost yet reduces the iteration count enough that the solve phase compensates. Chapters 3 and 4 use ILU(0) as the working preconditioner and examine whether the build phase can be avoided after grid refinement by mapping an existing preconditioner to the new grid.

2.4 Adaptive mesh refinement

Many physical processes involve phenomena that span a wide range of length scales simultaneously. Certain features, such as shock waves, can be orders of magnitude smaller than the overall size of

the system, yet must be resolved accurately for the solution to be physically meaningful. Applying a uniformly fine grid across the entire simulation domain is computationally impractical, as the number of cells scales as N^d in d spatial dimensions for N cells per dimension.

Adaptive mesh refinement (AMR) addresses this by dynamically concentrating resolution only in regions where the solution demands it, leaving the grid coarse elsewhere [10, p. 484]. The resolution adapts as the solution evolves, so computational effort is always focused where it is physically needed. This section describes the block-structured variant of AMR, which organizes the domain into a hierarchy of fixed-size rectangular blocks. The following subsections describe the grid hierarchy, the criterion used to trigger refinement, and the operations that transfer field values between levels.

Block-structured grids

In the block-structured approach to AMR, the computational domain is organized as a hierarchy of refinement levels $\ell = 0, 1, \ldots, \ell_{\max}$. The base level $\ell = 0$ covers the entire domain with a coarse uniform grid of spacing h_0. Each successive level ℓ is obtained by subdividing cells of the previous level by a fixed *refinement ratio* r, giving a local grid spacing

$$h_\ell = \frac{h_0}{r^\ell}.$$

The most common choice is $r = 2$, which halves the spacing at each level. Refined regions at level ℓ are not individual cells but rectangular *blocks*, each containing a fixed number n_{cells} of cells per spatial dimension. When a region is refined, the coarser block covering that region remains in the hierarchy alongside the newly created fine-level blocks. The hierarchy is strictly nested in the sense that every fine block lies entirely within the spatial extent of exactly one coarser block.

Because all blocks contain the same number of cells, each block has a locally uniform structure and the computational work per block is predictable. This contrasts with approaches that refine individual cells, where the irregular grid structure complicates bookkeeping and the exchange of values between blocks.

Refinement criteria

The algorithm must decide where to introduce or remove refinement. The standard approach applies a *refinement criterion* to each block at the end of each time step. A widely used choice is a normalized second-derivative estimator that measures, for a selected scalar quantity such as density or pressure, how non-linearly that quantity varies across each cell relative to its surrounding gradient [11, §2]. The resulting dimensionless ratio is large near physically significant features such as shock waves and near zero where the solution is smooth, so refinement is concentrated where it is needed.

If the estimator exceeds an upper threshold, the block is flagged for *refinement* and replaced by r^d child blocks at the next finer level. Conversely, if the estimator falls below a lower threshold, the block is a candidate for *derefinement*, and the sibling blocks sharing the same parent are merged back into a single coarser block. The two thresholds are chosen so that the lower one is strictly below the upper, and the gap between them prevents a block from being repeatedly refined and derefined when the estimator is near a threshold.

Grid transfer

Whenever a block is refined or derefined, field values must be transferred between grid levels. When a block is refined, the solution on the parent block is interpolated onto the newly created child blocks. The simplest scheme assigns each child cell the value of the parent cell that contains it, while more accurate schemes use values from neighboring parent cells to construct a smoother approximation. When a block is derefined, the solution on the child blocks is averaged back onto the parent block. For conservative quantities this averaging is volume-weighted, ensuring that the total conserved quantity is preserved under the transfer.

The interpolation and averaging operations used here are developed into explicit matrix operators in Section 3.2.

From the perspective of the linear solver, however, a refinement event does more than redistribute solution values. It replaces the system matrix A entirely, since the new discretization lives on a different grid. Any preconditioner M that was built for the old system is no longer aligned with the new one, yet rebuilding it from scratch at every refinement step is expensive. The question of whether M can instead be updated geometrically, using only the grid transfer operators, is the central problem of Chapter 3, where it is first studied in a simplified one-dimensional setting.

3

Preconditioner reuse in a 1D model problem

With the theoretical tools established in Chapter 2, this chapter tests the geometric mapping approach in a controlled setting before the FLASH implementation in Chapter 4. The model problem is the one-dimensional steady-state diffusion equation, discretized by the finite volume method on a non-uniform grid. Working in one spatial dimension separates the mapping operator logic from the geometric complexity of higher-dimensional configurations and avoids the temporal coupling present in time-dependent systems.

Three stages are developed in sequence. The prolongation operator P and restriction operator R are derived from the grid geometry alone and validated against seven properties. The error analysis reveals that applying P injects high-frequency components into the residual that the mapped preconditioner M_{map}^{-1} cannot resolve on its own. Adding a smoother before and after the mapped correction forms the composite preconditioner M_{comp}^{-1}, which suppresses these components. Numerical experiments on 135 problem configurations confirm that the composite achieves iteration counts that are independent of grid size across three orders of magnitude in problem size and across several smoother and solver variants.

Section 3.1 introduces the model problem and validates the discrete system. Section 3.2 derives and validates the mapping operators and analyzes the errors they introduce. Section 3.3 develops the composite preconditioner. Section 3.4 reports the numerical experiments.

3.1 1D steady-state diffusion equation as a model problem

Before integrating new numerical methods in a complex, large-scale codebase like FLASH, it is crucial to validate the algorithmic logic in an isolated environment. This separates fundamental algorithmic flaws from implementation errors. For this purpose, we use the one-dimensional steady-state diffusion equation as our model problem. Restricting the domain to one dimension isolates the derivation of mapping operators from the geometric complexity of higher-dimensional grids. Because grid mapping is a purely spatial operation, a time-independent equation eliminates temporal discretization as a potential source of error. By decoupling these spatial operators from temporal and multidimensional dynamics, any subsequent deviations in convergence behavior can be attributed to the new mapping scheme. The *finite volume method* (FVM), discretized in Section 3.1.1, is used in preference to finite differences because it mirrors the discretization strategy employed by FLASH, making the mapping operators developed here applicable to FLASH after the generalizations described in Chapter 4. With $u(x)$ denoting the quantity of interest (e.g. temperature or density), $D(x) > 0$ the spatially varying

diffusion coefficient, and $f(x)$ a source term, the equation reads

$$-\frac{d}{dx}\left(D(x)\frac{d}{dx}u(x)\right) = f(x), \quad x \in [0,1]. \tag{3.1}$$

Throughout the numerical experiments of this chapter, nine physics configurations are considered, combining three solution profiles with three diffusion coefficient profiles. Each profile is defined on the unit domain $[0,1]$ with homogeneous Dirichlet boundary conditions $u(0) = u(1) = 0$. For the parabolic and sinusoidal profiles, a *manufactured solution* is used, where $u(x)$ is prescribed analytically and the source term derived from it as $f(x) = -(D(x)\,u'(x))'$, so that an exact analytical reference is available for error comparison.

- **Triangle:** The source term is a Dirac delta at the midpoint, $f(x) = \delta(x - \frac{1}{2})$, whose piecewise-linear solution follows by integration.
- **Parabolic:** The manufactured solution $u(x) = x(1-x)$, with $f(x) = -(D(x)\,u'(x))'$.
- **Sinusoidal:** The manufactured solution $u(x) = \sin(2\pi x)$, with $f(x) = -(D(x)\,u'(x))'$.

The diffusion coefficient $D(x)$ takes one of three forms.

- **Constant:** $D(x) = 1$.
- **Linear:** $D(x) = x + 1$.
- **Smooth step:** $D(x) = 3 + 2\tanh\big(50\,(x - \frac{1}{2})\big)$, a near-discontinuous transition from $D \approx 1$ to $D \approx 5$ centered at $x = \frac{1}{2}$.

The parameter sweeps in Section 3.4 use all nine physics configurations, covering a range of physics settings to test the robustness of the conclusions. The following subsections derive the FVM discretization of equation (3.1), verify the numerical properties of the resulting linear system, and describe the grid adaptation scheme used in the experiments.

3.1.1 Finite volume discretization

As illustrated in Figure 3.1, the domain is divided into N control volumes with faces at integer indices x_i and cell centers at fractional indices $x_{i+1/2}$. Two distinct length scales characterize the grid. The cell width $h_{i+1/2} \coloneqq x_{i+1} - x_i$ measures the span of a control volume, while the distance $h_i \coloneqq x_{i+1/2} - x_{i-1/2}$ measures the spacing between adjacent centers, and the two are equal only on a uniform grid. The cell state $u_{i+1/2}$ is the volume average of the continuous solution over its control volume, $u_{i+1/2} \coloneqq \frac{1}{h_{i+1/2}} \int_{x_i}^{x_{i+1}} u(x)\,dx$, rather than its pointwise value at the cell center.

Integrating (3.1) over each control volume and approximating the face fluxes by central differences (see Appendix A.1 for the full derivation) yields the stencil coefficients

$$\alpha_i \coloneqq -\frac{D_i}{h_i}, \qquad \beta_i \coloneqq \frac{D_i}{h_i} + \frac{D_{i+1}}{h_{i+1}}, \qquad \gamma_i \coloneqq -\frac{D_{i+1}}{h_{i+1}}, \tag{3.2}$$

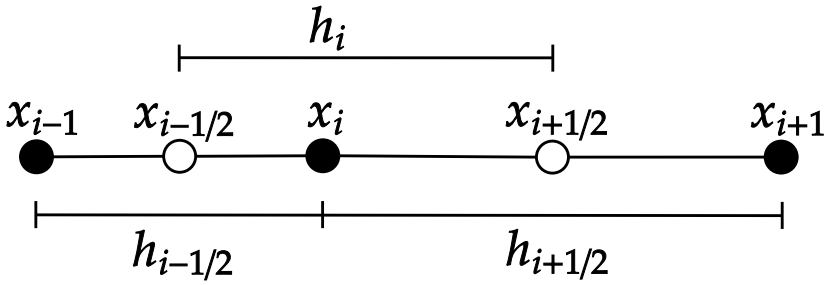


Figure 3.1: Schematic of the local non-uniform finite volume grid, defining the control volume faces at integer indices x_i, cell centers at fractional indices $x_{i+1/2}$, and the respective distances $h_{i+1/2}$ and h_i.

where $D_i = D(x_i)$ is the diffusion coefficient evaluated at face x_i. Incorporating Dirichlet boundary conditions $u(x_0) = f_L$ and $u(x_N) = f_R$ through a half-cell adjustment at both ends of the domain assembles the N interior unknowns into the tridiagonal linear system $Au = f$ with

$$\begin{bmatrix} \beta_0 & \gamma_0 & & & & & & \\ \alpha_1 & \beta_1 & \gamma_1 & & & & & \\ & & \ddots & & & & & \\ & & \alpha_{i-1} & \beta_{i-1} & \gamma_{i-1} & & & \\ & & & \alpha_i & \beta_i & \gamma_i & & \\ & & & & & \ddots & & \\ & & & & & \alpha_{N-2} & \beta_{N-2} & \gamma_{N-2} \\ & & & & & & \alpha_{N-1} & \beta_{N-1} \end{bmatrix} \begin{bmatrix} u_{1/2} \\ u_{3/2} \\ \vdots \\ u_{i-1/2} \\ u_{i+1/2} \\ \vdots \\ u_{N-3/2} \\ u_{N-1/2} \end{bmatrix} = \begin{bmatrix} f_{1/2}h_{1/2} - f_L\alpha_0 \\ f_{3/2}h_{3/2} \\ \vdots \\ f_{i-1/2}h_{i-1/2} \\ f_{i+1/2}h_{i+1/2} \\ \vdots \\ f_{N-3/2}h_{N-3/2} \\ f_{N-1/2}h_{N-1/2} - f_R\gamma_{N-1} \end{bmatrix}. \tag{3.3}$$

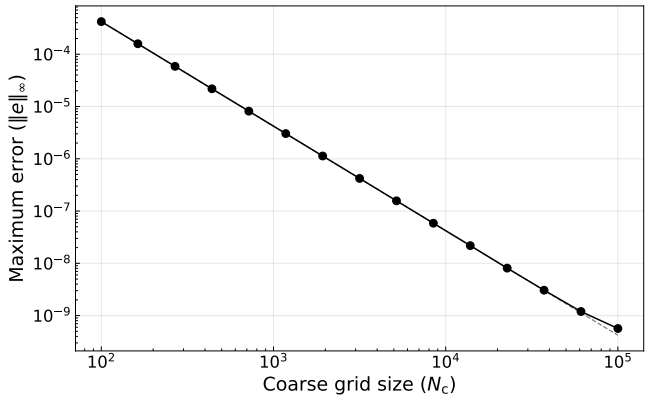
Because $\alpha_{i+1} = \gamma_i$ for all interior rows, A is symmetric. Since $D(x) > 0$ throughout the domain, A is also symmetric positive definite (SPD) [12, p. 36].

3.1.2 Validation of the discrete system

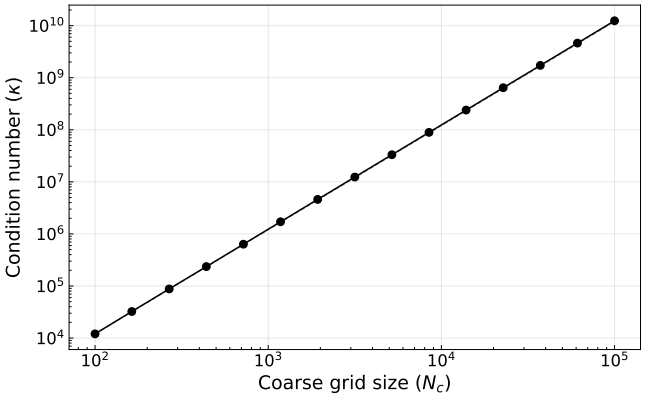
Before using the FVM system as the basis for preconditioner experiments, its numerical properties are verified. The test uses a sinusoidal solution with a linear diffusion coefficient solved on a non-uniform grid with N_c base cells of spacing $h_0 = 1/N_c$. One level of dyadic refinement ($r = 2$, $\ell = 1$) is applied to cells whose centers fall within the central 20% of the domain, giving cell widths $h_1 = h_0/2$ there and h_0 elsewhere. This refinement structure is held fixed as N_c increases, so all cell widths shrink proportionally and the error does not stagnate at a fixed-resolution floor. The solution error is measured as $\|u(x) - u\|_\infty$, where $u(x)$ is the exact solution evaluated at cell centers and u is the FVM solution vector.

Despite the flux approximation being first-order accurate on non-uniform grids, Figure 3.2a shows the global solution error $\|u(x) - u\|_\infty$ converging at $\mathcal{O}(N_c^{-2})$, consistent with $\mathcal{O}(h_0^2)$. This *supraconvergence*, where a first-order consistent scheme nevertheless achieves second-order global accuracy [16, p. 540], is a general property of this discretization on non-uniform grids, independent of where the refinement region is placed (see Appendix A.1, §A.1.1).

Figure 3.2b shows $\kappa(A)$ growing as $\mathcal{O}(N_c^2)$ under grid refinement. Halving the grid spacing roughly quadruples $\kappa(A)$. Without an effective preconditioner, the solver degrades with every refinement step, motivating the rest of this chapter.



(a) Maximum solution error $\|u(x) - u\|_\infty$, converging as $\mathcal{O}(N_c^{-2})$.



(b) Condition number $\kappa(A)$, growing as $\mathcal{O}(N_c^2)$.

Figure 3.2: Solution error and condition number as functions of coarse grid size N_c on a log–log scale, for a sinusoidal solution with linear diffusion coefficient on a non-uniform grid with the central 20% of the domain bisected.

3.1.3 Grid adaptation

The grid adaptation used here is not dynamic. No gradient-based criterion selects which cells to refine or derefine. Instead, a target region specified by a center and a radius is provided, and cells are marked for refinement or derefinement based on their overlap with this region. Because the mapping operators are purely geometric, verifying their implementation only requires mimicking the structural changes to the grid, isolating the linear algebra from any AMR decision logic.

The adaptation scheme mirrors some of the constraints of FLASH's AMR implementation, which is built on the PARAMESH library [13, §II]. Each grid change is applied one level at a time. When refining, a marked cell is split into two equal children. When derefining, two equal children are merged into a single coarse parent. To illustrate this, Figure 3.3 shows a locally adapted 1D grid generated in two discrete steps from an initially uniform base grid, first refining one targeted region and subsequently derefining another.

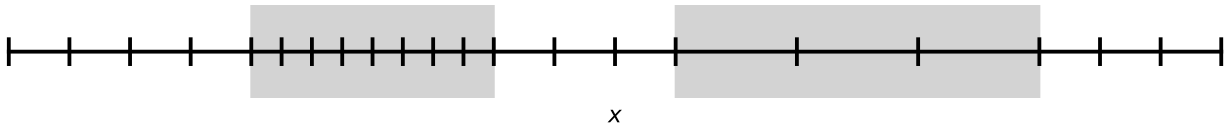


Figure 3.3: A 1D finite volume grid after two steps of adaptation. The gray shaded regions indicate the targeted areas. The left region was refined by bisecting the original cells, while the right region was derefined by merging adjacent pairs of cells.

3.2 Mapping operators

To avoid the computationally expensive recalculation of the preconditioner M following a grid adaptation, the existing preconditioner must be transformed to fit the new grid geometry. This is accomplished using geometric mapping operators. Let $\mathbb{R}^{N_c}$ and $\mathbb{R}^{N_f}$ denote the vector spaces associated

with the coarse and fine grids, respectively, where N_c and N_f are the number of cells at each resolution. A grid state at either resolution is a vector in the corresponding space, with one component per cell. As illustrated schematically in Figure 3.4, the prolongation operator $P : \mathbb{R}^{N_c} \to \mathbb{R}^{N_f}$ maps a state vector by splitting cells, while the restriction operator $R : \mathbb{R}^{N_f} \to \mathbb{R}^{N_c}$ maps a state vector by merging cells.

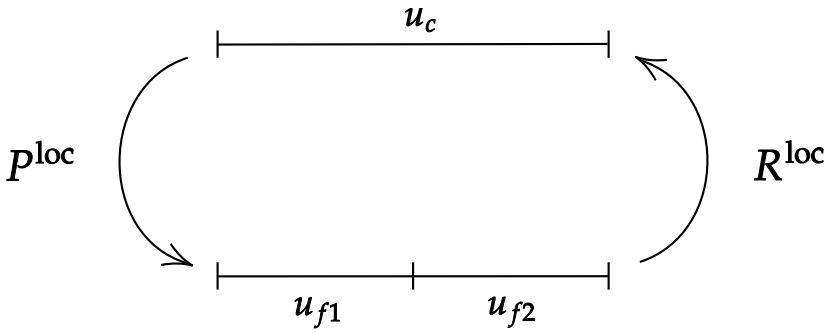


Figure 3.4: Schematic of the local prolongation and restriction operations. Prolongation P^{loc} splits a coarse cell into two fine cells, while restriction R^{loc} merges two fine cells into a coarse cell.

Using these operators, the mapped preconditioner takes one of two forms depending on the direction of grid adaptation.

$$M_{\text{map}}^{-1} := \begin{cases} PM_{\text{old}}^{-1}R & \text{for refinement,} \\ RM_{\text{old}}^{-1}P & \text{for derefinement.} \end{cases} \tag{3.4}$$

As established in Section 2.3.1, the convergence speed of Krylov subspace solvers depends on eigenvalue clustering. Consequently, the mapping must transfer the preconditioner to the new grid without destroying the eigenvalue clustering achieved by M_{old}. By ensuring the mapping operators are transposes of each other, $R = P^T$, the mapped preconditioner takes the form $M_{\text{map}}^{-1} = PM_{\text{old}}^{-1}P^T$, which is called a *Galerkin projection*. For symmetric systems, this guarantees that the eigenvalues of the mapped preconditioner remain bounded by the spectral extremes of the original preconditioner, as shown in Section 3.2.1.

Conversely, if this transpose relationship is violated, the mismatch $R^T - P$ shifts the eigenvalues [14, Thm. 6.3.2], destroying the valuable clustering achieved by the original preconditioner. Because Krylov methods require more iterations to converge when eigenvalues are scattered, this perturbation degrades solver performance. Therefore, to maintain tightly clustered eigenvalues and overall solver efficiency, the mapping must enforce the transpose condition $R = P^T$.

The following subsections derive the specific forms of P and R under the transpose condition $R = P^T$, validate these operators against a suite of algebraic and physical conservation properties, and analyze the truncation error introduced by the prolongation step.

3.2.1 Derivation of mapping operators

Before deriving the specific mathematical form of P and R, it is necessary to distinguish between intensive and extensive variables. *Intensive variables* are independent of the control volume size, such as temperature. In contrast, *extensive variables* scale proportionally with the control volume, such as mass. The mathematical treatment of these two variable types diverges when mapping state vectors between the two spaces. As will emerge from the derivation, this physical distinction

naturally enforces the transpose relationship $R = P^T$ that Section 3.2 requires for the preconditioner mapping to remain a Galerkin projection.

One geometric assumption underlying the derivation requires clarification. The FVM derivation in Section 3.1.1 uses a non-uniform grid because AMR creates varying cell sizes across the global domain. However, as established in Section 3.1.3, the actual refinement step simply bisects a cell into two equal halves. Therefore, the derivations below rely purely on this locally uniform split, even though the global grid is non-uniform.

Consider a single coarse cell with a known value u_c. When we refine the grid by bisection, we slice this cell into two equal fine cells, u_f1 and u_f2, as illustrated in Figure 3.4. How we assign values to these new cells depends on the physical nature of the variable. If the variable is intensive, like temperature, slicing the volume does not change the physical state. The temperature in each new half remains identical to the original whole. Mathematically, $u_\mathrm{f1} = u_\mathrm{c}$ and $u_\mathrm{f2} = u_\mathrm{c}$, so the local intensive prolongation operator takes the form

$$P_\mathrm{int}^\mathrm{loc} \coloneqq \begin{bmatrix} 1 \\ 1 \end{bmatrix}. \tag{3.5}$$

If the variable is extensive, like mass, the total quantity must be conserved and distributed according to the new volumes. Since the coarse cell is halved, each fine cell receives exactly half of the parent mass. Setting $u_\mathrm{f1} = 0.5\,u_\mathrm{c}$ and $u_\mathrm{f2} = 0.5\,u_\mathrm{c}$ gives the local extensive prolongation operator

$$P_\mathrm{ext}^\mathrm{loc} \coloneqq \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}. \tag{3.6}$$

Conversely, restriction takes the values of two fine cells, u_f1 and u_f2, and merges them to define the single coarse cell u_c. For intensive variables, combining two volumes with different temperatures gives the volume-weighted average. Since the fine cells possess equal volume, this is a simple arithmetic mean where $u_\mathrm{c} = 0.5\,u_\mathrm{f1} + 0.5\,u_\mathrm{f2}$, giving the local intensive restriction operator

$$R_\mathrm{int}^\mathrm{loc} \coloneqq \begin{bmatrix} 0.5 & 0.5 \end{bmatrix}. \tag{3.7}$$

For extensive variables, merging two masses means the total mass in the coarse cell must exactly equal the sum of the masses in its constituent fine cells. This local summation, $u_\mathrm{c} = u_\mathrm{f1} + u_\mathrm{f2}$, gives the local extensive restriction operator

$$R_\mathrm{ext}^\mathrm{loc} \coloneqq \begin{bmatrix} 1 & 1 \end{bmatrix}. \tag{3.8}$$

To generalize to an arbitrary number of cells, consider a global coarse grid consisting of N_c cells. The global prolongation operator P_int is constructed as an $N_\mathrm{f} \times N_\mathrm{c}$ block-diagonal matrix, where N_f is the total number of cells in the resulting fine grid. The j-th diagonal block B_j is determined by the local refinement status of the k-th coarse cell,

$$B_j = \begin{cases} 1 & \text{cell } j \text{ unchanged,} \\ P^\mathrm{loc} & \text{cell } j \text{ refined,} \\ R^\mathrm{loc} & \text{cell } j \text{ derefined.} \end{cases} \tag{3.9}$$

For example, if a domain begins with three coarse cells ($N_\mathrm{c} = 3$) and only the central cell is bisected, the resulting fine grid contains four cells ($N_\mathrm{f} = 4$). Assembling the blocks $B_1 = 1$, $B_2 = P_\mathrm{int}^\mathrm{loc}$, and

$B_3 = 1$ produces the global intensive prolongation operator

$$P_{\text{int}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & P_{\text{int}}^{\text{loc}} & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{3.10}$$

The global restriction operator R_{int} performs the reverse operation, mapping a state vector from $\mathbb{R}^{N_{\text{f}}}$ back to $\mathbb{R}^{N_{\text{c}}}$ using $R_{\text{int}}^{\text{loc}}$ for the merged cells,

$$R_{\text{int}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & R_{\text{int}}^{\text{loc}} & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0.5 & 0.5 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{3.11}$$

Comparing the four local operators, the extensive variants are exact transposes of their intensive counterparts, $P_{\text{ext}}^{\text{loc}} = (R_{\text{int}}^{\text{loc}})^T$ and $R_{\text{ext}}^{\text{loc}} = (P_{\text{int}}^{\text{loc}})^T$. Since the global matrices are block-diagonal assemblies of these local blocks, the relationship carries to the global level, $P_{\text{ext}} = R_{\text{int}}^T$ and $R_{\text{ext}} = P_{\text{int}}^T$. This transpose structure arises from the physics of the variable types rather than being imposed externally. The transpose condition $R = P^T$ required by the preconditioner mapping (Section 3.2) is therefore satisfied only by cross-type pairs, (P_{int}, R_{ext}) or (P_{ext}, R_{int}), but not by same-type pairs.

The FVM discretization determines which of these two pairs applies in each direction. In the preconditioned iteration, the mapped preconditioner acts on the residual $r = f - Au$, so the residual inherits the physical type of the right-hand side f. From (3.3), each entry of f carries an explicit factor of the cell width $h_{i+1/2}$, making f and therefore r extensive quantities. Since the solution u is a volume-averaged, intensive field, A maps an intensive input to an extensive output. Consequently, the approximate inverse operator M_{old}^{-1} acts in the opposite direction, transforming an extensive residual into an intensive correction. For refinement, the mapped preconditioner is applied as $PM_{\text{old}}^{-1}Rr$. The operator first applied to the extensive residual must handle extensive variables, fixing $R = R_{\text{ext}}$, while the final operator must handle intensive variables, fixing $P = P_{\text{int}}$. For derefinement, where the application is $RM_{\text{old}}^{-1}Pr$, the same reasoning fixes $P = P_{\text{ext}}$ and $R = R_{\text{int}}$. That the dimensional argument selects exactly the pairs the transpose condition permits reflects the physical consistency of the FVM discretization.

3.2.2 Validation of mapping operators

To ensure the mapping operators behave as intended before applying them to the preconditioner, a suite of numerical tests was developed. These tests evaluate whether the operators satisfy fundamental matrix algebra rules and conserve physical quantities. Throughout this validation framework, the infinity norm is used, since it evaluates the maximum absolute pointwise error and guarantees that no single element violates the constraints. The framework verifies the following key properties.

- **Identity mapping:** When the initial and final grids are identical, the mapping operators must leave the state vector unchanged, collapsing into the identity matrix I. Using P and R as shorthand notation representing all intensive and extensive variants, this property requires that the error norms $\|P - I\|_\infty$ and $\|R - I\|_\infty$ are zero.
- **Sparsity preservation:** The number of nonzero elements (nnz) in any global mapping matrix must equal the total number of fine cells, N_{f}. This constraint thus requires $\text{nnz}(P) = \text{nnz}(R) = N_{\text{f}}$.

- **Transpose symmetry:** The extensive operators must act as the matrix transposes of their intensive counterparts. This relationship dictates that the matrix differences $\|P_{\text{ext}} - R_{\text{int}}^T\|_\infty$ and $\|R_{\text{ext}} - P_{\text{int}}^T\|_\infty$ evaluate to zero.
- **Projection identity:** Mapping a state vector from $\mathbb{R}^{N_\text{c}}$ to $\mathbb{R}^{N_\text{f}}$ and back must recover the original vector in $\mathbb{R}^{N_\text{c}}$. This guarantees that this sequence does not accumulate error, requiring $\|I_\text{c} - R_{\text{int}}P_{\text{int}}\|_\infty = 0$ and $\|I_\text{c} - R_{\text{ext}}P_{\text{ext}}\|_\infty = 0$. The reverse is not true, which will be explored in Section 3.2.3.
- **Extensive conservation:** Both extensive operators must be conservative across the entire domain. When mapping an arbitrary extensive vector u_{ext} between the coarse grid (N_c cells) and the fine grid (N_f cells) in either direction, the global sum must be preserved.

$$\sum_{j=1}^{N_\text{f}}(P_{\text{ext}}u_\text{c})_j = \sum_{i=1}^{N_\text{c}}(u_\text{c})_i \quad \text{and} \quad \sum_{i=1}^{N_\text{c}}(R_{\text{ext}}u_\text{f})_i = \sum_{j=1}^{N_\text{f}}(u_\text{f})_j.$$

- **Intensive equilibrium:** The intensive prolongation operator must duplicate coarse parent values into their respective fine child cells. If $u_\text{f} = P_{\text{int}}u_\text{c}$, then for any fine child cell j nested within a coarse parent cell i, the discrete values must match, requiring $(u_\text{f})_j = (u_\text{c})_i$.
- **Integral conservation:** The intensive restriction operator must reconstruct spatial averages. Applying R_{int} to fine-grid states u_f obtained by integrating any continuous non-linear function over the fine cells must exactly recover the exact coarse-grid states u_c, requiring $\|R_{\text{int}}u_\text{f} - u_\text{c}\|_\infty = 0$.

When subjected to this testing suite, the global mapping operators derived in the previous section satisfied all these properties. In every test where mathematical equality to zero was expected, the measured error norms of the implementation were below 10^{-15}, confirming that the operators are limited only by double-precision floating-point arithmetic.

3.2.3 Error analysis

While the numerical validation in the previous section confirms the algebraic correctness and physical consistency of the mapping operators, it is equally important to evaluate the error introduced during the grid transfer process. The discretization accuracy of the solution is governed by the finite volume matrix A and the right-hand side f. Because the mapped preconditioner only transforms the search space of the iterative solver without altering this solution, any mapping errors it contains cannot degrade the discretization accuracy. However, evaluating this error remains critical because it can leave certain residual components beyond the reach of the mapped preconditioner.

As confirmed numerically by the projection identity test in Section 3.2.2, both same-type pairs satisfy $RP = I_\text{c}$. The local product $R^{\text{loc}}P^{\text{loc}} = 1$ for both, and the global identity follows from the block-diagonal assembly. The reverse roundtrip, PR, does not recover the fine-grid identity I_f. Because $(PR)^2 = P(RP)R = PI_\text{c}R = PR$, the operator PR possesses the defining property of a projection matrix. It maps $\mathbb{R}^{N_\text{f}}$ to $\mathbb{R}^{N_\text{f}}$ but projects onto an N_c-dimensional subspace.

Because PR is a projection matrix, it decomposes the ambient space into a direct sum of two non-overlapping subspaces, $\mathbb{R}^{N_\text{f}} = \text{range}(PR) \oplus \text{null}(PR)$ [7, §1.12.1]. The first subspace, $\text{range}(PR) = \text{range}(P)$, comprises all vectors reachable from $\mathbb{R}^{N_\text{c}}$ through P. Since $RP = I_\text{c}$, any $x \in \text{range}(P)$ satisfies $PRx = x$, confirming it lies in $\text{range}(PR)$. The second subspace, $\text{null}(PR) = \text{null}(R)$,

consists of the vectors the projection annihilates and isolates the mapping errors. This again follows because $RP = I_c$ ensures that $PRx = 0$ implies $Rx = 0$.

To characterize these errors, we can exploit the operator's complement, $I_f - PR$. A fundamental property of projections dictates that the null space of the original matrix is the range of its complement, yielding $\text{null}(PR) = \text{range}(I_f - PR)$ [7, p. 34]. Consequently, rather than explicitly solving for the null space, we can characterize the annihilated vectors by determining the basis of this complementary range. Evaluating this complement for either local same-type pair yields the same error block

$$I_f^{loc} - P^{loc}R^{loc} = \begin{bmatrix} 0.5 & -0.5 \\ -0.5 & 0.5 \end{bmatrix}. \tag{3.12}$$

Globally, $I_f - PR$ is a block-diagonal matrix where unrefined cells map to a zero block and refined cells map to the local error block above. The column space of this local error matrix is spanned by vectors of the form $\begin{bmatrix} 1 & -1 \end{bmatrix}^T$, indicating that any fine-grid vector subject to annihilation must consist of equal and opposite adjacent values. Geometrically, a vector that alternates in sign between adjacent cells represents a high-frequency oscillation, demonstrating that the mapping errors inherently take this form.

To visualize this phenomenon, Figure 3.5 plots this mapping error, $(I_f - P_{ext}R_{ext})r$, for the residual r of a sinusoidal solution mapped onto a locally refined grid. While the envelope of the error is dictated by the specific physics of the underlying residual, the high-frequency oscillation itself is a direct result of the zeroth-order prolongation.

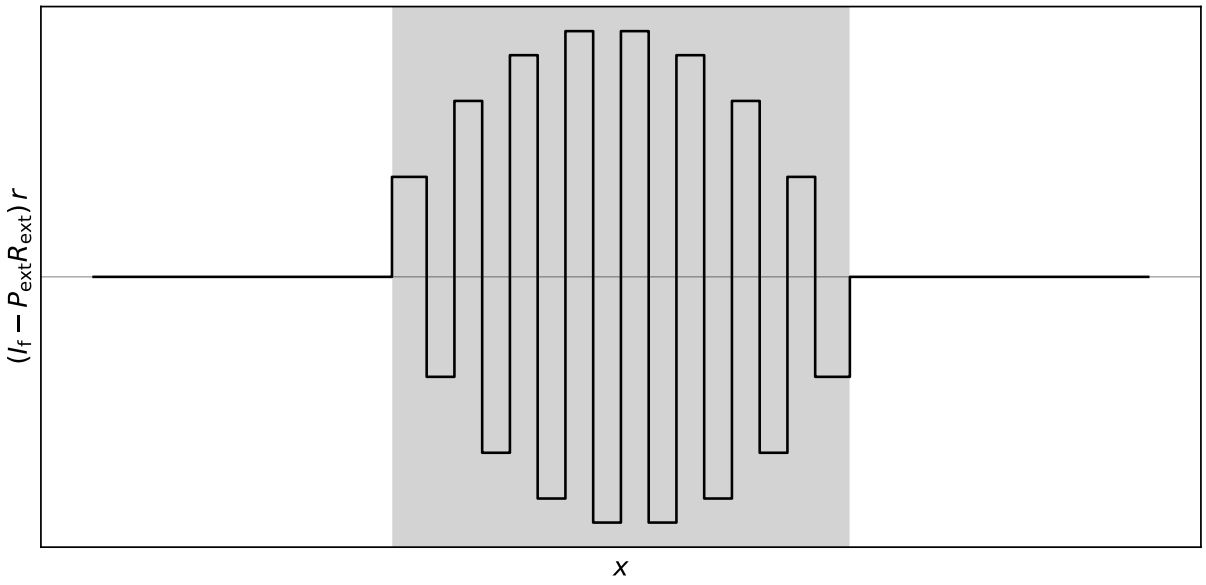


Figure 3.5: The orthogonal complement of the residual, $(I_f - P_{ext}R_{ext})r$, for a sinusoidal solution to the 1D diffusion problem. The gray shaded region indicates where the coarse grid was refined. The resulting error is a high-frequency oscillation, which resides in $\text{null}(R_{ext})$ and renders it invisible to the mapped preconditioner.

This exposes a failure mechanism of the mapped preconditioner after refinement, $M_{map}^{-1} = P_{int}M_{old}^{-1}R_{ext}$, during the construction of the Krylov subspace. When a preconditioned iterative solver generates its search directions, every basis vector is produced by a final application of P_{int}. To quantify the local truncation error injected by P_{int}, consider a continuous physical state $u(x)$ and a coarse cell centered

at $x_{i+1/2}$ with width $h_{i+1/2}$. Upon refinement, the coarse cell splits into two fine cells centered at $x_{i+1/4} = x_{i+1/2} - h_{i+1/2}/4$ and $x_{i+3/4} = x_{i+1/2} + h_{i+1/2}/4$, as shown in Figure 3.6.

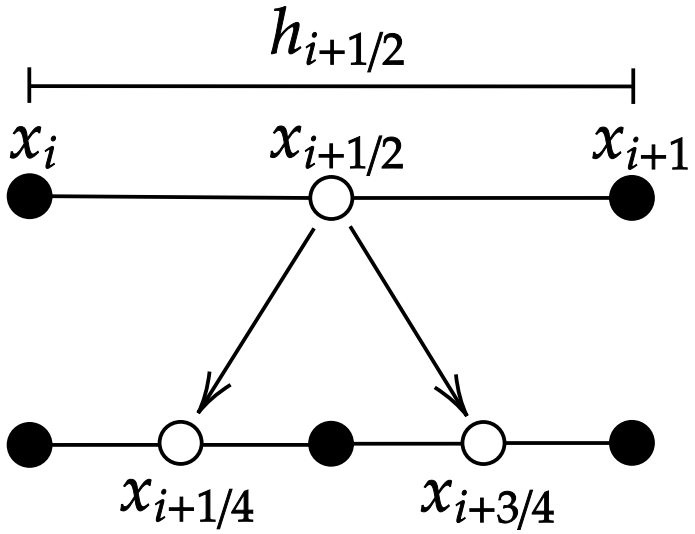


Figure 3.6: Schematic of the refinement process, defining the cell centers after refinement at fractional indices $x_{i+1/4}$ and $x_{i+3/4}$.

A Taylor expansion of the continuous solution about $x_{i+1/2}$, evaluated at each fine-cell center, yields

$$u(x_{i+1/2} \pm h_{i+1/2}/4) = u(x_{i+1/2}) \pm \frac{h_{i+1/2}}{4} u'(x_{i+1/2}) + \mathcal{O}(h_{i+1/2}^2). \tag{3.13}$$

Because the zeroth-order prolongation operator assigns the constant coarse-cell value to both fine cells, the localized truncation errors are

$$e_{i+1/4} = -\frac{h_{i+1/2}}{4} u'(x_{i+1/2}) + \mathcal{O}(h_{i+1/2}^2), \tag{3.14}$$

$$e_{i+3/4} = +\frac{h_{i+1/2}}{4} u'(x_{i+1/2}) + \mathcal{O}(h_{i+1/2}^2). \tag{3.15}$$

These error terms are equal and opposite in sign, proving that attempting to represent a gradient ($u' \neq 0$) within range(P_{int}) inherently injects high-frequency errors residing in null(R_{ext}). For any residual r, the projection $r_{\text{high}} = (I_{\text{f}} - P_{\text{ext}} R_{\text{ext}})r$ isolates this high-frequency component, and applying the mapped preconditioner gives

$$\begin{aligned} M_{\text{map}}^{-1} r_{\text{high}} &= P_{\text{int}} M_{\text{old}}^{-1} R_{\text{ext}} (I_{\text{f}} - P_{\text{ext}} R_{\text{ext}}) r \\ &= P_{\text{int}} M_{\text{old}}^{-1} (R_{\text{ext}} - R_{\text{ext}} P_{\text{ext}} R_{\text{ext}}) r \\ &= P_{\text{int}} M_{\text{old}}^{-1} (R_{\text{ext}} - I_{\text{c}} R_{\text{ext}}) r = 0, \end{aligned} \tag{3.16}$$

where the last step uses the projection identity $R_{\text{ext}} P_{\text{ext}} = I_{\text{c}}$. The zeroth-order prolongation operators derived in Section 3.2.1 are the simplest possible choice, selected here to test the feasibility of the preconditioner mapping concept with minimal complexity. Higher-order schemes could reduce the mapping error at the cost of a more involved derivation and implementation, but exploring that direction is only worthwhile if the concept proves effective at zeroth order. By injecting high-frequency oscillations during prolongation and annihilating them during restriction, the mapping operators leave these error modes uncorrected, necessitating the complementary fine-grid mechanism developed in Section 3.3.

3.3 Smoother

As shown in the error analysis of the mapping operators (Section 3.2.3), the zeroth-order prolongation operator P inherently injects high-frequency errors that the mapped preconditioner M_{map}^{-1} (3.4) cannot resolve, causing the Krylov solver to stall when used in isolation, as confirmed empirically in Section 3.4.1. To prevent this, we introduce smoothing preconditioners designed to target the remaining errors.

To combine the mapping operators with the smoothers, we employ a multiplicative two-grid preconditioner approach [7, §§13.4.2, 14.3.2]. An alternative method is the additive formulation, which applies all operators independently to the initial residual and sums their results. However, the multiplicative formulation is preferred here because it applies the operators sequentially, updating the residual between stages. This ensures that each subsequent operator only works on the errors left behind by the previous one, allowing for a clean separation of frequency targets.

Specifically, we construct a three-stage sequence consisting of pre-smoothing, coarse-grid correction (mapping), and post-smoothing. Given an initial residual r, the first stage applies one pre-smoother step to the auxiliary system $Az_{\text{pre}} = r$ from a zero initial guess, yielding the initial correction z_{pre},

$$z_{\text{pre}} \coloneqq M_{\text{pre}}^{-1} r. \tag{3.17}$$

Before the mapped preconditioner can be applied, the residual must be updated to reflect this partial correction. Multiplying the correction by the new-grid system matrix A maps it back to the residual space, giving the updated residual r_{pre},

$$r_{\text{pre}} \coloneqq r - Az_{\text{pre}} = (I - AM_{\text{pre}}^{-1})r. \tag{3.18}$$

The second stage obtains the coarse-grid correction z_{map} by applying the mapped preconditioner to this updated residual,

$$z_{\text{map}} \coloneqq M_{\text{map}}^{-1} r_{\text{pre}}. \tag{3.19}$$

The residual is updated once again to reflect the coarse-grid correction, giving r_{map},

$$r_{\text{map}} \coloneqq r_{\text{pre}} - Az_{\text{map}} = (I - AM_{\text{map}}^{-1})r_{\text{pre}}. \tag{3.20}$$

Finally, the post-smoother M_{post}^{-1} is applied to this updated residual, which contains any high-frequency noise left behind or injected by the mapping stage,

$$z_{\text{post}} \coloneqq M_{\text{post}}^{-1} r_{\text{map}}. \tag{3.21}$$

The final composite correction passed back to the Krylov solver is the sum of these three stages,

$$z_{\text{total}} \coloneqq z_{\text{pre}} + z_{\text{map}} + z_{\text{post}}. \tag{3.22}$$

By substituting the residual definitions into this sum and factoring out the initial residual r, the explicit structure of the composite preconditioner M_{comp}^{-1} is revealed,

$$M_{\text{comp}}^{-1} \coloneqq M_{\text{pre}}^{-1} + M_{\text{map}}^{-1}(I - AM_{\text{pre}}^{-1}) + M_{\text{post}}^{-1}(I - AM_{\text{map}}^{-1})(I - AM_{\text{pre}}^{-1}). \tag{3.23}$$

While this matrix summation appears complex, its underlying design principle is borrowed directly from classical multigrid theory, where operators are evaluated by their corresponding iteration matrices [9, p. 82].

Convergence and symmetry analysis

Recall that for any stationary iteration with splitting $A = M - N$, the error at each step is multiplied by the iteration matrix $G = I - M^{-1}A$ (Section 2.2.1). Substituting the composite preconditioner into this expression and factoring the right-hand side collapses the equation into the product of the three independent iteration matrices,

$$G_{\text{comp}} = I - M_{\text{comp}}^{-1}A = (I - M_{\text{post}}^{-1}A)(I - M_{\text{map}}^{-1}A)(I - M_{\text{pre}}^{-1}A) = G_{\text{post}}G_{\text{map}}G_{\text{pre}}. \tag{3.24}$$

Because the operators appear as factors in this product, suppression of a frequency by any single operator ensures its attenuation in the composite system. In classical multigrid theory, this factorization underlies the convergence argument. When G_{map} eliminates low-frequency errors and G_{pre} and G_{post} eliminate high-frequency errors, their product drives the total error to zero [9, pp. 83–84]. Since $M_{\text{comp}}^{-1}A = I - G_{\text{comp}}$, an eigenvalue λ of G_{comp} gives a corresponding eigenvalue $1 - \lambda$ of the preconditioned matrix. When the operators together suppress all frequency components, the eigenvalues of G_{comp} lie near zero, so the spectrum of $M_{\text{comp}}^{-1}A$ clusters near 1.

Furthermore, splitting the smoothing process into pre- and post-stages provides an additional benefit beyond damping high-frequency errors, namely symmetry of the composite preconditioner. While a single post-smoother is sufficient to damp high-frequency errors, a non-symmetric preconditioner restricts the choice of Krylov solvers.

If A and M_{old}^{-1} are both symmetric, the transpose condition $R = P^T$ (Section 3.2) ensures that $M_{\text{map}}^{-1} = PM_{\text{old}}^{-1}P^T$ is also symmetric. Choosing the splitting matrices such that $M_{\text{post}} = M_{\text{pre}}^T$ makes the entire composite M_{comp}^{-1} symmetric [7, §14.3.2]. Moreover, provided each component is positive definite and the pre-smoother splitting $A = M_{\text{pre}} - N_{\text{pre}}$ satisfies $y^T(M_{\text{pre}}^T + N_{\text{pre}})y \geq 0$ for all y, the composite preconditioner is symmetric positive definite (SPD). A proof is given in Appendix A.2. When A is SPD, the preconditioned conjugate gradient (PCG) method requires M_{comp} to be SPD.

To fulfill the roles of M_{pre}^{-1} and M_{post}^{-1}, we return to the stationary iterative methods detailed in Section 2.2.1. Stationary methods update each cell's value using only its immediate neighbors. Consequently, they are slow at propagating information across a global domain, making them ineffective at resolving low-frequency errors. However, this same local reliance makes them highly efficient at damping high-frequency errors on locally uniform grid regions [9, p. 13].

More generally, each smoother can be applied ν times per preconditioner call, replacing G_{pre} and G_{post} in the composite factorization by G_{pre}^{ν} and G_{post}^{ν}. The analysis above corresponds to $\nu = 1$, which is the default throughout this chapter unless stated otherwise.

3.4 Numerical experiments

The following experiments validate the composite multiplicative preconditioner M_{comp}^{-1} on the 1D steady-state diffusion problem. Iteration count serves as the primary convergence metric, as it is a mathematical quantity independent of hardware and implementation details. The following subsections study the contribution of each preconditioner component to overall convergence, tune the smoother relaxation parameter, compare smoother variants by wall-clock timing, and examine whether grid-independent convergence is preserved when the solver, equation, or preconditioner is varied.

All parameter sweeps in this section use fifteen logarithmically spaced coarse grid sizes ($N_{\text{c}} \in [100, 100\,000]$) combined with all nine physics configurations from Section 3.1, giving $15 \times 9 = 135$ configurations. Here N_{c} denotes the number of cells on the coarse grid before refinement, which is

the controlled parameter. In each configuration the central 20% of the unit domain is refined by bisection, producing a fine grid of $N_\mathrm{f} > N_\mathrm{c}$ cells.

3.4.1 Ablation study

To empirically validate the theoretical framework developed in the preceding sections, an ablation study was performed on the 1D steady-state diffusion model problem from Section 3.1. The test configuration employs a non-uniform grid and a spatially varying diffusion coefficient, making it the most demanding of the nine physics configurations defined in Section 3.1 and therefore the appropriate baseline for isolating component contributions. Deriving exact analytical eigenvalue bounds for such a non-uniform system is complex. Consequently, empirical testing is needed to confirm the high-frequency damping argument formulated in Section 3.3.

An ablation study evaluates a composite method by systematically removing one component at a time, isolating the contribution of each part to the overall performance. Figure 3.7 presents the convergence behavior of the GMRES solver across four distinct preconditioning configurations as a function of the coarse grid size N_c. The number of iterations required to reduce the relative residual below the 10^{-5} tolerance threshold is recorded for each configuration.

For the tridiagonal system being solved (3.3), an ILU(0) factorization is equivalent to a direct solve requiring only a single iteration. Therefore, this ablation study does not aim to outperform an ILU(0) solve on the new grid, but rather to verify the grid-independent scaling of the composite preconditioner.

The baseline unpreconditioned solver exhibits the expected growth in iteration count as the grid is refined, demonstrating poor scalability for the unpreconditioned system. Applying only the symmetric Gauss-Seidel smoother (M_pre^{-1}, M_post^{-1}) improves upon this baseline by a consistent offset, requiring fewer iterations, but retains the same poor scaling behavior. The pre-smoother performs a forward Gauss-Seidel sweep and the post-smoother performs the backward sweep, the transpose of the pre-smoother, satisfying the symmetry condition $M_\mathrm{post} = M_\mathrm{pre}^T$ established in Section 3.3. This is consistent with the smoothing property established there, where the method updates each cell sequentially using only its immediate neighbors, making it slow at propagating information across the global domain and therefore ineffective at resolving low-frequency errors.

Applying only the mapped coarse-grid correction (M_map^{-1}), on the other hand, causes the solver to stall entirely. As shown in Figure 3.7, the solver terminates early due to lack of progress rather than reaching the maximum iteration limit.[1] This failure confirms the mechanism identified in the error analysis of Section 3.2.3. The zeroth-order prolongation operator injects high-frequency errors into the search directions, and because the mapped preconditioner is blind to them, these error modes are left entirely uncorrected. Without a complementary mechanism to handle this high-frequency noise, GMRES is left with no effective means to minimize the residual.

The optimal configuration applies both the mapped correction and the smoother together as the composite multiplicative preconditioner M_comp^{-1}. Under this configuration, the solver converges in fewer iterations. More importantly, the iteration count remains stable regardless of the underlying grid size. This grid-independent convergence rate is the hallmark of optimal algorithmic scalability.

The reduction and stabilization of iterations for the full multiplicative preconditioner empirically confirm the central argument of this chapter. By targeting and damping the high-frequency errors introduced during the mapping phase, the smoother acts as the mathematical complement to the

[1] Early termination on stagnation is specific to the GMRES implementation used in these experiments, discussed further in Section 3.4.3.

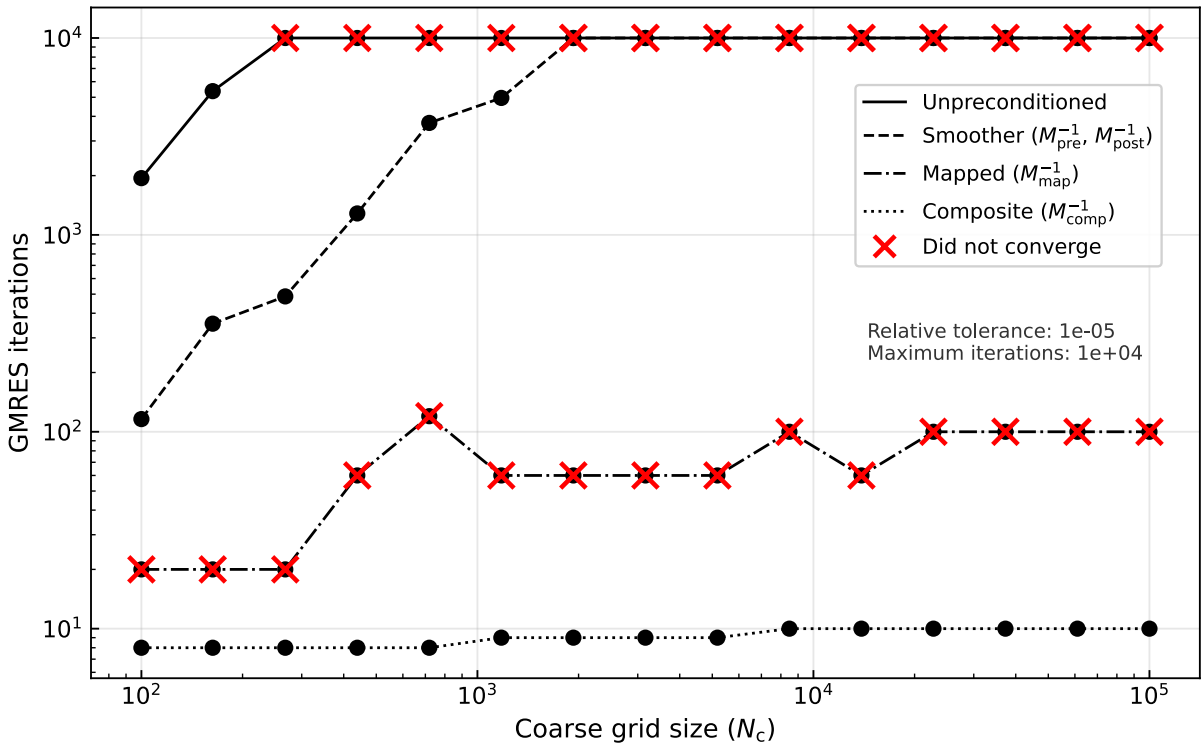


Figure 3.7: Ablation study of GMRES iteration counts across different preconditioning strategies as the central 20% of the unit domain was refined. The configurations isolate the individual and combined effects of a symmetric Gauss-Seidel smoother (M_{pre}^{-1}, M_{post}^{-1}) and a mapped ILU(0) preconditioner (M_{map}^{-1}). Only the full composite multiplicative preconditioner (M_{comp}^{-1}) achieves grid-independent convergence.

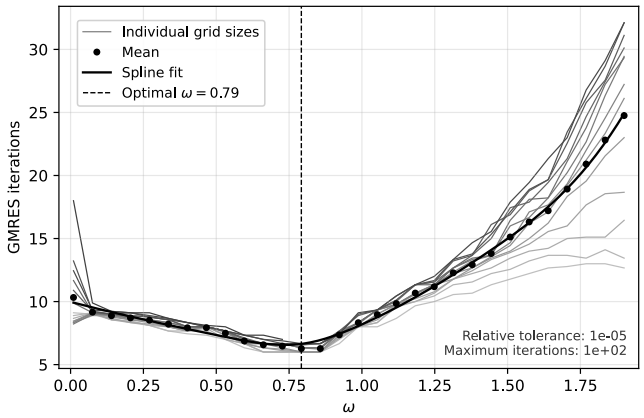
coarse-grid correction. Together, they filter the entire frequency spectrum, ensuring that the preconditioned system matrix $M_{\text{comp}}^{-1}A$ has a clustered eigenvalue spectrum and guarantees rapid, scalable convergence as the grid is refined.

3.4.2 Relaxation parameter tuning

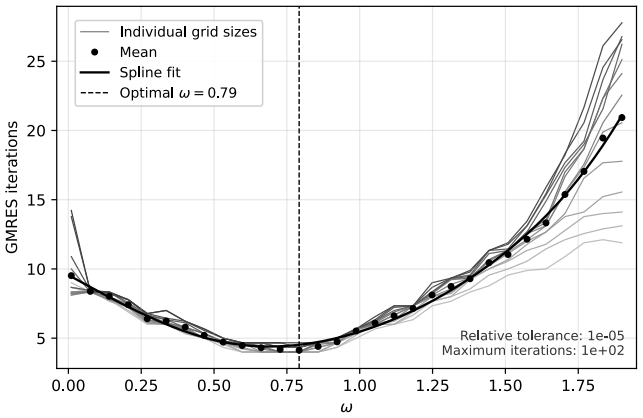
To ensure a fair comparison between smoothers, the relaxation factors (ω) of the successive over-relaxation (SOR), symmetric SOR (SSOR), and weighted Jacobi smoothers must first be tuned. A parameter sweep was conducted across the various grid sizes and the nine combinations of source terms and diffusion profiles defined in Section 3.1. Averaging the iteration counts across these configurations ensures that the observed performance is robust and not merely an artifact of a highly specific setup.

Averaging the optimal ω from individual tests is risky, as a value between two optima might perform poorly for both configurations. Instead, the relaxation factor that minimized the mean iteration count was selected.

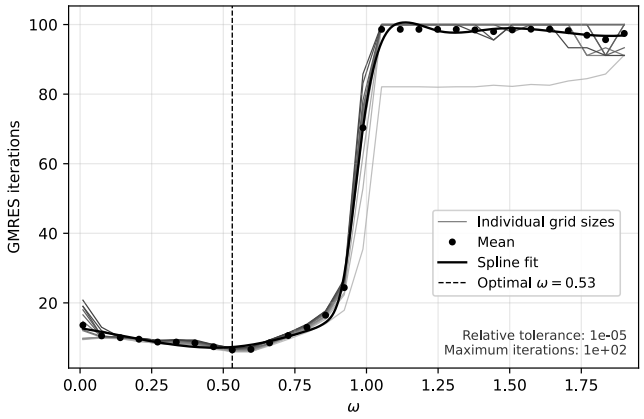
The SOR and SSOR smoothers share an optimal relaxation factor at $\omega = 0.79$. The weighted Jacobi smoother reaches its optimum at $\omega \approx 0.53$, as shown in Figure 3.8. Because the reported optimal ω for each smoother is identified directly from the minimum of the raw mean data, the final result remains independent of the visual smoothing spline.



(a) SOR smoother



(b) SSOR smoother



(c) Weighted Jacobi smoother

Figure 3.8: Optimization of the relaxation factor ω for the composite preconditioner. The parameter sweep evaluated fifteen logarithmically spaced coarse grid sizes ($N_c \in [100, 100\,000]$) across all nine combinations of source terms and diffusion profiles defined in Section 3.1. In every combination, the central 20% of the unit domain was refined. Thin gray lines display the iteration counts for individual grid sizes, confirming grid-independent behavior. For each tested $\omega \in [0.01, 1.9]$, the mean iteration count across all combinations is plotted alongside a smoothing spline to guide the eye. The vertical dashed line indicates the optimal ω that minimizes the mean iteration count.

Figure 3.8c shows that for the Jacobi smoother, a spike in iteration count near $\omega = 1$ occurs because the eigenvalue of the iteration matrix for the highest-frequency error approaches -1, which flips the error's sign rather than damping it [9, p. 21]. Because the prolongation operator injects exactly this error mode (Section 3.2.3), the unrelaxed smoother struggles to remove it. This leaves the spectrum unclustered, forcing GMRES to compensate and causing the iteration spike. Under-relaxation ($\omega < 1$) resolves this inefficiency by shifting these eigenvalues toward zero. The data show optimal convergence around $\omega \approx 0.53$, somewhat below $\omega = \frac{2}{3}$ derived for the uniform-grid 1D Laplacian [9, p. 21].

3.4.3 Smoother comparisons

This subsection compares the four smoothers (weighted Jacobi, Gauss-Seidel, SOR, and SSOR) across three complementary perspectives. First, wall-clock timing is used to confirm that both the build and solve phases scale linearly with grid size and to quantify the practical cost differences between smoothers. Second, the geometric mapping operators are repurposed to provide an initial guess for the Krylov solver. Third, the effect of the smoother application count ν on convergence and total execution time is examined.

Wall-clock timing

To understand these measurements, we first separate the computations into two distinct phases. Matrix assembly and the geometric mapping operations are implemented in Python via SciPy [18], while the Krylov iteration is executed by PETSc [19] through the `petsc4py` interface [20].[2] The build phase (t_{build}) covers the time required to construct the preconditioner, from assembling the operators in SciPy to completing the internal PETSc preconditioner initialization. The solve phase (t_{solve}) is the time the Krylov solver spends driving the relative residual below a threshold.

To isolate algorithmic scaling from hardware and OS noise, timings were recorded with automatic memory management suspended to prevent unpredictable pauses. Each phase's minimum time across three identical runs was retained. Additionally, the initial guess was reset to zero before each solve to ensure identical starting states.

As Figure 3.9 demonstrates, the build times are indistinguishable across all four smoothers. This is expected, since every stationary smoother extracts the same splitting from A, and the construction cost is therefore independent of the smoother choice. Both phases asymptotically approach linear $\mathcal{O}(N_{\text{c}})$ scaling, consistent with the tridiagonal structure of the 1D model problem [21, §4.3.6]. The solve times, however, exhibit a constant offset. Because a single iteration takes roughly the same amount of time regardless of the smoother used, the total solve time depends on how many iterations are required. Therefore, the offset simply reflects that some smoothers need fewer total iterations to reach convergence than others. The per-physics variance in solve time is masked by the averaging.

[2] Python 3.12.3; `petsc4py` 3.23.4.

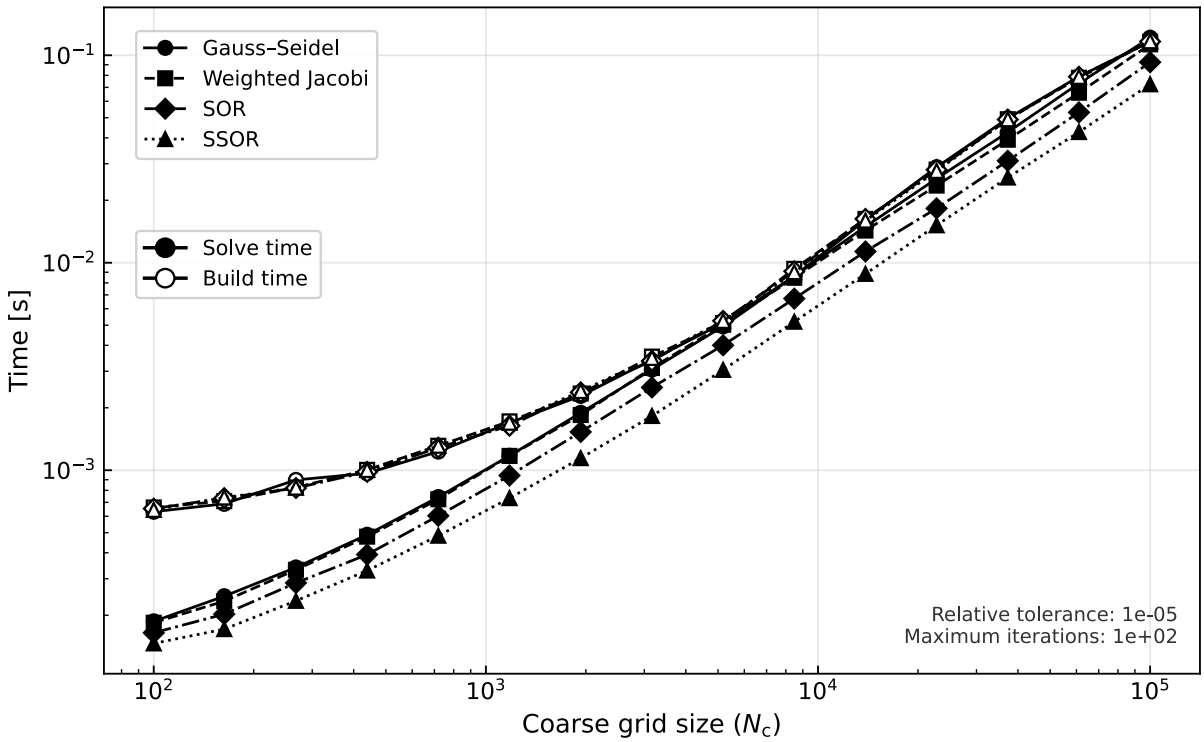


Figure 3.9: Build and solve times in the GMRES solver for the composite preconditioner with each of the four smoothers across increasing grid sizes, averaged over all nine physics configurations (Section 3.4). Both phases asymptotically approach linear $\mathcal{O}(N_c)$ scaling for larger grids. While the build times are indistinguishable across the smoothers, the solve times exhibit an offset for large grids corresponding to the total number of Krylov iterations required to reach convergence.

Mapped initial guess

A potential additional benefit of the geometric mapping operators is that they can be reused to project the solution computed on the old grid onto the new grid.

To evaluate this strategy, the GMRES solver with the composite preconditioner was tested using both a standard zero-vector initialization ($u^{(0)} = 0$) and the mapped initial guess ($u^{(0)} = Pu_c^*$), where $u_c^* \in \mathbb{R}^{N_c}$ is the exact coarse-grid solution. Both initializations were tested across all four smoothers at their respective optimal relaxation factors. Among the smoothers, SSOR requires the fewest baseline iterations, followed by the optimally weighted Jacobi and SOR smoothers, which achieve comparable counts. Gauss-Seidel, which has no tunable relaxation factor, requires the most iterations (Section 3.4.2).

Figure 3.10 shows that all four smoothers achieve fewer iterations with the mapped initial guess, but to different degrees. Jacobi, SOR, and SSOR each reduce their mean iteration count by roughly one to two, while Gauss-Seidel gains less than one iteration on average.

The difference reflects how effectively each smoother can reduce the initial residual $\|f - APu_c^*\|$. Applying post-smoothing sweeps to Pu_c^* damps the high-frequency oscillations introduced by prolongation before the solve begins, which lowers the initial residual and provides a better starting point. For Gauss-Seidel, post-smoothing is less effective at damping these oscillations, yielding only a modest benefit.

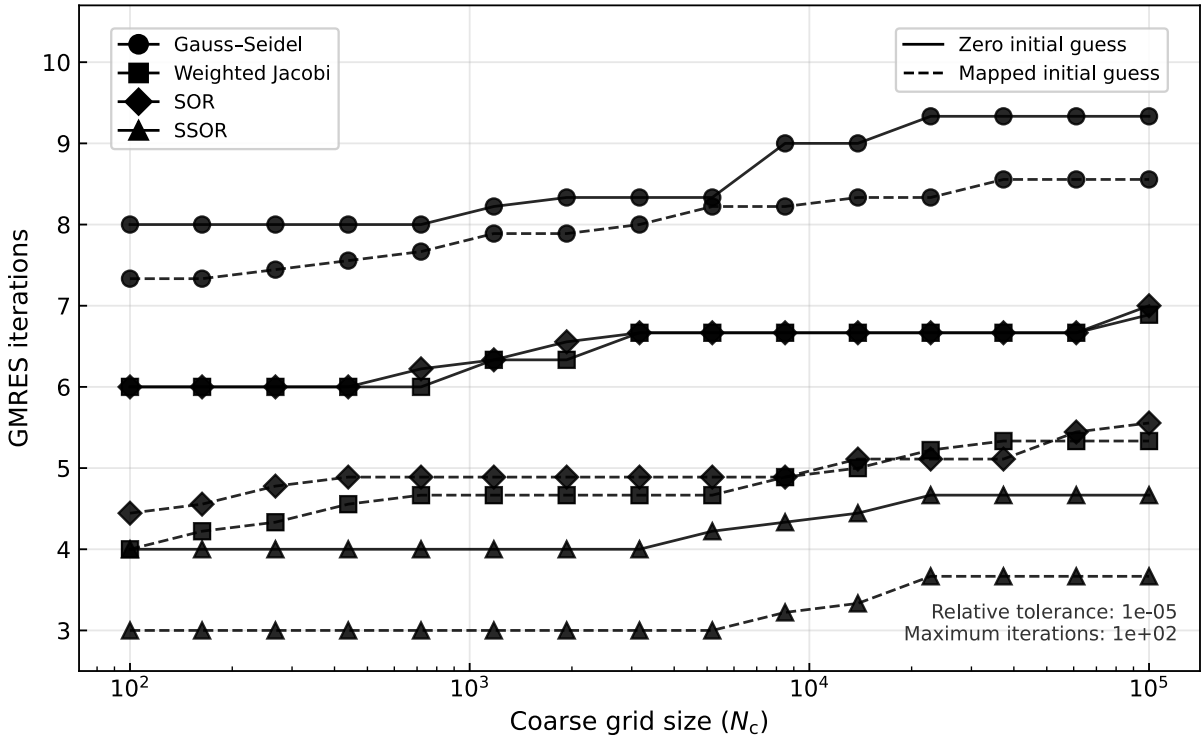


Figure 3.10: Impact of using the mapped coarse-grid solution as an initial guess for the GMRES solver. The mean iteration counts are averaged across all nine physics configurations (Section 3.4). All four smoothers benefit from the mapped initial guess, though the reduction is smaller for Gauss-Seidel than for the other three.

Smoother application count

As noted in Section 3.4.1, the ILU(0) factorization of the tridiagonal system is equivalent to a direct solve, so the iterations that remain after the coarse-grid correction are spent almost entirely reducing the high-frequency errors introduced by the prolongation operator. This motivates investigating whether applying the smoother $\nu > 1$ times per preconditioner call can damp these errors more aggressively.

The smoother application count ν, introduced in Section 3.3, controls how many times the pre- and post-smoother are each applied within a single preconditioner call. To assess its effect, the GMRES solver was run with all four smoothers at their respective optimal relaxation factors for $\nu \in \{1, 2, \ldots, 8\}$, averaged over all 135 configurations described in Section 3.4.

Figure 3.11 confirms that increasing ν reduces the mean iteration count across all four smoothers. The largest gains occur at small ν, after which the returns diminish. This reflects that the smoother most effectively targets high-frequency error modes in its first few applications. The remaining low-frequency errors are less sensitive to further smoothing and must be resolved by the coarse-grid correction regardless of how large ν becomes.

However, fewer iterations do not directly imply faster execution. Each unit increase in ν adds one extra pre-smoother sweep and one extra post-smoother sweep per preconditioner call, raising the computational cost per GMRES iteration. Because wall-clock solve times scale as $\mathcal{O}(N_c)$ for all four smoothers, the times are normalized by N_c to allow aggregation across grid sizes. Figure 3.12 shows this normalized solve time as a function of ν.

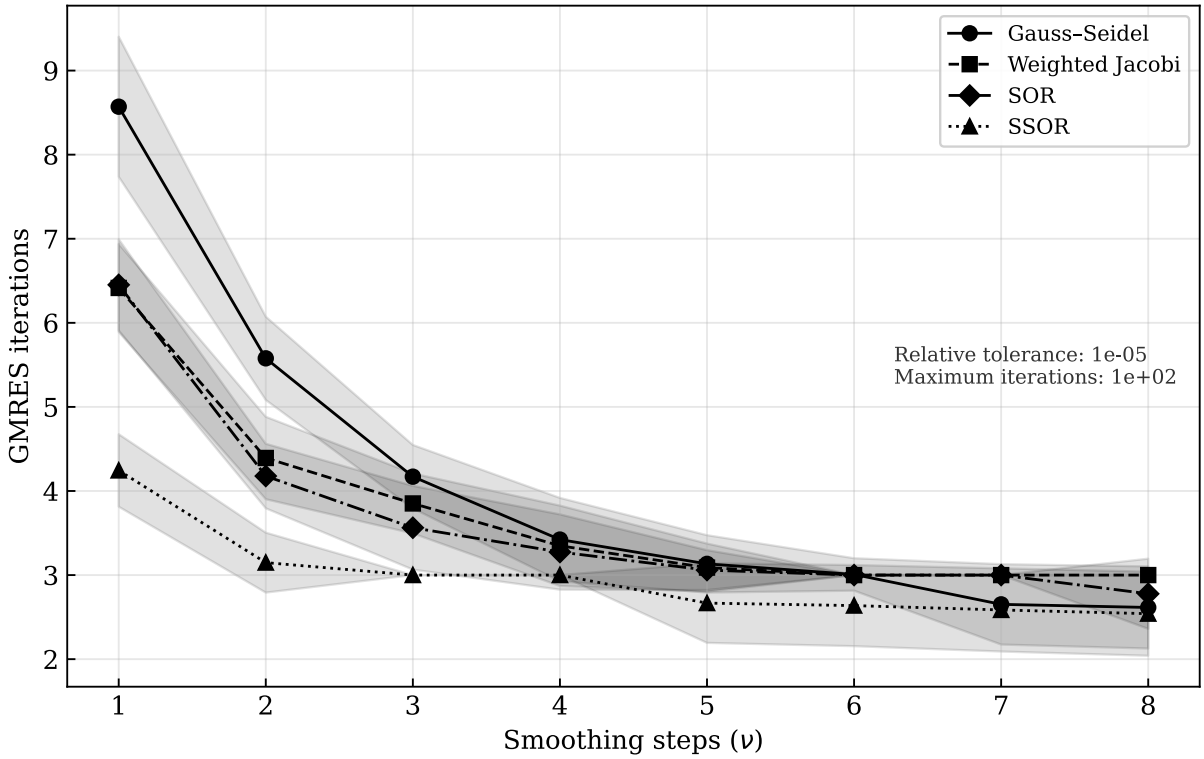


Figure 3.11: Mean GMRES iteration count as a function of smoother application count ν for all four smoothers at their respective optimal relaxation factors. Counts are averaged over all 135 configurations described in Section 3.4, with the shaded band showing ± 1 standard deviation across those configurations. Increasing ν reduces iteration count for all smoothers, with diminishing returns at larger values.

For small ν, the reduction in iterations outweighs the per-iteration cost, decreasing overall solve time. Beyond a crossover point, however, smoother overhead outpaces the diminishing iteration savings, causing solve time to rise. Because aggregating diverse problem configurations makes the exact minimum noisy, establishing a precise universal optimum is unnecessary. The critical finding is the presence of the crossover itself. This validates that the smoother should not act as a brute-force solver. Balancing smoother intensity against Krylov work requires keeping ν small.

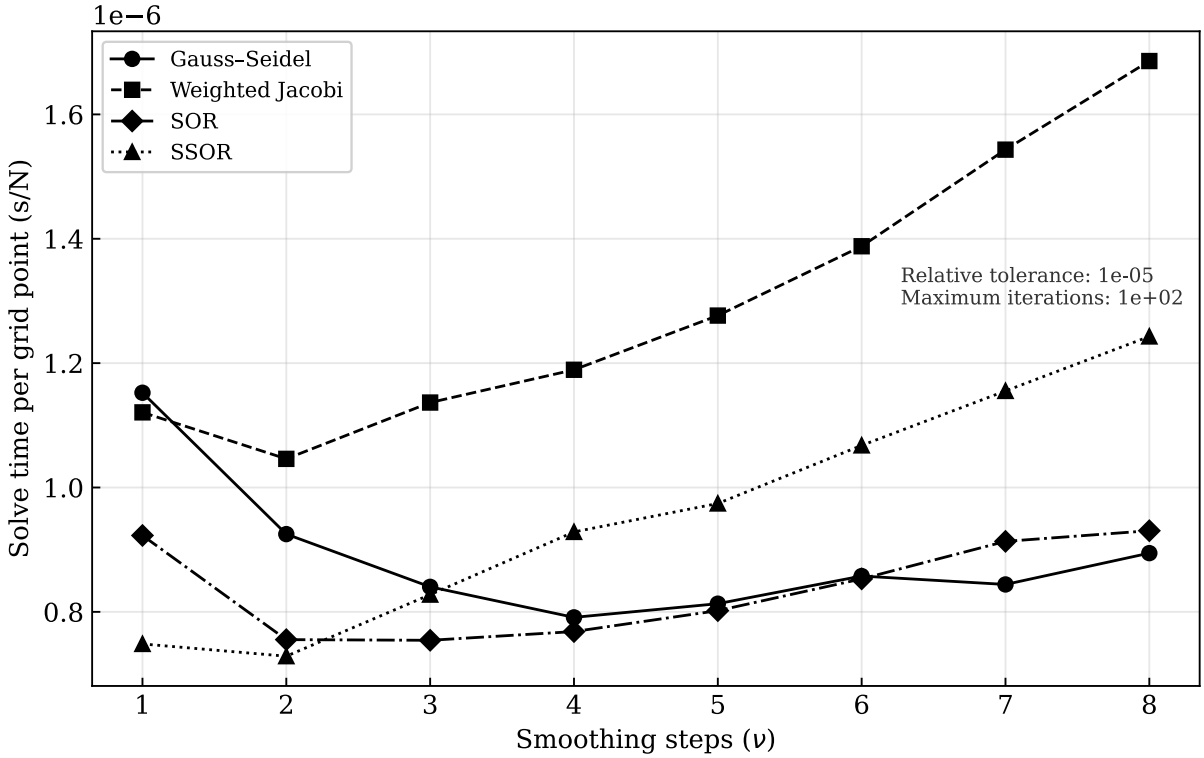


Figure 3.12: Solve time per grid point as a function of smoother application count ν for all four smoothers at their respective optimal relaxation factors. Wall-clock solve times are normalized by N_c and averaged over all 135 configurations described in Section 3.4. Beyond a crossover point, the additional cost per iteration outweighs the savings from fewer iterations.

3.4.4 Robustness analysis

The ablation study in Section 3.4.1 established that the full composite preconditioner achieves grid-independent convergence under one specific configuration, namely a purely diffusive equation, an exact coarse-grid solve via ILU(0), and the GMRES solver. To verify that this optimal scaling is a property of the mapped architecture rather than an artifact of that particular setup, the method is subjected to three independent structural variations.

In the first variation, the exact ILU(0) coarse-grid preconditioner is replaced by an unweighted Jacobi preconditioner, which ignores all off-diagonal spatial coupling. Since ILU(0) applied to the tridiagonal system is equivalent to a direct solve (Section 3.4.1), this substitution represents the most extreme downgrade in coarse-grid accuracy.

In the second variation, the governing equation is changed from pure diffusion to a two-variable reaction-diffusion system. Letting $u(x)$ and $v(x)$ denote two interacting quantities, the coupled steady-state system takes the form

$$\begin{cases} -\dfrac{d}{dx}\left(D(x)\dfrac{du}{dx}\right) + k_1 u - k_2 v = f(x) \\ -\dfrac{d}{dx}\left(D(x)\dfrac{dv}{dx}\right) - k_1 u + k_2 v = f(x), \end{cases} \tag{3.25}$$

with coupling constants $k_1 = 5.0$ and $k_2 = 2.0$, and the finite volume discretization handled by

FiPy [15]. The cross-coupling between the two components transforms the discrete system into a nonsymmetric $2N \times 2N$ block matrix.

In the third variation, the GMRES solver is replaced by the conjugate gradient (CG) method, which requires both the system and the preconditioner to be SPD. This property is guaranteed by the composite architecture when the smoother satisfies $M_{\text{post}} = M_{\text{pre}}^T$ (Section 3.3). For all three tests the smoother is fixed to symmetric Gauss-Seidel with $\nu = 1$, and all remaining settings, including the 135 configurations and the 10^{-5} relative tolerance, match those used in the preceding experiments.

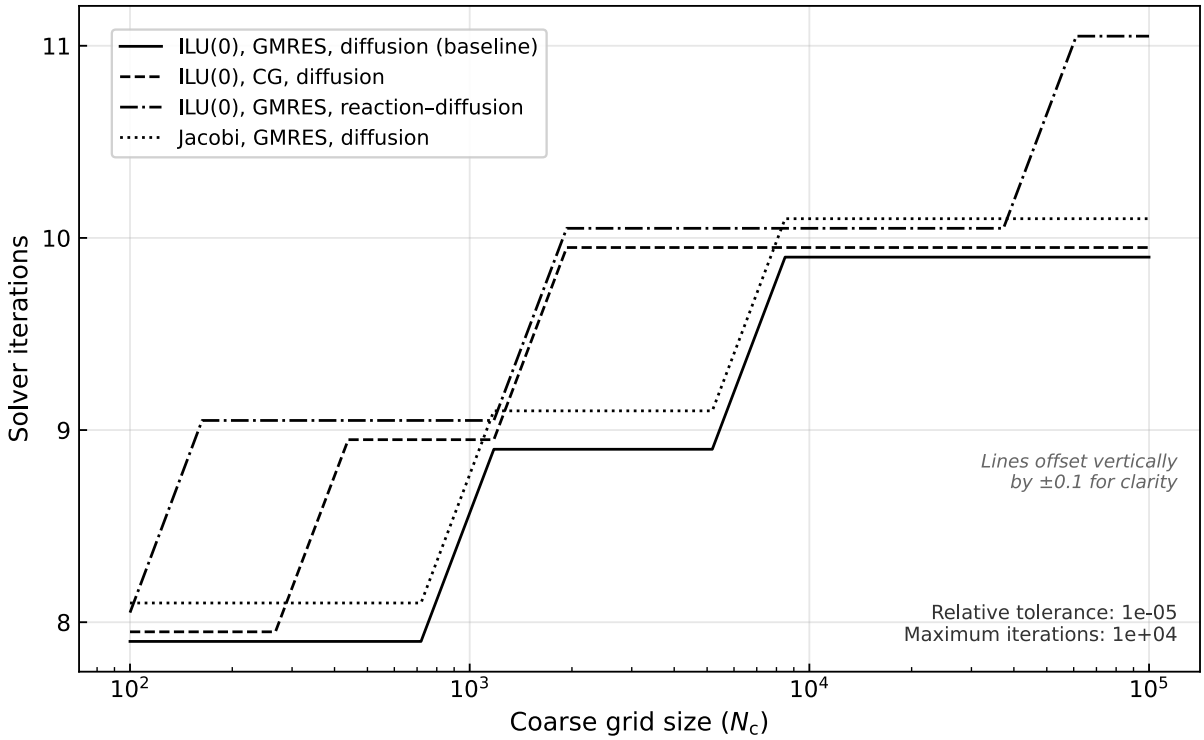


Figure 3.13: Mean iteration counts as a function of coarse grid size N_c for three structural variations of the baseline configuration (ILU(0) coarse-grid solve, GMRES solver, pure diffusion) from Section 3.4.1. Each data point represents the average over the nine physical configurations defined in Section 3.1. Replacing the ILU(0) coarse-grid preconditioner with an unweighted Jacobi preconditioner, introducing a coupled reaction-diffusion system, and switching to the CG solver each leave the iteration count approximately flat across three orders of magnitude in N_c.

Figure 3.13 presents the mean iteration counts for all three variations alongside the baseline from Section 3.4.1. Despite the degradation in coarse-grid solve quality, the introduction of coupled multiphysics, and the use of the CG solver, the iteration counts remain approximately flat across three orders of magnitude in N_c in every case. Each variation was tested independently rather than in combination. Whether the architecture sustains grid-independent convergence under simultaneous stress, such as the Jacobi coarse-grid preconditioner applied to the reaction-diffusion system with the CG solver, remains an open question. The survival of this optimal convergence rate under each structural variation nevertheless confirms that the grid-independent scaling observed throughout this chapter is not an artifact of a specific solver or equation, but a structural consequence of the mapped architecture clustering the eigenvalues of the preconditioned system.

4

Application to FLASH

Chapter 3 validated preconditioner reuse on a 1D model problem using prolongation and restriction operators P and R. The grid there was a plain array of cell faces, and the operators were derived directly from the old and new coordinate arrays with no awareness of any block structure.

Lifting this approach into FLASH requires two additional ingredients that Chapter 3 did not need. FLASH manages the grid via PARAMESH, a parallel adaptive mesh refinement library that organizes the domain as a distributed binary tree of fixed-size blocks [13, §II]. Each block carries per-block metadata, including the refinement level, corner index, and node type, that must be used to identify what changed between two grid states. Furthermore, an AMR step in FLASH is not restricted to a single global operation and can refine some blocks while derefining others in the same step, a mixed case that P and R individually cannot represent. The method is implemented within FLASH as an extension to its HYPRE-based linear solver [17, p. 632].[3]

Section 4.1 introduces the PARAMESH block data structure and the stable level–corner-index address that makes cross-grid block matching possible. Section 4.2 defines the transition operator T, which generalizes P and R to the mixed case. Section 4.3 tests the implementation on a 1D heat-conduction problem and measures the performance gains from preconditioner reuse.

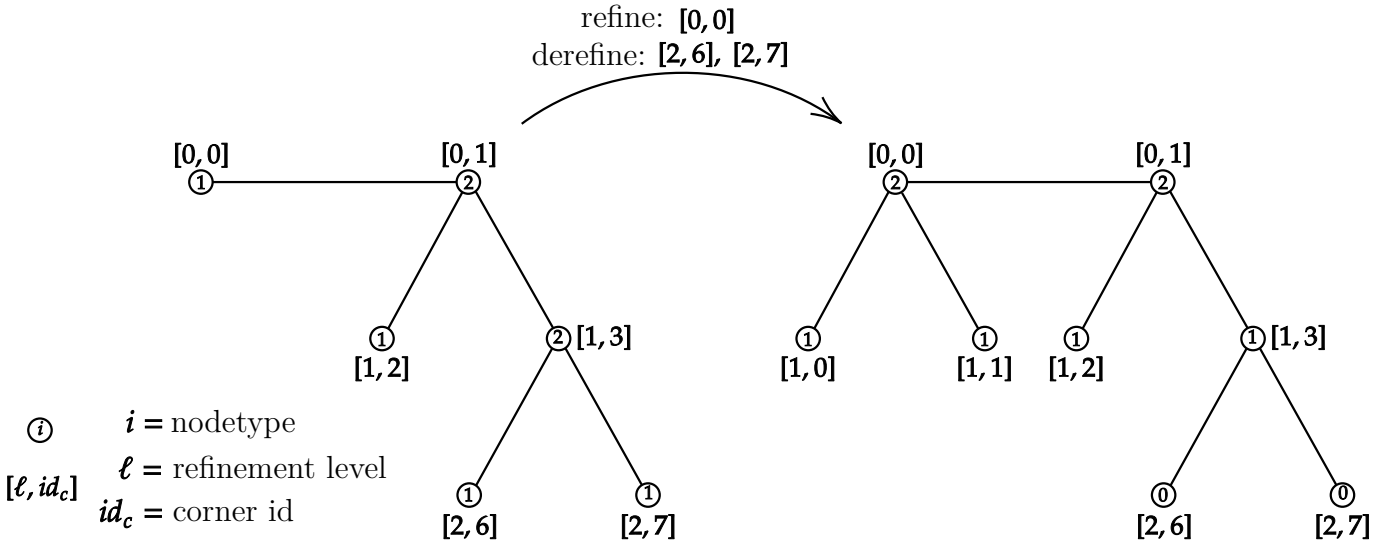
4.1 The PARAMESH structure

Section 2.4 described adaptive mesh refinement (AMR) at the conceptual level. PARAMESH realizes this structure as a tree hierarchy of fixed-size blocks, with each block carrying a stable spatial identity that persists across refinement events elsewhere in the domain [13, §II]. This section defines that data structure, because the transition operator of Section 4.2 uses it to determine how the preconditioner is mapped from the old grid to the new one. The running example throughout is the one illustrated in Figure 4.1, in which refining block [0, 0] and derefining the sibling pair {[2, 6], [2, 7]} in a single AMR step produces three outcomes simultaneously, namely an unchanged block, a refined block, and a derefined block. The following subsections define the block hierarchy, the nodetype flag, and the stable spatial address used throughout Section 4.2.

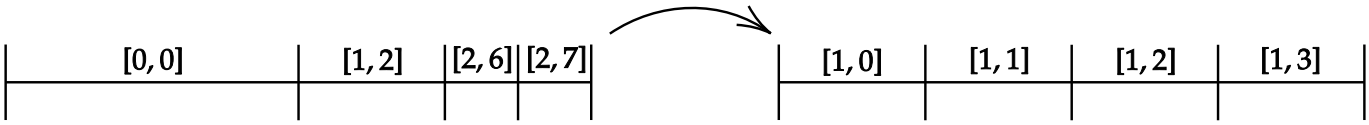
Block hierarchy

In PARAMESH, the domain $[x_{\min}, x_{\max}]$ is partitioned at the coarsest level into n_{root} uniform root blocks, each containing n_{cells} cells. Section 2.4 left this count implicit, but PARAMESH fixes it as a parameter so that every block at every refinement level has the same number of cells. Grid refinement proceeds by splitting blocks, not by inserting additional cells within a block.

[3]The work in this chapter is the author's own. The software engineering required to integrate the method into FLASH was carried out by Alan Cotino.



(a) Binary tree structure, with each node showing its nodetype (circled integer) and $[\ell,\, id_c]$ pair of refinement level ℓ and corner index id_c. Nodetype 1 marks active leaves, nodetype 2 marks inactive parents, and nodetype 0 marks deactivated children.



(b) Spatial decomposition of the domain. Active leaf blocks are labeled by their $[\ell,\, id_c]$ pair, with ticks marking block boundaries.

Figure 4.1: The two AMR states used as the running example throughout this section. Starting from the left state, refining block [0, 0] and derefining the sibling pair {[2, 6], [2, 7]} produces the right state, illustrating an unchanged block, a refined block, and a derefined block simultaneously.

When a block is refined, it is replaced by r^d child blocks, where $r = 2$ is the refinement ratio and d is the number of spatial dimensions, giving two children in 1D, four in 2D, and eight in 3D. The depth of a block in the tree equals its refinement level ℓ, stored in the per-block `level` quantity, with $\ell = 0$ for root blocks. Two per-block quantities encode the tree structure, as illustrated in Figure 4.1a. The parent index `parent_idx` stores the index of the parent block in PARAMESH's block array and is -1 for root blocks. The children indices `children_idx` store the indices of the 2^d child blocks and are empty for leaf blocks. Each block also carries a `nodetype` flag and a stable spatial identifier, defined in the following two subsections.

Node types

Every block carries a `nodetype` flag that records its current role in the tree. A block with `nodetype` = 1 is an *active leaf*, part of the current grid. A block with `nodetype` = 2 is an *inactive parent* that has been refined and no longer belongs to the current grid. A block with `nodetype` = 0 is a *deactivated child* whose parent has been derefined and reactivated. Only the active leaves contribute rows to the system matrix A, each occupying n_{cells} contiguous rows.

When a block is refined, it transitions from nodetype $1 \to 2$, and 2^d new child blocks are assigned nodetype 1. In derefinement, both siblings transition $1 \to 0$ and their parent transitions $2 \to 1$. Both

siblings must derefine together, as a parent cannot be reactivated while either child remains active.

Block addressing

The active leaf blocks, sorted from left to right by spatial position, determine the row ordering of the system matrix A. The cells of the spatially leftmost block occupy the first n_{cells} rows, the next block the following n_{cells} rows, and so on, and each block is assigned a starting row index n_{start} that locates its cells within A.

Since n_{start} depends on the current active leaf set, it changes whenever the grid is refined or derefined. The refinement level ℓ and the *corner index* id_c together provide a stable spatial address that persists across grid changes. At level ℓ, the domain is divided into 2^ℓ equal sub-intervals, and id_c identifies which one the block covers, counting from the left. The pair $[\ell,\, id_c]$ always refers to the same spatial sub-interval, regardless of what refinements have occurred elsewhere.

The corner index obeys a simple inheritance rule. When a block at level ℓ with corner index id_c is refined, its left child receives corner index $2\,id_c$ and its right child receives $2\,id_c + 1$, both at level $\ell + 1$. For example, refining block [0, 0] in Figure 4.1a produces children [1, 0] and [1, 1]. On derefinement, the parent's corner index is recovered as $\lfloor id_c/2 \rfloor$ from either child, where $\lfloor \cdot \rfloor$ denotes the floor function. For the running example, the sibling pair {[2, 6], [2, 7]} recovers parent [1, 3].

The *address book* maps each active leaf's $[\ell,\, id_c]$ pair to its starting row index n_{start} in A.

4.2 The transition operator

P and R from Section 3.2 handle pure refinement and derefinement steps respectively. However, a single AMR step may refine some blocks while derefining others simultaneously, as in Figure 4.1, so neither P nor R covers this case. The transition operator T handles the general case by assembling the same local operators from Section 3.2.1 into a single $N_{\text{new}} \times N_{\text{old}}$ matrix, where N_{new} and N_{old} are the total cell counts on the new and old grids. With T in place of P and R, the mapped preconditioner becomes

$$M_{\text{map}}^{-1} = T M_{\text{old}}^{-1} T^T, \tag{4.1}$$

and the composite structure and symmetry properties of Section 3.3 carry over without modification, since neither the composite formula nor the error factorization assumes any specific form for M_{map}^{-1}, and $T M_{\text{old}}^{-1} T^T$ is symmetric whenever M_{old}^{-1} is. The following subsections describe how T is assembled from the address book of Section 4.1, how this assembly generalizes to grids that differ by more than one refinement level, and how the implementation decides when to rebuild the preconditioner from scratch.

Matrix assembly

T follows the same block-by-block assembly as P and R, applying $P_{\text{int}}^{\text{loc}}$, $R_{\text{int}}^{\text{loc}}$, or the identity to each new leaf block depending on how its $[\ell,\, id_c]$ pair relates to the old address book.

If the pair appears in the old address book, the block is unchanged and each new cell copies the corresponding old cell via the identity matrix. If the pair is absent but an ancestor is found by walking up the tree, the block is the result of a refinement. The sibling position $id_c \bmod 2$ selects which half of the old leaf's cells the new block covers, and each new fine cell copies the corresponding old coarse cell using $P_{\text{int}}^{\text{loc}}$. If no ancestor is found, the block is the result of a derefinement, as the old

grid covered its domain with finer leaves. Each new coarse cell then averages the old fine cells that fall within it, using $R^{\text{loc}}_{\text{int}}$.

For the AMR step of Figure 4.1, the old active leaf blocks in spatial order are $[0,0]$, $[1,2]$, $[2,6]$, $[2,7]$, and the new active leaf blocks are $[1,0]$, $[1,1]$, $[1,2]$, $[1,3]$, giving the transition operator for $n_{\text{cells}} = 2$ as

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.5 & 0.5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0.5 \end{bmatrix}. \tag{4.2}$$

New blocks $[1,0]$ and $[1,1]$ are refined descendants of old leaf $[0,0]$. Both cells of $[1,0]$ lie within the left cell of $[0,0]$, so each takes the value of that old coarse cell. Both cells of $[1,1]$ lie within the right cell, so each takes its value. Block $[1,2]$ is unchanged, so each new cell copies the old cell at the same position. Block $[1,3]$ is the result of a derefinement, since no ancestor of $[1,3]$ appears in the old address book. Old leaves $[2,6]$ and $[2,7]$ covered its domain, so the left cell of $[1,3]$ averages the two cells of $[2,6]$ and the right cell averages the two cells of $[2,7]$.

Multi-level transfer

The assembly above assumes that the old and new grids differ by at most one refinement level at any block boundary. The grid at which the preconditioner was most recently built is held fixed as the *anchor grid*. In a time-dependent FLASH simulation, multiple AMR steps can accumulate between successive solves, so the anchor grid and the current grid may differ by more than one level.

Every assembly of T maps from the anchor to the new grid. Each new leaf block is still classified as unchanged, refined, or derefined, but the upward trace for the refined case now walks as many levels as needed until an ancestor is found.

When an ancestor is found at level ℓ_{old}, with level difference $\Delta\ell = \ell - \ell_{\text{old}}$, the new leaf occupies sub-interval $s = id_c \bmod 2^{\Delta\ell}$ of the ancestor's domain. Each new fine cell i then copies old coarse cell $\lfloor (s \cdot n_{\text{cells}} + i)/2^{\Delta\ell} \rfloor$, generalizing $P^{\text{loc}}_{\text{int}}$ to the multi-level case.

For the derefinement case, old leaves at a finer level cover the new block's domain and may themselves sit at different levels relative to the anchor. All old leaves $[\ell_{\text{old}},\, id_{c,\text{old}}]$ satisfying $\lfloor id_{c,\text{old}}/2^{\ell_{\text{old}}-\ell} \rfloor = id_c$ are spatial descendants of the new block, and each contributes to the row of T for every new coarse cell it overlaps. Each old cell within a contributing leaf at level ℓ_{old} is assigned weight $2^{-(\ell_{\text{old}}-\ell)}$, the fraction of the new coarse cell's spatial extent that old cell occupies. The multi-level case is not separately verified in the numerical experiments of Section 4.3.

Anchor rebuild

M_{old} was built for the system at the anchor grid. As the simulation advances, both the grid and the problem coefficients can change, making M_{old} a worse approximation of the new system A. The iteration count can therefore serve as a proxy for preconditioner quality. After a fresh anchor is built, the iteration count from that first solve is recorded as the baseline k_0. On each subsequent reuse-path solve, if the iteration count exceeds $\max(2k_0, 10)$, a fresh anchor rebuild is scheduled for

the next solve. The threshold $2k_0$ tolerates transient spikes while catching genuine degradation, and the floor of ten prevents a small baseline from triggering premature rebuilds, though both values are heuristic. The rebuild recomputes M_{old} on the new grid. The new iteration count establishes a new baseline, and the cycle begins again.

In the FLASH implementation, T is assembled using HYPRE's matrix-assembly routines, with rows classified using the refinement level and corner index that PARAMESH provides for every leaf block. The anchor, comprising the matrix M_{old}^{-1} and the corresponding grid, is maintained between grid refinement and linear solver setup at each time step. The rebuild heuristic monitors the PCG iteration count returned by the solver after each step and triggers a fresh factorization when the count exceeds the adaptive threshold.

4.3 Numerical experiments in FLASH

The test problem models the diffusion of heat in a 1D domain under dynamic adaptive mesh refinement (AMR). Unlike the static, prescribed refinement of Chapter 3, the grid here evolves according to a refinement criterion evaluated at each step, so the topology is not known in advance. The domain is the interval $[0, 4]$, the diffusion coefficient D is constant, and the governing equation is

$$\frac{\partial u}{\partial t} = D\,\frac{\partial^2 u}{\partial x^2}. \tag{4.3}$$

The boundary conditions are zero-flux at both ends, $\partial u/\partial x\big|_{x=0} = \partial u/\partial x\big|_{x=4} = 0$. The initial condition is a Gaussian centered at $x = 2$,

$$u(x,\,0) = \frac{Q}{\sqrt{4\pi D \times 10^{-3}}}\exp\!\left(-\frac{(x-2)^2}{4D \times 10^{-3}}\right), \tag{4.4}$$

with $Q = 10^6$. PARAMESH is configured with $n_{\text{root}} = 4$ root blocks of $n_{\text{cells}} = 8$ cells each and a maximum refinement level $\ell_{\max} = 8$. Time is discretized with a fixed step $\Delta t = 10^{-6}$ s using the Backward Euler method, and the PCG solver uses a relative tolerance of 10^{-12} and an ILU preconditioner. The simulation runs for 2000 steps. All runs use a single MPI process, and extension to multi-process distribution is left for future work.[4]

Two runs are compared throughout this section. The baseline run rebuilds the ILU preconditioner at every time step. The reuse run applies the composite preconditioner of Section 3.3, with the mapped component (4.1) and a standard Jacobi smoother ($\omega = 1$, $\nu = 1$) as the pre- and post-smoother. A fresh ILU rebuild is triggered only when the criterion of Section 4.2 is met. Before comparing their performance, two properties of the simulation must be established. The first is that both refinement and derefinement are triggered during the run, confirming that the simulation operates under dynamic AMR. The second is that *mixed steps* occur, so T is actually invoked on steps where both row types must be assembled simultaneously.

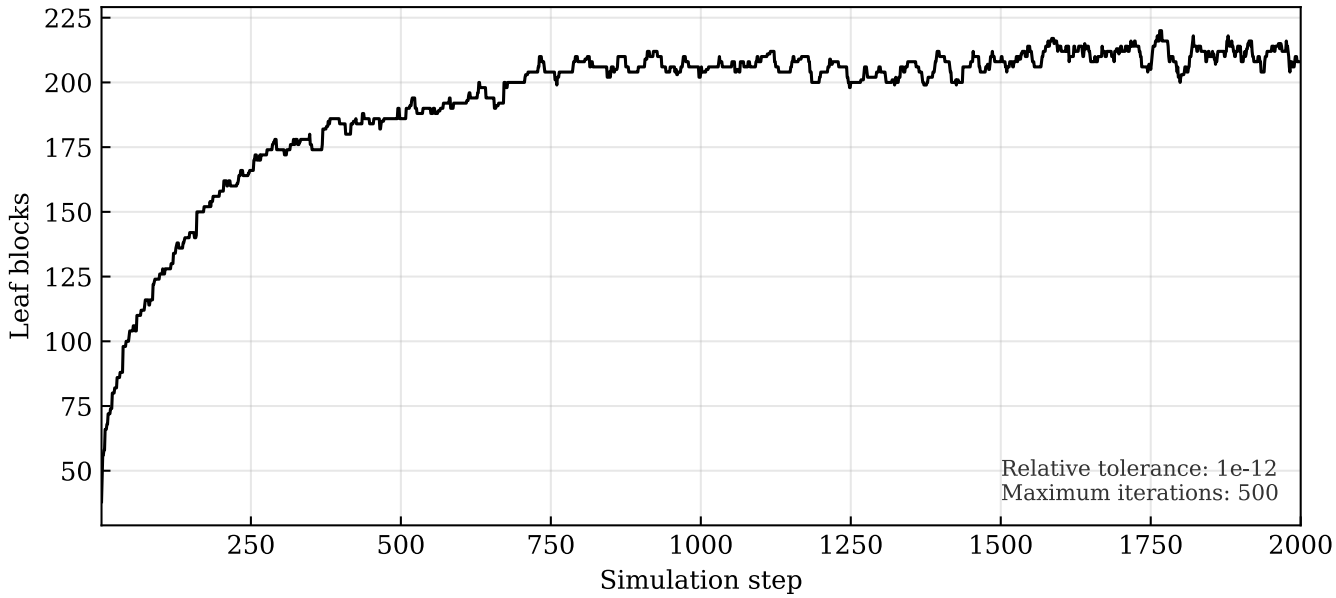
[4]Detailed configuration files and simulation logs are maintained internally by Ignition Computing. For further information or to request specific setup details, please contact info@ignitioncomputing.com.

AMR dynamics and the transition operator

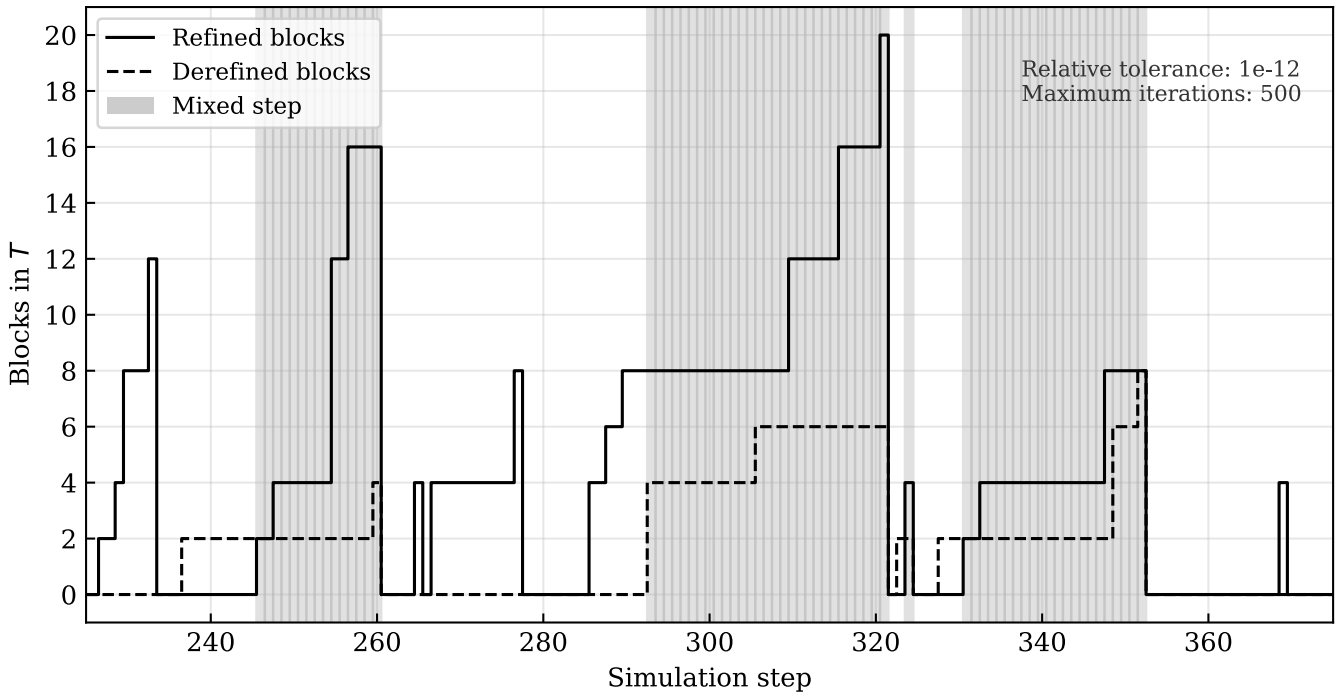
Figure 4.2 characterizes the AMR behavior of the simulation and the role of T within it.

Figure 4.2a shows the leaf block count over all 2000 steps. The grid grows as the diffusing profile spreads into the initially flat regions to its sides, and reaches an equilibrium between refinement and derefinement. Both events occur throughout the run.

Figure 4.2b shows the refined and derefined block counts appearing in T at each step over a representative window, with mixed steps shaded. Mixed steps occur throughout the run, confirming that T must handle both row types simultaneously. Multi-level transfer, in which the anchor and new grids differ by more than one refinement level, is implemented as described in Section 4.2, but is not separately verified in this test case.



(a) Leaf block count as a function of step number.



(b) Refined and derefined block counts in T over steps 225–375, with mixed steps shaded.

Figure 4.2: AMR dynamics of the test problem. (a) Both refinement and derefinement occur throughout the run. (b) Mixed steps occur throughout the run, requiring T to assemble both row types simultaneously.

Performance

Figure 4.3 shows the PCG iteration count per simulation step for both runs, with the threshold $\max(2k_0, 10)$ overlaid. The baseline requires several iterations per step rather than one, because HYPRE's ILU factorization does not reduce to a direct solve as it does for the tridiagonally ordered system of Chapter 3 (Section 3.4.1). The reuse run shows a sawtooth pattern. After each anchor rebuild, the count is low and close to the baseline, but as the anchor ages and the grid drifts from it, the count climbs until it crosses the threshold, after which the next step triggers a rebuild. Of the 2000 steps, 199 trigger a rebuild, giving a reuse rate of 90.1%. PCG convergence is maintained throughout the run, with the iteration count reaching at most 34.

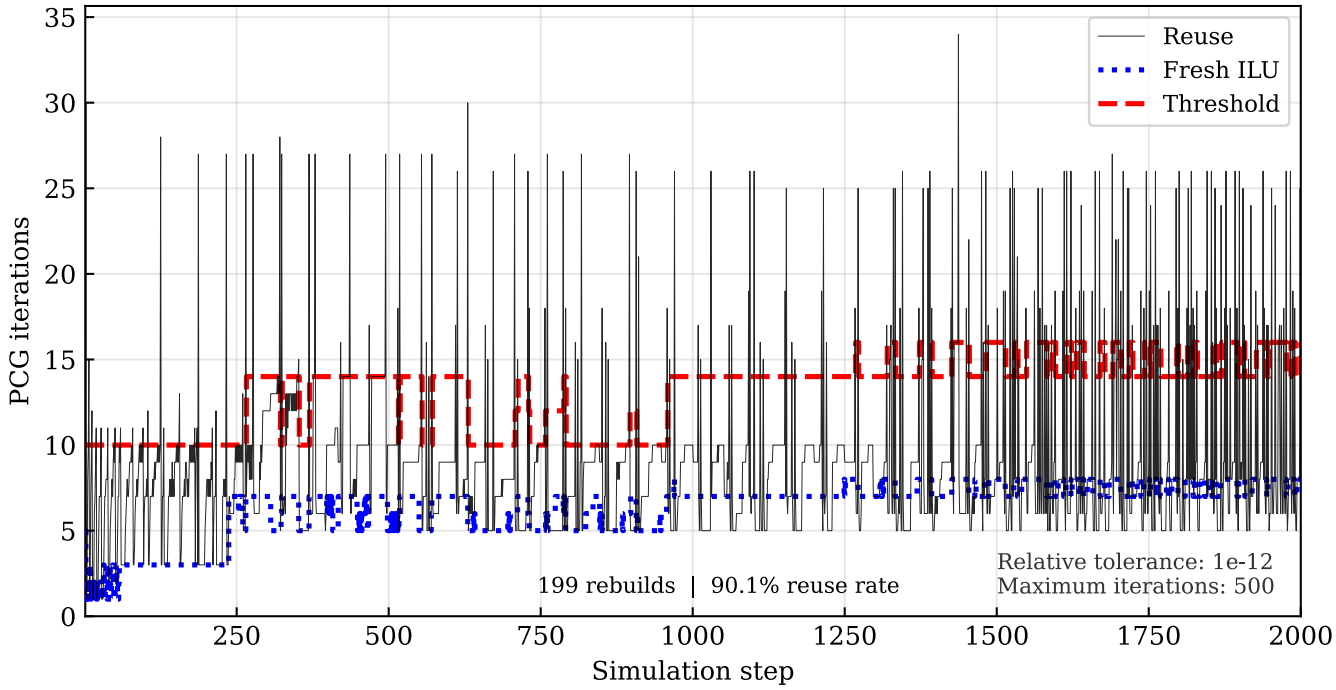


Figure 4.3: Per-step PCG iteration count for the reuse run and the fresh ILU baseline, with the threshold $\max(2k_0, 10)$ overlaid. The reuse run shows a sawtooth pattern with spikes reaching up to 34 iterations before each rebuild, while the baseline is stable in the range 1–8. Of the 2000 steps, 199 trigger a rebuild, giving a reuse rate of 90.1%.

Figure 4.4 quantifies the total iteration overhead. The reuse run accumulates 17 802 PCG iterations against the baseline's 12 758, a 39.5% overhead. The roughly linear growth of both curves indicates that the average per-step iteration cost is approximately constant throughout the run for each method.

More PCG iterations per step does not imply higher wall-clock cost, because each reuse step avoids an ILU factorization. Figure 4.5 shows the cumulative wall-clock time for both runs. The reuse run completes in 1.65 s against the baseline's 1.73 s, a reduction of 4.4%, suggesting that the 1801 avoided ILU factorizations outweigh the 39.5% iteration overhead in this setting.

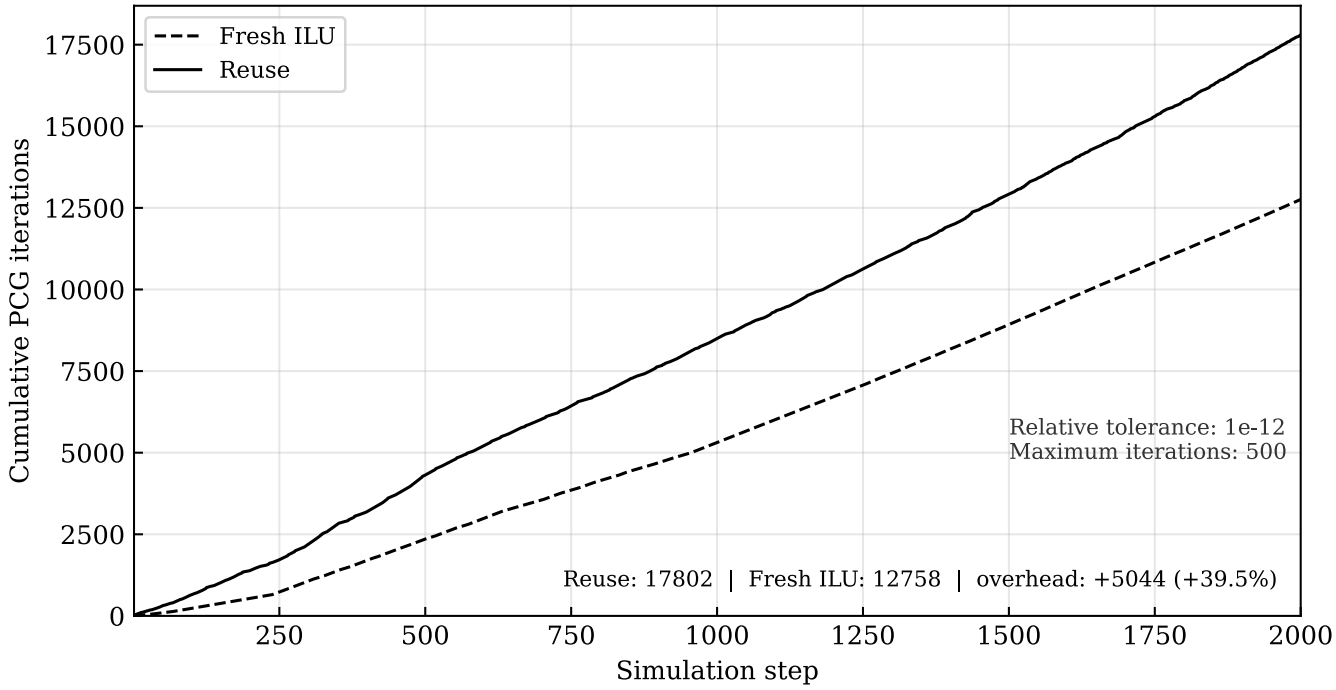


Figure 4.4: Cumulative PCG iterations as a function of simulation step. The reuse run accumulates 39.5% more iterations in total, with both curves growing roughly linearly, indicating approximately constant average per-step cost throughout the run.

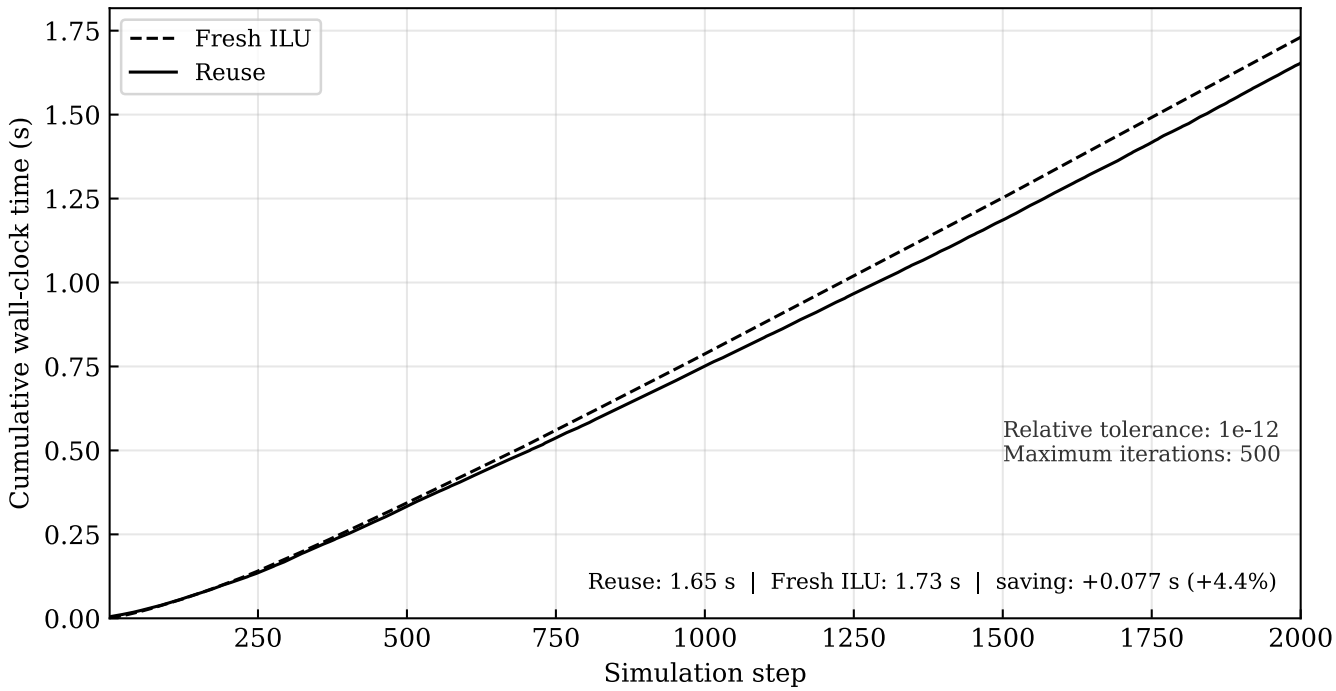


Figure 4.5: Cumulative wall-clock time as a function of simulation step. The reuse run completes in 1.65 s against the baseline's 1.73 s, a reduction of 4.4%, suggesting that the 1801 avoided ILU factorizations outweigh the iteration overhead.

5

Conclusions

This thesis investigated whether a preconditioner M computed on one grid can be reused after adaptive mesh refinement, rather than rebuilt from scratch. The approach constructs mapping operators from the old and new grid geometries and uses them to transfer M to the updated system. Chapter 3 established that this is viable, provided smoothing sweeps are applied to suppress the interpolation errors that the mapping introduces, and Chapter 4 implemented the strategy in FLASH, where it reduced wall-clock time relative to a baseline that rebuilds M at every step.

5.1 Summary

Each grid adaptation renders the old preconditioner M incompatible with the new system matrix. Rebuilding from scratch discards all structural information that M encodes, even though it might still be a relevant approximation of the updated system matrix A. The approach transfers M to the updated system via purely geometric operators, requiring no access to the system matrix or the governing equation, so the reuse strategy is independent of the PDE being solved.

Chapter 3 developed the approach for a 1D model problem. The prolongation operator P and restriction operator R were derived from the cell geometry of the finite volume discretization, without reference to the system matrix or its entries. Applied without smoothing, the mapped preconditioner M_{map}^{-1} introduces high-frequency components in the error that stall convergence. The composite preconditioner introduced in Section 3.3 resolves this by surrounding the application of M_{map}^{-1} with pre- and post-smoothing sweeps that damp these components, showing that the zeroth-order mapping error can be compensated without modifying the mapping operators themselves.

The numerical experiments of Chapter 3 covered grid sizes spanning three orders of magnitude, with the diffusion coefficient and source term varied across multiple configurations. Without preconditioning, the condition number of the discrete system grows as the grid is refined, causing iteration counts to grow accordingly. The composite preconditioner eliminated this growth, producing grid-independent iteration counts across all tested configurations. Section 3.4.4 confirmed that this scaling is robust to variations in the preconditioner, the system, and the solver, establishing that the composite approach does not depend on properties specific to the base case.

The mapping operators also provide a secondary benefit. Mapping the converged old-grid solution onto the new grid and applying smoother sweeps to suppress the high-frequency errors yields a better starting point for the Krylov solver than the standard zero vector. Section 3.4.3 showed that this reduces GMRES iteration counts by one to two for most smoothers. The mapping operators are already assembled for the preconditioner transfer, so no additional construction is required.

Chapter 4 transferred the approach to FLASH. The transition operator T, developed in Section 4.2, generalizes P and R to PARAMESH's block-structured grid hierarchy. Like P and R, T is assembled from grid geometry alone and requires no information about the system matrix or the

governing equation. T handles the general case, including mixed steps where both refinement and derefinement occur simultaneously. An adaptive criterion monitors the iteration count and triggers a rebuild only when the preconditioner quality has degraded past a threshold, so rebuilds are avoided for as long as the iteration count remains acceptable.

The FLASH prototype was tested on a 1D heat diffusion problem over 2000 time steps. With the threshold as configured, 199 steps triggered a rebuild and the remaining 1801 steps reused the anchored preconditioner. The reuse run accumulated 39.5% more iterations than the baseline that always rebuilds, but completed in 1.65 s against the baseline's 1.73 s, a reduction of 4.4%, because the 1801 avoided ILU factorizations outweighed the additional iteration cost. In this prototype the base preconditioner is ILU(0) on a 1D tridiagonal system, where the build cost is modest, yet a net reduction is still achieved. Deferring rebuilds is already worthwhile when the build is cheap, suggesting that the per-rebuild reduction would be larger for more expensive preconditioners. The grid-independent iteration counts established in Chapter 3, however, were not verified in the FLASH prototype, leaving open whether the theoretical scaling carries over to the implementation.

Together, these results establish a proof of concept that geometric transfer operators are sufficient to carry a preconditioner across a refinement event, and that combining them with smoothing sweeps resolves the interpolation error that transfer alone leaves behind. Both results rest on the 1D setting, and the central open question is whether the smoother overhead that suffices in 1D scales to the higher-dimensional problems for which AMR is most relevant.

5.2 Limitations and future work

Extending the approach to two and three spatial dimensions would open the method to realistic multi-dimensional problems. In higher dimensions the zeroth-order mapping error is larger, because a single refinement event in 3D produces eight child cells rather than two. The volume of interpolation error injected per event therefore grows. It is therefore not guaranteed that the smoother configuration that produced grid-independent iteration counts in 1D will do so in 2D or 3D. Those problems also tend to require more expensive preconditioners, so if the mapping approach does generalize, the reduction per avoided rebuild would be expected to grow. However, the reuse strategy requires the anchored preconditioner M and the transition operator T to be retained alongside the current system matrix. In a 3D AMR simulation, the active grid can contain millions of degrees of freedom, and M is a sparse matrix whose storage scales with the number of non-zeros. AMR itself is used precisely because it concentrates resolution only where needed, keeping active cell counts manageable, but carrying an additional sparse matrix of comparable size partially offsets that gain. Whether the storage overhead is acceptable in practice depends on the problem size, the sparsity pattern of M, and the available system memory. This has not been quantified.

The mapping operators derived in Chapter 3 are zeroth-order approximations. Higher-order schemes would reduce the truncation error but require information from neighboring cells, adding implementation complexity that was out of scope for the present work.

The smoother sweep count ν, the relaxation parameter ω, and the rebuild threshold are tunable parameters whose interaction has not been fully explored. Section 3.4.3 showed that ν has a cost-benefit crossover beyond which additional sweeps cost more than they save. The rebuild threshold presents an analogous trade-off between the extra iterations a degraded preconditioner incurs and the cost of a rebuild. A multi-dimensional implementation that addresses the questions raised here would be the natural next step toward establishing the approach as a general-purpose tool for AMR-based solvers.

Bibliography

[1] Tzeferacos, P., Fatenejad, M., Flocke, N., Graziani, C., Gregori, G., Lamb, D. Q., Lee, D., Meinecke, J., Scopatz, A., & Weide, K. (2015). FLASH MHD simulations of experiments that study shock-generated magnetic fields. *High Energy Density Physics*, 17, 24–31. https://doi.org/10.1016/j.hedp.2014.11.003

[2] Carr, A., de Sturler, E., & Gugercin, S. (2021). Preconditioning parametrized linear systems. *SIAM Journal on Scientific Computing*, 43(3), A2242–A2267. https://doi.org/10.1137/20M1331123

[3] Singh, N. P., & Ahuja, K. (2020). Reusing preconditioners in projection based model order reduction algorithms. *IEEE Access*, 8, 133233–133247. https://doi.org/10.1109/ACCESS.2020.3009456

[4] Bramble, J. H., Ewing, R. E., Pasciak, J. E., & Schatz, A. H. (1988). A preconditioning technique for the efficient solution of problems with local grid refinement. *Computer Methods in Applied Mechanics and Engineering*, 67(2), 149–159.

[5] Maes, J., & Oswald, P. (2009). Multilevel finite element preconditioning for $\sqrt{3}$ refinement. *Mathematics of Computation*, 78(268), 1869–1890. https://doi.org/10.1090/S0025-5718-09-02246-7

[6] Trefethen, L. N., & Bau III, D. (1997). *Numerical linear algebra*. Society for Industrial and Applied Mathematics (SIAM).

[7] Saad, Y. (2003). *Iterative methods for sparse linear systems* (2nd ed.). Society for Industrial and Applied Mathematics (SIAM).

[8] Saad, Y., & Schultz, M. H. (1986). GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3), 856–869. https://doi.org/10.1137/0907058

[9] Briggs, W. L., Henson, V. E., & McCormick, S. F. (2000). *A multigrid tutorial* (2nd ed.). Society for Industrial and Applied Mathematics (SIAM).

[10] Berger, M. J., & Oliger, J. (1984). Adaptive mesh refinement for hyperbolic partial differential equations. *Journal of Computational Physics*, 53(3), 484–512. https://doi.org/10.1016/0021-9991(84)90073-1

[11] Löhner, R. (1987). An adaptive finite element scheme for transient problems in CFD. *Computer Methods in Applied Mechanics and Engineering*, 61(3), 323–338. https://doi.org/10.1016/0045-7825(87)90098-3

[12] LeVeque, R. J. (2007). *Finite difference methods for ordinary and partial differential equations: Steady-state and time-dependent problems*. Society for Industrial and Applied Mathematics (SIAM).

[13] MacNeice, P., Olson, K. M., Mobarry, C., de Fainchtein, R., & Packer, C. (2000). PARAMESH: A parallel adaptive mesh refinement community toolkit. *Computer Physics Communications*, 126(3), 330–354. https://doi.org/10.1016/S0010-4655(99)00501-9

[14] Horn, R. A., & Johnson, C. R. (2013). *Matrix analysis* (2nd ed.). Cambridge University Press.

[15] Guyer, J. E., Wheeler, D., & Warren, J. A. (2009). FiPy: Partial differential equations with Python. *Computing in Science & Engineering*, 11(3), 6–15. https://doi.org/10.1109/MCSE.2009.52

[16] Ferreira, J. A., & Grigorieff, R. D. (2006). Supraconvergence and supercloseness of a scheme for elliptic equations on nonuniform grids. *Numerical Functional Analysis and Optimization*, 27(5–6), 539–564. https://doi.org/10.1080/01630560600796485

[17] Falgout, R. D., & Yang, U. M. (2002). *hypre*: A library of high performance preconditioners. In P. M. A. Sloot et al. (Eds.), *Computational Science — ICCS 2002*, LNCS 2331, pp. 632–641. Springer-Verlag.

[18] Virtanen, P., Gommers, R., Oliphant, T. E., et al. (2020). SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods*, 17(3), 261–272. https://doi.org/10.1038/s41592-019-0686-2

[19] Balay, S., Abhyankar, S., Adams, M. F., et al. (2021). *PETSc users manual*. Argonne National Laboratory.

[20] Dalcin, L. D., Paz, R. R., Kler, P. A., & Cosimo, A. (2011). Parallel distributed computing using Python. *Advances in Water Resources*, 34(9), 1124–1139. https://doi.org/10.1016/j.advwatres.2011.04.013

[21] Golub, G. H., & Van Loan, C. F. (2013). *Matrix computations* (4th ed.). Johns Hopkins University Press.

A

Derivations

A.1 Discretization of the 1D diffusion equation using the finite volume method

The discretization of the one-dimensional steady-state diffusion equation

$$-\frac{d}{dx}\left(D(x)\frac{du(x)}{dx}\right) = f(x), \tag{A.1}$$

on a non-uniform grid relies on the principle of local conservation. As illustrated in Figure 3.1, the domain is divided into N discrete control volumes, giving $N+1$ faces at integer indices x_i and N cell centers at fractional indices $x_{i+1/2}$. The width of a control volume is denoted by $h_{i+1/2} = x_{i+1} - x_i$, and the physical distance between adjacent cell centers is denoted by $h_i = x_{i+1/2} - x_{i-1/2}$.

Subscript notation refers to quantities belonging to the discrete system, such as the cell-center values $u_{i+1/2}$ and the face coefficients D_i and $h_{i+1/2}$. Continuous quantities evaluated at grid locations but not part of the discrete system, such as the exact face derivative $u'(x_i)$, retain explicit function arguments.

We begin by integrating the governing equation over a control volume extending from the left face x_i to the right face x_{i+1}

$$-\int_{x_i}^{x_{i+1}} \frac{d}{dx}\left(D(x)\frac{du(x)}{dx}\right) dx = \int_{x_i}^{x_{i+1}} f(x)\, dx.$$

Evaluating the fluxes across the control volume boundaries yields

$$-\left[D(x_{i+1})u'(x_{i+1}) - D(x_i)u'(x_i)\right] = \int_{x_i}^{x_{i+1}} f(x)\, dx. \tag{A.2}$$

This balance enforces local conservation. The difference between the flux entering the left face and the flux leaving the right face equals the total internal generation.

To render this numerically solvable, we approximate the source term integral by assuming the cell-centered value $f_{i+1/2}$ remains constant across the cell width $h_{i+1/2}$. Although the true source function may vary continuously, evaluating the integral as a simple rectangular area is justified because the function's variation within a single cell becomes negligible as the grid spacing decreases. The continuous derivatives at the faces are then approximated using a central difference scheme, relying on the adjacent cell-centered nodal values. A formal analysis of the errors introduced by both

of these approximations is deferred to the subsequent section.

$$\int_{x_i}^{x_{i+1}} f(x)\,dx \approx f_{i+1/2}h_{i+1/2}, \tag{A.3a}$$

$$u'(x_i) \approx \frac{u_{i+1/2} - u_{i-1/2}}{h_i}, \tag{A.3b}$$

$$u'(x_{i+1}) \approx \frac{u_{i+3/2} - u_{i+1/2}}{h_{i+1}}. \tag{A.3c}$$

Substituting these finite difference approximations back into the integrated flux balance and letting $D(x_i) = D_i$ represent the material property evaluated at the face results in the discrete conservation equation for a general interior cell

$$\frac{D_i}{h_i}(u_{i+1/2} - u_{i-1/2}) - \frac{D_{i+1}}{h_{i+1}}(u_{i+3/2} - u_{i+1/2}) = f_{i+1/2}h_{i+1/2}.$$

Expanding and factoring out the unknown cell-center variables isolates the local numerical stencil associated with the nodes $u_{i-1/2}$, $u_{i+1/2}$, and $u_{i+3/2}$

$$-\frac{D_i}{h_i}u_{i-1/2} + \left(\frac{D_i}{h_i} + \frac{D_{i+1}}{h_{i+1}}\right)u_{i+1/2} - \frac{D_{i+1}}{h_{i+1}}u_{i+3/2} = f_{i+1/2}h_{i+1/2}. \tag{A.4}$$

Solving this system yields the stencil coefficients from (3.2),

$$\alpha_i \coloneqq -\frac{D_i}{h_i}, \qquad \beta_i \coloneqq \frac{D_i}{h_i} + \frac{D_{i+1}}{h_{i+1}}, \qquad \gamma_i \coloneqq -\frac{D_{i+1}}{h_{i+1}},$$

where $D_i = D(x_i)$ is the diffusion coefficient evaluated at face x_i. To construct the global linear system, Dirichlet boundary conditions $u_0 = f_L$ and $u_N = f_R$ must be incorporated. For the first cell centered at $x_{1/2}$, the distance to the left boundary face is precisely half the local cell width, $h_{1/2}/2$. Modifying the derivative approximation to reflect this truncated spacing yields the boundary flux. A similar half-cell adjustment applies to the right boundary cell centered at $x_{N-1/2}$. Grouping the boundary values and interior unknowns produces the specialized boundary cell equations

$$-\frac{2D_0}{h_{1/2}}u_0 + \left(\frac{2D_0}{h_{1/2}} + \frac{D_1}{h_1}\right)u_{1/2} - \frac{D_1}{h_1}u_{3/2} = f_{1/2}h_{1/2}, \tag{A.5a}$$

$$-\frac{D_{N-1}}{h_{N-1}}u_{N-3/2} + \left(\frac{D_{N-1}}{h_{N-1}} + \frac{2D_N}{h_{N-1/2}}\right)u_{N-1/2} - \frac{2D_N}{h_{N-1/2}}u_N = f_{N-1/2}h_{N-1/2}. \tag{A.5b}$$

Using the generalized coefficients α_i, β_i, and γ_i already defined for the interior nodes, and extracting the corresponding boundary coefficients from (A.5a) and (A.5b), the entire discretized domain can be initially expressed as an $(N+2) \times (N+2)$ linear system. Enforcing the exact Dirichlet boundary

conditions in the first and last rows yields the full system

$$\begin{bmatrix} 1 & & & & & & & & & \\ \alpha_0 & \beta_0 & \gamma_0 & & & & & & & \\ & \alpha_1 & \beta_1 & \gamma_1 & & & & & & \\ & & \ddots & \ddots & \ddots & & & & & \\ & & & \alpha_{i-1} & \beta_{i-1} & \gamma_{i-1} & & & & \\ & & & & \alpha_i & \beta_i & \gamma_i & & & \\ & & & & & \ddots & \ddots & \ddots & & \\ & & & & & & \alpha_{N-2} & \beta_{N-2} & \gamma_{N-2} & \\ & & & & & & & \alpha_{N-1} & \beta_{N-1} & \gamma_{N-1} \\ & & & & & & & & & 1 \end{bmatrix} \begin{bmatrix} u_0 \\ u_{1/2} \\ u_{3/2} \\ \vdots \\ u_{i-1/2} \\ u_{i+1/2} \\ \vdots \\ u_{N-3/2} \\ u_{N-1/2} \\ u_N \end{bmatrix} = \begin{bmatrix} f_L \\ f_{1/2}h_{1/2} \\ f_{3/2}h_{3/2} \\ \vdots \\ f_{i-1/2}h_{i-1/2} \\ f_{i+1/2}h_{i+1/2} \\ \vdots \\ f_{N-3/2}h_{N-3/2} \\ f_{N-1/2}h_{N-1/2} \\ f_R \end{bmatrix}. \quad \text{(A.6)}$$

While the extended system fully captures the physics of the domain, retaining the explicit boundary rows introduces numerical scaling issues. The unit diagonal entries for the boundary conditions can differ in magnitude from the interior coefficients that scale inversely with the grid spacing. This disparity inflates the condition number of the matrix, which degrades the convergence speed of iterative solvers. Because u_0 and u_N are already known constants, treating them as computational unknowns is also unnecessary and artificially increases the system dimension. It is standard practice to substitute these boundary values directly into the adjacent cell equations and transfer the constant terms, $f_L\alpha_0$ and $f_R\gamma_{N-1}$, to the source vector on the right side of the equation. Eliminating these trivial equations reduces the system to the N unknown interior nodes. The resulting tridiagonal finite volume matrix is symmetric positive definite for any $D(x) > 0$ [12, p. 36]

$$\begin{bmatrix} \beta_0 & \gamma_0 & & & & & & \\ \alpha_1 & \beta_1 & \gamma_1 & & & & & \\ & & \ddots & & & & & \\ & & \alpha_{i-1} & \beta_{i-1} & \gamma_{i-1} & & & \\ & & & \alpha_i & \beta_i & \gamma_i & & \\ & & & & & \ddots & & \\ & & & & & \alpha_{N-2} & \beta_{N-2} & \gamma_{N-2} \\ & & & & & & \alpha_{N-1} & \beta_{N-1} \end{bmatrix} \begin{bmatrix} u_{1/2} \\ u_{3/2} \\ \vdots \\ u_{i-1/2} \\ u_{i+1/2} \\ \vdots \\ u_{N-3/2} \\ u_{N-1/2} \end{bmatrix} = \begin{bmatrix} f_{1/2}h_{1/2} - f_L\alpha_0 \\ f_{3/2}h_{3/2} \\ \vdots \\ f_{i-1/2}h_{i-1/2} \\ f_{i+1/2}h_{i+1/2} \\ \vdots \\ f_{N-3/2}h_{N-3/2} \\ f_{N-1/2}h_{N-1/2} - f_R\gamma_{N-1} \end{bmatrix}. \quad \text{(A.7)}$$

A.1.1 Error analysis of the finite volume discretization

To evaluate the accuracy of the finite volume discretization, we must analyze the error introduced by the numerical scheme. As established in (A.2), the conservation equation is constructed as a balance of continuous fluxes across the cell faces. Consequently, the total numerical error of a given cell is dictated by the accuracy of the discrete approximations at these individual faces. Our analysis must therefore begin by defining the error introduced at these cell faces.

To find the error introduced by the discretization, we temporarily assume a constant diffusion coefficient $D = 1$. This reduces the governing equation (A.1) to $-u''(x) = f(x)$. We will reintroduce a variable diffusion coefficient $D(x)$ after the fundamental error of the discrete gradient approximations has been established.

The discrete equation for an interior cell is extracted from the global linear system (A.7). For our simplified case, the balance of fluxes equals the integrated source term $f_{i+1/2}h_{i+1/2}$. Since it

represents a volume integral, it cannot be directly compared to the exact point-wise continuous equation $-u''(x_{i+1/2}) = f(x_{i+1/2})$. We must divide it by the cell width $h_{i+1/2}$ to match the continuous operator

$$-\frac{1}{h_{i+1/2}}\left(\frac{u_{i+3/2}-u_{i+1/2}}{h_{i+1}}-\frac{u_{i+1/2}-u_{i-1/2}}{h_i}\right)\approx f_{i+1/2}. \tag{A.8}$$

The fractions inside the parentheses represent the gradient approximations at the right face x_{i+1} and the left face x_i. To construct the total error of the cell we evaluate these gradient approximations individually, starting with the left face x_i which is given by (A.3b)

$$u'(x_i)\approx\frac{u_{i+1/2}-u_{i-1/2}}{h_i}.$$

Taylor expanding $u_{i-1/2}$ and $u_{i+1/2}$ about x_i gives

$$u_{i-1/2}=u(x_i)-\frac{h_{i-1/2}}{2}u'(x_i)+\frac{1}{2!}\left(\frac{h_{i-1/2}}{2}\right)^2u''(x_i)-\frac{1}{3!}\left(\frac{h_{i-1/2}}{2}\right)^3u'''(x_i)+\mathcal{O}(h_{i-1/2}^4),$$
$$u_{i+1/2}=u(x_i)+\frac{h_{i+1/2}}{2}u'(x_i)+\frac{1}{2!}\left(\frac{h_{i+1/2}}{2}\right)^2u''(x_i)+\frac{1}{3!}\left(\frac{h_{i+1/2}}{2}\right)^3u'''(x_i)+\mathcal{O}(h_{i+1/2}^4).$$

Subtracting the left expansion from the right and substituting the nodal distance $h_i=\frac{h_{i+1/2}+h_{i-1/2}}{2}$ illustrated in Figure 3.1 yields

$$\frac{u_{i+1/2}-u_{i-1/2}}{h_i}=u'(x_i)+\frac{h_{i+1/2}-h_{i-1/2}}{4}u''(x_i)+\mathcal{O}(h_i^2).$$

This result demonstrates that the gradient approximation at x_i is first-order accurate. We define the numerical gradient error ϵ_i at x_i

$$\epsilon_i\coloneqq\frac{h_{i+1/2}-h_{i-1/2}}{4}u''(x_i). \tag{A.10}$$

It is proportional to the difference in adjacent cell widths $h_{i+1/2}-h_{i-1/2}$. The gradient approximation only recovers second-order accuracy when the adjacent cells are of equal width $h_{i+1/2}=h_{i-1/2}$, which causes the first-order error term to vanish.

Applying the Taylor expansion procedure to the right face x_{i+1} given by (A.3c) yields an analogous gradient error term ϵ_{i+1}. We substitute these expanded gradient approximations back into the normalized discrete equation (A.8)

$$-\frac{1}{h_{i+1/2}}\left[(u'(x_{i+1})+\epsilon_{i+1})-(u'(x_i)+\epsilon_i)\right]\approx f_{i+1/2}.$$

Rearranging the terms separates the exact continuous gradients from the numerical gradient errors

$$-\frac{u'(x_{i+1})-u'(x_i)}{h_{i+1/2}}-\frac{\epsilon_{i+1}-\epsilon_i}{h_{i+1/2}}\approx f_{i+1/2}.$$

Taylor expanding the exact continuous gradients $u'(x_{i+1})$ and $u'(x_i)$ about the cell center $x_{i+1/2}$ simplifies the first quotient to $u''(x_{i+1/2})+\mathcal{O}(h_{i+1/2}^2)$. Since the continuous solution satisfies the

simplified governing equation $-u''(x_{i+1/2}) = f(x_{i+1/2})$, these terms cancel. The remaining terms define the local truncation error (LTE) of the interior cell

$$\mathrm{LTE}_{i+1/2} \coloneqq -\frac{\epsilon_{i+1} - \epsilon_i}{h_{i+1/2}}. \tag{A.11}$$

We now reintroduce the variable diffusion coefficient $D(x)$. The flux across a cell face is $J(x) = D(x)u'(x)$. The numerical flux J_i at the left face x_i is then constructed by multiplying D_i by the gradient approximation derived previously

$$J_i = D_i \left[u'(x_i) + \epsilon_i\right].$$

Applying this logic to the right face x_{i+1} yields the numerical flux $J_{i+1} = D_{i+1}\left[u'(x_{i+1}) + \epsilon_{i+1}\right]$. We substitute these two numerical fluxes back into the normalized discrete equation (A.8) in place of the simplified gradients. Applying the same cancellation of the governing equation $-J'(x_{i+1/2}) = f(x_{i+1/2})$ produces the final local truncation error for the 1D diffusion equation on a non-uniform grid

$$\mathrm{LTE}_{i+1/2} = -\frac{D_{i+1}\epsilon_{i+1} - D_i\epsilon_i}{h_{i+1/2}}. \tag{A.12}$$

Equation (A.12) reveals a fundamental inconsistency when evaluated on an arbitrary grid. As established in (A.10), the numerical gradient error at the face is proportional to the difference between adjacent cell widths. On an arbitrary grid, this difference is of the same magnitude as the local grid spacing itself, meaning the face error scales as $\epsilon_i = \mathcal{O}(h_i)$. Because both the numerator and the denominator of the local truncation error (A.12) scale linearly with the local cell size, dividing them yields an $\mathrm{LTE}_{i+1/2}$ of $\mathcal{O}(1)$. In classical finite difference theory, an $\mathcal{O}(1)$ local truncation error implies the discrete equation fails to converge to the exact continuous differential equation as the grid is refined [12, p. 20].

The finite volume method overcomes this point-wise failure through global flux conservation. The discrete numerical flux evaluated at a shared cell face x_{i+1} represents the exact same physical quantity leaving cell i and entering cell $i+1$. To observe how this affects the global solution, we must revert the normalized equation back to its volume-integrated form by multiplying by the cell width $h_{i+1/2}$. Summing these integrated errors over all interior cells creates a telescoping series where all internal numerical gradient errors cancel exactly

$$\sum_{i=0}^{N-1} \mathrm{LTE}_{i+1/2} h_{i+1/2} = -\sum_{i=0}^{N-1} \Big(D_{i+1}\epsilon_{i+1} - D_i\epsilon_i\Big) = -\Big(D_N\epsilon_N - D_0\epsilon_0\Big).$$

Because the local cell width terms acting as the denominator are canceled out by the volume integration, the global error is bounded by the raw numerical gradient errors at the boundaries, ϵ_0 and ϵ_N. Since these raw numerical gradient errors scale as $\mathcal{O}(h_{\max})$, where $h_{\max}$ is the global maximum cell width, the global formulation rescues the simulation from $\mathcal{O}(1)$ divergence, guaranteeing at least first-order global convergence.

This global conservation links the accuracy of the method to the grid generation technique. If the grid is generated with a smoothly varying expansion function, the individual cell widths are related by a continuous derivative. On such a smooth grid, the difference between adjacent cell widths becomes constrained, scaling with the square of the local cell size, $(h_{i+1/2} - h_{i-1/2}) \sim \mathcal{O}(h_{i+1/2}^2)$. Substituting this geometric constraint into (A.10) alters the scaling of the face error, dropping it to $\epsilon_i \sim \mathcal{O}(h_i^2)$. Because the global error inherits the exact order of these numerical gradient errors, the domain-wide

solution upgrades to second-order accuracy, $\mathcal{O}(h_{\max}^2)$. This phenomenon, where the global solution converges at a higher rate than the local truncation error dictates, is known as supraconvergence [16, p. 540].

Finally, in addition to the spatial derivatives, we must account for the error introduced by assuming the source term remains constant across the control volume. Taylor expanding the continuous source function $f(x)$ about the cell center $x_{i+1/2}$ and integrating over the cell width yields

$$\begin{aligned}\int_{x_i}^{x_{i+1}} f(x)\,dx &= \int_{x_i}^{x_{i+1}} \left[f(x_{i+1/2}) + (x - x_{i+1/2}) f'(x_{i+1/2}) + \frac{(x - x_{i+1/2})^2}{2} f''(x_{i+1/2}) + \mathcal{O}(h^3) \right] dx \\ &= f(x_{i+1/2}) h_{i+1/2} + \frac{h_{i+1/2}^3}{24} f''(x_{i+1/2}) + \mathcal{O}(h^5).\end{aligned}$$

Dividing by the cell width $h_{i+1/2}$ to match the reconstructed continuous operator isolates the truncation error associated with the source term approximation

$$\frac{1}{h_{i+1/2}} \int_{x_i}^{x_{i+1}} f(x)\,dx = f(x_{i+1/2}) + \underbrace{\frac{h_{i+1/2}^2}{24} f''(x_{i+1/2})}_{\epsilon_{\text{source}}} + \mathcal{O}(h^4).$$

The error introduced by treating the source term as a simple rectangular area is second-order accurate, $\mathcal{O}(h^2)$. This validates the decision to use this simple approximation. Because a numerical scheme is only as accurate as its weakest link, and the global convergence rate dictated by the supraconvergent flux terms is $\mathcal{O}(h^2)$, employing a more computationally expensive, higher-order integration method for the source term would not improve the overall accuracy of the solver.

A.2 SPD property of the composite preconditioner

This section proves that the composite preconditioner of Section 3.3,

$$M_{\text{comp}}^{-1} = M_{\text{pre}}^{-1} + M_{\text{map}}^{-1}(I - AM_{\text{pre}}^{-1}) + M_{\text{post}}^{-1}(I - AM_{\text{map}}^{-1})(I - AM_{\text{pre}}^{-1}),$$

is positive definite under the following four assumptions.

(i) A is SPD.

(ii) M_{map}^{-1} is positive definite.

(iii) $M_{\text{post}} = M_{\text{pre}}^T$.

(iv) The splitting $A = M_{\text{pre}} - N_{\text{pre}}$ satisfies $y^T(M_{\text{pre}}^T + N_{\text{pre}})y \geq 0$ for all $y \in \mathbb{R}^N$.

Symmetry of M_{comp}^{-1} follows from assumptions (i)–(iii) and is established in Section 3.3, so it remains to show $v^T M_{\text{comp}}^{-1} v > 0$ for every non-zero v. The variables u, z, x, and y used below are local to this proof.

Rearranging $G_{\text{comp}} = I - M_{\text{comp}}^{-1}A$ gives $M_{\text{comp}}^{-1} = (I - G_{\text{comp}})A^{-1}$. Substituting this and introducing $u = A^{-1}v$ with $v^T = u^T A$ (from the symmetry of A) reduces the target inner product to

$$v^T M_{\text{comp}}^{-1} v = u^T A u - u^T A G_{\text{comp}} u. \tag{A.13}$$

Since A is SPD and $v \neq 0$, it follows that $u \neq 0$. To simplify the second term, we use assumption (iii) to write $G_{\text{post}} = I - (M_{\text{pre}}^{-1})^T A$. Taking the transpose of $G_{\text{pre}} = I - M_{\text{pre}}^{-1} A$ and using symmetry of A gives $G_{\text{pre}}^T = I - A(M_{\text{pre}}^{-1})^T$. Comparing these two expressions establishes the identity $G_{\text{pre}}^T A = A G_{\text{post}}$. Substituting the factorization $G_{\text{comp}} = G_{\text{post}} G_{\text{map}} G_{\text{pre}}$ into (A.13) and applying this identity to replace $A G_{\text{post}}$ with $G_{\text{pre}}^T A$ gives

$$u^T A G_{\text{comp}} u = u^T G_{\text{pre}}^T A G_{\text{map}} G_{\text{pre}} u = z^T A G_{\text{map}} z, \tag{A.14}$$

where $z = G_{\text{pre}} u$ is the error vector after pre-smoothing. Substituting back into (A.13) and then expanding $G_{\text{map}} = I - M_{\text{map}}^{-1} A$ with $x = Az$ gives the decomposition

$$v^T M_{\text{comp}}^{-1} v = (u^T A u - z^T A z) + x^T M_{\text{map}}^{-1} x. \tag{A.15}$$

By assumption (ii), the second term is strictly positive when $x \neq 0$ and zero otherwise.

It remains to bound the first term. Substituting $z = (I - M_{\text{pre}}^{-1} A)u$ into $z^T A z$, expanding, and subtracting from $u^T A u$ yields

$$u^T A u - z^T A z = u^T A M_{\text{pre}}^{-1} A u + u^T A M_{\text{pre}}^{-T} A u - u^T A M_{\text{pre}}^{-T} A M_{\text{pre}}^{-1} A u, \tag{A.16}$$

where $M_{\text{pre}}^{-T} = (M_{\text{pre}}^{-1})^T$. Introducing $y = M_{\text{pre}}^{-1} A u$, so that $Au = M_{\text{pre}} y$ and $y^T = u^T A M_{\text{pre}}^{-T}$, each term on the right simplifies. Specifically, $u^T A M_{\text{pre}}^{-1} A u = y^T M_{\text{pre}}^T y$, $u^T A M_{\text{pre}}^{-T} A u = y^T M_{\text{pre}} y$, and $u^T A M_{\text{pre}}^{-T} A M_{\text{pre}}^{-1} A u = y^T A y$. The difference therefore factors as

$$u^T A u - z^T A z = y^T (M_{\text{pre}}^T + M_{\text{pre}} - A)\, y = y^T (M_{\text{pre}}^T + N_{\text{pre}})\, y, \tag{A.17}$$

where the second equality uses $N_{\text{pre}} = M_{\text{pre}} - A$. By assumption (iv) this is non-negative, confirming that both terms in (A.15) are non-negative. Combining (A.15) and (A.17) gives the final decomposition

$$v^T M_{\text{comp}}^{-1} v = y^T (M_{\text{pre}}^T + N_{\text{pre}})\, y + x^T M_{\text{map}}^{-1} x. \tag{A.18}$$

When $z = 0$, the second term vanishes, and from (A.17) the first term equals $u^T A u > 0$, since A is SPD and $u \neq 0$. When $z \neq 0$, A being invertible gives $x = Az \neq 0$, so by assumption (ii) the second term is strictly positive. In both cases $v^T M_{\text{comp}}^{-1} v > 0$, which completes the proof.

Assumption (iv) can be verified for any specific smoother by substituting the corresponding splitting matrices M_{pre} and $N_{\text{pre}} = M_{\text{pre}} - A$ from Section 2.2.1 into $M_{\text{pre}}^T + N_{\text{pre}}$ and checking positive semidefiniteness directly.

B

Numerical parameters for figures

B.1 Stationary method visualization parameters

The residual trajectories illustrated in Section 2.2.1 (Figures 2.1, 2.2, and 2.3) were generated by simulating the respective stationary iterative methods on a 2×2 linear system $Au = f$. The system matrix is strictly diagonally dominant but non-symmetric,

$$A = \begin{bmatrix} 2.0 & -1.0 \\ 1.0 & 2.0 \end{bmatrix}. \tag{B.1}$$

The right-hand side vector and initial guess are

$$f = \begin{bmatrix} 1.0 \\ 5.0 \end{bmatrix}, \qquad u^{(0)} = \begin{bmatrix} -5.0 \\ -5.0 \end{bmatrix}. \tag{B.2}$$

The iteration is terminated when the 2-norm of the residual falls below the threshold $\|r\|_2 < 1.0$. For the successive over-relaxation (SOR) trajectories, the relaxation parameters were set to $\omega = 0.9$ (under-relaxation) and $\omega = 1.1$ (over-relaxation).

B.2 Eigenvalue spectrum and residual polynomial parameters

The eigenvalue spectrum and residual polynomial illustrated in Section 2.3.1 (Figure 2.5) were generated using a 2D Poisson problem on a 15×15 uniform grid, giving 225 degrees of freedom. The operator Q is the 15×15 tridiagonal second-difference matrix

$$Q = \begin{bmatrix} 2 & -1 & & \\ -1 & 2 & -1 & \\ & \ddots & \ddots & \ddots \\ & & -1 & 2 \end{bmatrix},$$

and the system matrix $A \in \mathbb{R}^{225 \times 225}$ has the block tridiagonal structure

$$A = \begin{bmatrix} Q + 2I & -I & & \\ -I & Q + 2I & -I & \\ & \ddots & \ddots & \ddots \\ & & -I & Q + 2I \end{bmatrix},$$

where each block is 15×15 and I is the 15×15 identity matrix. The preconditioned system uses an incomplete LU factorization with fill factor 5.3 as the preconditioner M.

The residual polynomial $p_3(\lambda) = 1 + c_1\lambda + c_2\lambda^2 + c_3\lambda^3$ is constructed so that its constant term satisfies $p_3(0) = 1$ by construction. The remaining coefficients $c = [c_1, c_2, c_3]^T$ are found by solving the least-squares problem

$$\min_{c \in \mathbb{R}^3} \|\mathbf{1} + Vc\|_2^2,$$

where $\mathbf{1} \in \mathbb{R}^{225}$ is a vector of ones and $V \in \mathbb{R}^{225\times 3}$ is the Vandermonde-like matrix with columns $[\lambda_1^j, \ldots, \lambda_{225}^j]^T$ for $j = 1, 2, 3$, where λ_i are the exact eigenvalues of the respective system.

The iteration counts of 54 (unpreconditioned) and 11 (preconditioned) were obtained by applying GMRES to the same system with a relative residual tolerance of 10^{-10}, initial guess $u^{(0)} = 0$, and a fixed random right-hand side f drawn from a standard normal distribution and normalized to unit norm (NumPy seed 42).

DEPARTMENT OF PHYSICS
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY