Fast Solvers for Elliptic Partial Differential Equations Based on Spectral and High-Order Methods

HARVARD UNIVERSITY

Graduate School of Arts and Sciences

DISSERTATION ACCEPTANCE CERTIFICATE

The undersigned, appointed by the

Harvard John A. Paulson School of Engineering and Applied Sciences
have examined a dissertation entitled:

"Fast Solvers for Elliptic Partial Differential Equations Based on Spectral and High-Order Methods"

presented by: Daniel Frank Fortunato

Signature _____
Typed name: Professor C. Rycroft

Signature _____
Typed name: Professor A. Townsend

Signature _____
Typed name: Professor E. Kaxiras

Signature _____
Typed name: Dr. R. Saye

August 31, 2020

Fast Solvers for Elliptic Partial Differential Equations Based on Spectral and High-Order Methods

by

Daniel Frank Fortunato

A dissertation submitted in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

APPLIED MATHEMATICS

in the

SCHOOL OF ENGINEERING AND APPLIED SCIENCES

at

HARVARD UNIVERSITY

accepted on the recommendation of

Prof. Dr. Chris H. Rycroft, advisor
Prof. Dr. Alex Townsend, advisor
Prof. Dr. Efthimios Kaxiras
Dr. Robert Saye

August 2020

Dissertation Advisor: Chris Rycroft, Alex Townsend Daniel Frank Fortunato

Fast Solvers for Elliptic Partial Differential Equations Based on Spectral and High-Order Methods

Abstract

This thesis develops numerical methods for the solution of elliptic partial differential equations (PDEs), all focused on delivering high-order accuracy with low computational complexity. Three algorithms for solving elliptic PDEs are presented: a fast direct solver for a global spectral method on simple two- and three-dimensional domains; a fast direct solver for a spectral element method in two dimensions; and a multigrid solver for a discontinuous Galerkin method in two and three dimensions.

In Chapter 2, an optimal complexity, spectrally-accurate method is presented for the solution of Poisson's equation on a square, solid cylinder, solid sphere, and cube. The method employs a carefully chosen spectral basis to discretize Poisson's equation as a Sylvester equation with pentadiagonal matrices, which is solved using the alternating direction implicit (ADI) method. A separated spectra property in our discretizations is exploited to allow the ADI method to be used as a direct solver.

In Chapter 3, we introduce a sparse spectral element method in two dimensions based on the ultraspherical spectral method, and a corresponding fast direct solver based on the hierarchical Poincaré–Steklov scheme for solving second-order linear PDEs on polygonal domains with unstructured quadrilateral or triangular meshes. The spectral element method achieves an overall computational complexity of $\mathcal{O}(p^4/h^3)$ for mesh size h and polynomial order p in 2D, enabling hp-adaptivity to be efficiently performed. We develop an open-source software system, `ultraSEM`, for flexible, user-friendly spectral element computations in MATLAB.

In Chapter 4, an efficient hp-multigrid scheme is presented for local discontinuous Galerkin (LDG) discretizations of elliptic problems, formulated around the idea of separately coarsening the underlying discrete gradient and divergence operators. We show that traditional multigrid coarsening of the primal formulation leads to poor and suboptimal multigrid performance, whereas coarsening of the flux formulation leads to essentially optimal convergence and is equivalent to a purely geometric multigrid method. We show that good multigrid convergence rates are achieved in a variety of numerical tests on 2D and 3D uniform and adaptive Cartesian grids, as well as for curved domains using implicitly defined meshes and for multi-phase elliptic interface problems with complex geometry.

Contents

To Mom & Dad

Acknowledgments

First, I must thank my advisors, Chris Rycroft and Alex Townsend. Chris, thank you for bringing such happiness to everything mathematical. My days were always made better by a piece of British humor, a Mac rumor, or the sight of a colored tee. Your dedication to the growth of numerics at Harvard is noble—thank you for making a space for computational research at SEAS. I am grateful for our summer trips to Berkeley and for your mentorship throughout my graduate career. Alex, getting to work with you has been an honor that wholly transformed my graduate school experience. Your unrelenting optimism and excitement for applied mathematics are inspiring, and your creativity contagious. I am grateful to have had a front-row seat. I look back fondly on our conference trips together as well as my trips to Cornell. Your guidance has truly been a gift and I am lucky to call you a mentor and a trusted friend.

My family has been a pillar of support for me during my time in graduate school. To my mom and dad, Marie and Frank: thank you for always being there for me. Your love has helped me through the toughest times and is an unfathomable source of happiness in my life. To my sister, Marisa, and brother-in-law, Kevin: thanks for the fun memories and for your thoughtful advice. To Ellen: thank you for your unwavering support and infinite kindness.

My friends have been a constant source of encouragement throughout the last five years. To my bandmates in the American Symphony of Soul: Alek, Alex, Andrew, Erich, Kristen, Jackson, Chris, Willie, Mary—getting to play music with you all is a joy and has kept me sane throughout graduate school. I'm lucky to have you as friends. To Jordan and Shruti: thank you for your innumerable sledges, pranks, and late-night hangs. I'll fondly miss our trivia team. To Jon, Carlos, Mark, Nico, Doug, Mike, and Ben: thank you for your lifelong friendships.

I am fortunate to have had many wonderful colleagues and mentors throughout my time in graduate school. Robert Saye has been an endless source of knowledge on high-order methods and high-performance C++, and has been a wonderful collaborator. I'm lucky to have experienced Nick Hale's MATLAB wizardry and deep expertise in spectral methods firsthand, and working with him has been a pleasure. Nick Trefethen's enthusiasm for numerical analysis and its communication has been infectious and inspires me daily. I've greatly benefitted from friendships and conversations with countless others, including Sheehan Olver, Alex Barnett, Rasmus Tamstorf, Keaton Burns, Heather Wilber, Marc Gilles, Andrew Horning, Ben Zhang, Ricardo Baptista, Thomas Fai, Chen-Hung Wu, Eder Medina, Nick Derr, and Luna Lin.

Finally, to my thesis committee—Chris, Alex, Robert, and Efthimios Kaxiras—thank you for taking time to read this thesis and for your detailed feedback.

My research has been generously funded by the Department of Defense through the National Defense Science & Engineering Fellowship and by National Science Foundation grants 1645445 and 1818757. This research was also supported in part by the Applied Mathematics Program of the U.S. Department of Energy Office of Advanced Scientific Computing Research under contract number DE-AC02-05CH11231. Some computations used resources of the National Energy Research Scientific Computing Center, a Department of Energy Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under contract number DE-AC02-05CH11231.

Chapter 1

Introduction

Physical simulation is an important part of modern science. In addition to theory and experiment, it has emerged as a third pillar for scientific inquiry. The ability to run experiments, test hypotheses, and validate data on a computer has quickened the pace of discovery and prototyping in academia and industry over the last half-century, and today computation plays a fundamental role in nearly all areas of science and engineering.

Underpinning the computation revolution is the application of mathematical ideas to the development of numerical algorithms. In tandem with the decades-long advancement in computing power due to Moore's law [120], strides in numerical analysis—realized as efficient and accurate algorithms to solve the problems of continuous mathematics—have enabled computational science to flourish [145]. The focus of this thesis is on the development of efficient algorithms for the high-order accurate numerical solution of elliptic partial differential equations (PDEs).

1.1 Elliptic PDEs

Elliptic PDEs arise as components in the physical models and solution methods for a variety of problems, and commonly appear in the following contexts:

- **Physical models.** Elliptic PDEs describe the steady-state, equilibrium behavior of many physical phenomena, such as gravitation, electrostatics, and elastostatics. Poisson's equation is the prototypical elliptic PDE,

  $$\nabla^2 \phi = f,$$

  and arises in a wide variety of physical contexts. Interpreting ϕ as the density of some quantity (e.g., chemical concentration in space), Poisson's equation describes the distribution of ϕ in equilibrium when subject to external forcing f [62]. If ρ is an electric charge density, then the electric potential ϕ due to ρ satisfies the Poisson equation

  $$\nabla^2 \phi = -\frac{\rho}{\epsilon_0},$$

where ϵ_0 is the permittivity of free space. If ρ is a mass density, then the gravitational potential ϕ due to ρ satisfies the Poisson equation

$$\nabla^2 \phi = 4\pi G \rho,$$

where G is the gravitational constant.

- **Projection methods**. Consider the incompressible Navier–Stokes equations,

$$\rho \left(\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} \right) = -\nabla p + \mu \nabla^2 \boldsymbol{u}, \tag{1.1.1}$$

$$\nabla \cdot \boldsymbol{u} = 0, \tag{1.1.2}$$

which describe the motion of an incompressible viscous fluid with velocity $\boldsymbol{u}$, pressure p, constant density ρ, and constant viscosity μ. To solve these equations numerically, projection methods [45, 46] are often used, which enforce the fluid velocity $\boldsymbol{u}$ to be divergence-free at each time step of a simulation. Let $\boldsymbol{u}^k$ and p^k denote the velocity and pressure, respectively, at time step k of a simulation with step size Δt. Projection methods compute an intermediate velocity $\boldsymbol{u}^*$ through an update rule which fixes the pressure term in (1.1.1) (e.g., by extrapolating a guess for p from the previous time step or by setting $p = 0$, as below),

$$\rho \left(\frac{\boldsymbol{u}^* - \boldsymbol{u}^k}{\Delta t} + \boldsymbol{u}^k \cdot \nabla \boldsymbol{u}^k \right) = \mu \nabla^2 \boldsymbol{u}^*,$$

followed by an implicit update rule for $\boldsymbol{u}^{k+1}$ which ignores everything but the pressure term,

$$\rho \left(\frac{\boldsymbol{u}^{k+1} - \boldsymbol{u}^*}{\Delta t} \right) = -\nabla p^{k+1}. \tag{1.1.3}$$

However, the pressure p^{k+1} is not known and must be computed so that the resulting velocity field $\boldsymbol{u}^{k+1}$ is incompressible. Taking the divergence of (1.1.3) and enforcing the incompressibility condition (1.1.2) yields the pressure Poisson equation for the unknown pressure p^{k+1},

$$\nabla^2 p^{k+1} = \frac{\rho}{\Delta t} \nabla \cdot \boldsymbol{u}^*. \tag{1.1.4}$$

This elliptic PDE is the key step to enforcing incompressibility in any projection method and must be solved at each time step. An elliptic PDE solver is often the computational bottleneck in large-scale incompressible fluid simulations due to its role in both projection and viscous solves [107].

- **Implicit time-stepping schemes.** Consider a time-dependent parabolic PDE such as the heat equation,

$$\frac{\partial u}{\partial t} = \kappa \nabla^2 u, \tag{1.1.5}$$

which describes how temperature u diffuses in a domain with diffusivity κ. Implicit time-stepping schemes are often used for equations where diffusion processes dominate, as their stability permits large time steps to be taken beyond the Courant–Friedrichs–Lewy limit of $\Delta t \lesssim h^2$ in explicit schemes [53], which can be restrictive when the mesh size h is very small. The implicit Euler scheme applied to (1.1.5) yields a semi-discrete equation for the unknown temperature, u^{k+1}, at time step $k+1$,

$$u^{k+1} - \Delta t \kappa \nabla^2 u^{k+1} = u^k.$$

This is an elliptic PDE which must be solved once per time step to obtain the unknown temperature at the next point in time. In equations where diffusion is not the dominant physical process (e.g., where diffusion competes with convection and reaction processes), implicit-explicit (IMEX) time integration schemes can be used to alleviate the severe time step restrictions imposed by competing length scales [12, 147], and an elliptic PDE may need to be solved at each implicit step of an IMEX scheme.

The development of fast solvers for elliptic PDEs is a beautiful subfield of numerical analysis and has a rich history, from the fast Fourier transform (FFT) and fast diagonalization techniques (1965) [52], to cyclic reduction (1970) [41], to multigrid methods (1977) [33], to hierarchical low-rank approximation and the fast multipole method (1987) [86]. The application of fast methods to high-order accurate discretizations of PDEs is an area of active research, and forms the motivation for much of this thesis.

1.2 High-order methods

Algorithms for the solution of elliptic PDEs have been developed based on a broad range of mathematical ideas. In general, a PDE is discretized by approximating functions and differential operators according to some parameter, e.g., a grid spacing or mesh size h, a polynomial order p, or a quadrature size N. Error between the computed solution and true solution, ϵ, is decreased by changing this parameter, e.g., by using a finer grid ($h \to 0$), a higher degree polynomial ($p \to \infty$), or more quadrature points ($N \to \infty$). Classical discretization techniques, such as finite difference methods or finite volume methods, typically control this error to a low order of accuracy. For a kth order finite difference scheme, the error ϵ decreases algebraically as $\mathcal{O}(h^k)$, and low-order schemes often use $1 \leq k \leq 4$. When $k = 2$, for instance, halving the parameter h by using twice as many grid points in each dimension causes the error to

decrease by a factor of four. Obtaining multiple digits of accuracy may require many successive refinements of h, each requiring more computation time than the last.

While low-order methods have found widespread success in many fields, high-order methods—typified by their high rate of convergence to the solution—offer an attractive alternative. For a given computational cost, high-order methods can produce results to a higher accuracy than traditional low-order techniques can. In large-scale simulations where spatial resolution is already as refined as modern computers allow (e.g., in simulations of the Earth's mantle [146] or aerospace simulations [31]), high-order accurate algorithms may allow for increased resolution of turbulent fluid flows or high-frequency wave scattering at the same computational cost. On high-performance computers where simulations are increasingly memory-bound, simulation time can depend largely on the amount of communication per degree of freedom. High-order methods efficiently utilize this communication bandwidth by inherently providing more computation per degree of freedom (e.g., by interpolating data from many neighboring grid points at once), making them well suited to data parallelism and well poised to leverage high-bandwidth hardware such as graphics processing units (GPUs) [157].

We now briefly describe three high-order accurate discretization techniques, ordered by the smoothness of the approximations they construct, from most smooth to least smooth. Each of the main chapters in this thesis focuses on deriving fast solvers for elliptic PDEs based on one of these methods.

1.2.1 Global spectral methods

On simple geometries, the solution to a PDE may be approximated as a linear combination of basis functions defined over the entirety of the geometry. Such a representation—known as a global spectral method—is global in the sense that the smoothness of the resulting approximation is only limited by the smoothness of the continuous solution. Approximations based on global spectral methods can converge rapidly when the solution is smooth [169], assuming a stable spectral method is used [168]. For example, the error in the degree-p Fourier series approximation to a C^∞ periodic function on $[-1, 1]$ decreases as $\mathcal{O}(p^{-m})$ for every m; if the function is analytic in a complex strip about the real axis, then convergence at a rate of $\mathcal{O}(C^{-p})$ for $C > 1$ is achieved [168]. In practice, one may approximate functions via coefficients in a series expansion or via values through interpolation on a grid (i.e., spectral collocation). Differential operators may be represented by their action on these coefficients or values, allowing linear differential equations to be discretized into linear systems of equations.

The choice of basis affects important properties of the resulting linear systems. In a spectral collocation method, differentiation maps function values on a grid to values of the derivative of the unique global interpolant over that same grid. That is, given a set of $p + 1$ nodes $\{x_k\}_{k=0}^p$ and values $\{u(x_k)\}_{k=0}^p$ of a function u, the derivative $u'(x)$

is approximated at $\{x_k\}$ through differentiation of the unique interpolant, $\mathbb{I}_{\{u(x_j)\}}(x)$, i.e.,

$$u'(x_k) \approx (\mathbb{I}_{\{u(x_j)\}})'(x_k), \qquad k = 0, \ldots, p.$$

As the derivative of u at the kth point depends on the values of u at all points, differentiation is discretized as a dense matrix, and spectral collocation leads to dense discretizations of differential equations.

In stark contrast, the classical Fourier spectral method—which represents periodic functions through coefficients in a Fourier series expansion—exploits the fact that differentiation can be represented in the Fourier basis as a diagonal scaling. If we approximate a function u, periodic on $[-\pi, \pi]$ and continuous with bounded variation, by the degree-p Fourier series

$$u(x) \approx \sum_{k=-p/2}^{p/2} a_k e^{ikx}, \qquad x \in [-\pi, \pi],$$

whose coefficients $\{a_k\}$ can be computed via the fast Fourier transform [52], then the derivative of u can be naturally approximated in the same form, i.e.,

$$u'(x) \approx \sum_{k=-p/2}^{p/2} (ika_k) e^{ikx}, \qquad x \in [-\pi, \pi].$$

This fact allows the Fourier spectral method to discretize differential equations with periodic boundary conditions into sparse linear systems. In the non-periodic setting, a function u, continuous with bounded variation on $[-1, 1]$, may be naturally represented by the degree-p Chebyshev series

$$u(x) \approx \sum_{k=0}^{p} a_k T_k(x), \qquad x \in [-1, 1],$$

with Chebyshev coefficients $\{a_k\}$, where $T_k(x) = \cos(k \cos^{-1} x)$ is the degree-k Chebyshev polynomial of the first kind. However, differentiation is an upper-triangular operation here, as the representation of $T_k'(x)$ in the Chebyshev basis is a global one, i.e.,

$$T_k'(x) = \begin{cases} 2k \sum_{j \text{ odd}}^{k-1} T_j(x), & k \text{ even} \\ 2k \sum_{j \text{ even}}^{k-1} T_j(x) - 1, & k \text{ odd} \end{cases}$$

Approaches for non-periodic problems based on differentiation in the Chebyshev basis—notably, the Chebyshev tau method [83, 102, 128, 130, 131]—thus lead to dense linear systems of equations. However, recent advances in non-periodic spectral methods allow differential equations to be discretized into sparse linear systems by carefully changing polynomial bases when differentiation is performed [125–127, 163].

Such sparse discretizations have renewed interest in the development of fast solvers for spectral methods, and play a central role in Chapters 2 and 3 of this thesis.

These ideas extend beyond one dimension. Multivariate polynomials that are orthogonal on a variety of simple domains may be constructed directly [60] (e.g., Zernike polynomials on the disk [181], spherical harmonics on the sphere [13], Koornwinder polynomials on the triangle [100]) or through tensor products of one-dimensional bases [32, 164, 167, 168, 176, 177], and global spectral methods for the solution of PDEs may be developed analogously. Chapter 2 develops optimal complexity solvers for Poisson's equation based on global spectral methods in two and three dimensions.

1.2.2 Spectral element methods

Spectral element methods (SEMs) [134] extend the approximation power of global spectral methods to domains with meshed geometries. In an SEM, a domain is meshed into a union of elements (e.g., triangles or quadrilaterals in two dimensions) and a piecewise polynomial basis on each element is used to approximate functions locally. A global representation of the solution is obtained through the union of the spectral representations local to each element, which are typically enforced to be continuous or continuously differentiable across element interfaces. Convergence is achieved by either refining the mesh (h-refinement) or increasing the polynomial degree on the elements (p-refinement). In theory, super-algebraic convergence can be observed—even for solutions with singularities—by optimally selecting a refinement strategy (hp-adaptivity) [18]. However, hp-adaptivity theory can require high polynomial degrees which are rarely used in practice, as traditional collocation-based methods can have prohibitive computational costs and numerical stability issues in this regime.

In particular, constructing efficient solvers for traditional high-order nodal element methods can be challenging. Direct solvers can become computationally intractable even for relatively small polynomial degrees as nodal discretizations result in dense linear algebra; in d dimensions, the computational complexity for a direct solver naïvely scales as $\mathcal{O}(p^{3d})$. Iterative solvers may require an increasing number of iterations as p increases because of the difficulties in designing robust preconditioners in the high p regime [129]. Because of these challenges, traditional element methods are typically restricted to low polynomial degrees, and h-refinement is generically preferred over p-refinement irrespective of local error estimators [171]. Even when hp-adaptivity theory—based on the regularity of the PDE solution—indicates that high p should be used, this advice is often ignored due to the computational cost of the high-p regime.

In Chapter 3, we present an SEM based on a sparse global spectral method with a direct solver that scales as $\mathcal{O}(p^4)$ when a degree $p \times p$ discretization is used on each element in 2D. The method is competitive for very high polynomial degrees (e.g., $p \leq 100$), allowing for extensive p-refinement to be performed in regions

where the solution is smooth, and is packaged into a software library, `ultraSEM`, suitable for flexible spectral element computations. The sparse discretizations used by `ultraSEM` enable users to compute with higher p than current SEM libraries (such as Nektar++ [42, 121], Nek5000 [9], MFEM [3], and Firedrake [142]) allow.

1.2.3 Discontinuous Galerkin methods

An approximation to the solution need not be continuous over the entire geometry. By replacing continuity conditions between elements with suitable jump or flux conditions, one obtains the discontinuous Galerkin (DG) method [143], in which the continuous solution is approximated by a piecewise continuous function with (possible) discontinuities between elements.

DG methods have gained broad popularity in recent years. Similar in spirit to finite volume methods that track fluxes on a grid, DG methods can be constructed to conserve mass at the discrete level [91]. They are well-suited to hp-adaptivity, provide high-order accuracy, and can be applied to a wide range of problems on complex geometries with unstructured meshes. Although DG methods were first applied to the discretization of hyperbolic conservation laws, they have been extended to handle elliptic problems and diffusive operators [11]. Such methods include the symmetric interior penalty (SIP) method [10, 57], the Bassi–Rebay (BR1, BR2) methods [25, 26], the local discontinuous Galerkin (LDG) method [51], the compact discontinuous Galerkin (CDG) method [137], the line-based discontinuous Galerkin method [138], and the hybridizable discontinuous Galerkin (HDG) method [48]. In particular, the development of efficient solvers for DG discretizations of elliptic problems is an active area of research. The multigrid method has emerged as a natural candidate due to its success in the continuous finite element and finite difference communities, both as a standalone solver and as a preconditioner for the conjugate gradient (PCG) method. However, the direct application of black-box multigrid techniques to DG discretizations can result in suboptimal performance [5, 82].

In Chapter 4, we present an efficient hp-multigrid scheme for a DG discretization of elliptic problems, formulated around the idea of separately coarsening the underlying discrete gradient and divergence operators. We show that traditional multigrid coarsening of the primal DG formulation leads to poor and suboptimal multigrid performance, whereas coarsening of the flux DG formulation leads to essentially optimal convergence and is equivalent to a purely geometric multigrid method. The resulting operator-coarsening schemes do not require the entire mesh hierarchy to be explicitly built, thereby obviating the need to compute quadrature rules, lifting operators, and other mesh-related quantities on coarse meshes.

1.3 Scientific software

The development of user-friendly scientific software is an important way to facilitate the adoption of new algorithms by computational scientists. Independent of the mathematical algorithms implemented, scientific software libraries that are well maintained, well documented, easy to install, and easy to use are more likely to find success in the scientific community [105]. Software for the solution of elliptic PDEs is no exception, and many open-source scientific software libraries have found widespread success due to their efficient algorithms and user-friendly interfaces for solving elliptic PDEs. Available software libraries for solving elliptic PDEs with high-order methods include:

- Chebfun [58]: A MATLAB toolbox for spectral methods. It includes an automated PDE solver for a spectral method on the rectangle, as well as Helmholtz solvers on the disk, sphere, and ball.
- Dedalus [40]: A Python framework for solving PDEs using spectral methods on semi-periodic domains. It supports MPI-based parallelism, with a focus on computational fluid dynamics applications.
- ApproxFun [124]: A Julia package for spectral methods, including a sparse method for solving PDEs on simple domains.
- MFEM [3]: A C++ library that implements a wide variety of high-order FEM discretizations. It supports MPI-based parallelism and can interface with a range of solvers.
- Nektar++ [42, 121]: A C++ library for spectral element computation and *hp*-adaptivity using both nodal and modal bases.
- Nek5000 [9]: A Fortran library for computational fluid dynamics using the spectral element method.
- deal.II [22]: A C++ library for general finite element computation with a wide variety of high-order FEM discretizations. It supports MPI-based parallelism and can interface with a range of solvers.
- Firedrake [142]: A Python library for the automated solution of PDEs using a variety of finite element discretizations.
- PETSc [20, 21]: A general toolbox that implements a range of scalable solvers, linear algebra routines, preconditioners, and iterative methods.
- hypre [64]: A C library of high-performance preconditioners and solvers for the solution of large, sparse linear systems of equations, with a focus on parallel algebraic multigrid methods.

In that vein, source code for much of the work presented in this thesis is publicly available on GitHub. Moreover, the algorithms described in Chapter 3 have been packaged into a software library that aims to be well-documented and easy to use. The following repositories contain code corresponding to each chapter:

- Chapter 2: https://github.com/danfortunato/fast-poisson-solvers
- Chapter 3: https://github.com/danfortunato/ultraSEM
- Chapter 4: https://github.com/danfortunato/multigrid-ldg

Chapter 2

Fast Poisson solvers for spectral methods[†]

2.1 Introduction

Consider Poisson's equation on a square with homogeneous Dirichlet conditions:

$$u_{xx} + u_{yy} = f, \quad (x, y) \in [-1, 1]^2, \qquad u(\pm 1, \cdot) = u(\cdot, \pm 1) = 0, \tag{2.1.1}$$

where f is a known continuous function and u is the desired solution. When (2.1.1) is discretized by the finite difference (FD) method with a five-point stencil on an $(n+1) \times (n+1)$ equispaced grid, there is a FFT-based algorithm that computes the values of the solution in an optimal[1] $\mathcal{O}(n^2 \log n)$ operations [90]. Many fast Poisson solvers have been developed for low-order approximation schemes using uniform and nonuniform discretizations based on cyclic reduction [41], the fast multipole method [116], and multigrid [76]. This work began with a question:

Is there an optimal complexity spectral method for (2.1.1)?

We find that the answer is yes. In section 2.3, we describe a practical $\mathcal{O}(n^2(\log n)^2)$ algorithm based on the alternating direction implicit (ADI) method [136]. We go on to derive optimal complexity spectral methods for Poisson's equation with homogeneous Dirichlet conditions for the cylinder and solid sphere in section 2.4 and for the cube in section 2.5. In section 2.6, we extend our approach to Poisson's equation with Neumann and Robin boundary conditions. Optimal complexity spectral methods already exist for Poisson's equation on the disk [176,177] and surface of the sphere [167]. This chapter can be seen as an extension of that work.

A typical objection to the practical relevance of spectral methods for Poisson's equation on domains such as the square and cylinder is that the solution generically can have weak corner singularities, which necessarily restricts the convergence rate

[†]This chapter is a modified version of the following jointly authored publication: D. Fortunato and A. Townsend, *Fast Poisson solvers for spectral methods*, IMA J. Numer. Anal., 40 (2020), pp. 1994–2018, https://doi.org/10.1093/imanum/drz034.

[1]True optimal complexity means that the number of operations scales directly with the number of degrees of freedom, which would be $\mathcal{O}(n^2)$ here. Throughout this work, "optimal complexity" and "fast" mean a computational complexity that is optimal up to polylogarithmic factors.

of classical spectral methods to subexponential convergence (for example, consider $\nabla^2 u = -1$ with homogeneous Dirichlet boundary conditions [32, (2.39)]). Therefore, high degree polynomial approximants may be required to globally resolve such solutions. Since the spectral Poisson solvers we describe have optimal complexity, high degree approximants can be computed to resolve solutions with weak corner singularities with a computational cost comparable to low-order methods using the same number of degrees of freedom. Furthermore, methods have been developed to deal with corner singularities for spectral methods, such as mapping and singularity subtraction [32].

The classical finite difference fast Poisson solver provides useful insight into the development of fast solvers. The FD discretization of (2.1.1) with a five-point stencil on an $(n+1) \times (n+1)$ equispaced grid can be written as the following Sylvester matrix equation:

$$KX + XK^T = F, \qquad K = -\frac{1}{h^2} \begin{bmatrix} 2 & -1 & & \\ -1 & 2 & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & 2 \end{bmatrix} \in \mathbb{R}^{(n-1)\times(n-1)}, \tag{2.1.2}$$

where $h = 2/n$, $X_{jk} = u(-1+kh, -1+jh)$, and $F_{jk} = f(-1+kh, -1+jh)$ for $1 \le j,k \le n-1$. Here, the matrix X represents the values of the solution on the interior nodes of the $(n+1) \times (n+1)$ equispaced grid. The eigendecomposition of K is $K = S\Lambda S^{-1}$, where S is the normalized discrete sine transformation (of type I) matrix [104, (2.24)] and $\Lambda = \operatorname{diag}(\lambda_1, \ldots, \lambda_{n-1})$ with $\lambda_k = -4/h^2 \sin^2(\pi k/(2n))$ for $1 \le k \le n-1$ [104, (2.23)]. Substituting $K = S\Lambda S^{-1}$ into $KX + XK^T = F$ and rearranging, we find a simple formula for X:

$$X = S\left(C \circ (S^{-1}FS^{-T})\right)S^T, \qquad C_{jk} = \frac{1}{\lambda_j + \lambda_k}, \tag{2.1.3}$$

where '$\circ$' is the Hadamard matrix product, i.e., $(A \circ B)_{jk} = A_{jk}B_{jk}$. Since $S = S^T = S^{-1}$ and matrix-vector products with S can be computed in $\mathcal{O}(n \log n)$ operations using the FFT [39], X can be computed via (2.1.3) in a total of $\mathcal{O}(n^2 \log n)$ operations.

Now suppose that K in (2.1.2) is replaced by a diagonalizable matrix A so that (2.1.1) has a spectral discretization of the form $AX + XA^T = F$. Then, an analogous formula to (2.1.2) still holds by using the eigendecomposition of A. However, the corresponding formula to (2.1.3) does not lead to a fast Poisson solver because the eigenvectors of A are not known in closed form [174], and deriving an optimal matrix-vector product for the eigenvector matrix of A is an ambitious project in itself. While FFT-based Poisson solvers exploit structured eigenvectors—which spectral discretization matrices do not possess—our method exploits the fact that the spectra of A and $-A$ are separated using the ADI method (see section 2.3).

The ADI method is an iterative method for solving Sylvester equations of the form $AX - XB = F$. It is computationally efficient, compared to the $\mathcal{O}(n^3)$ Bartels–Stewart algorithm [23], when A and B have certain properties (see, for example, P1, P2, and P3 in section 2.2). By carefully designing spectral discretizations for Poisson's equation on the square (see section 2.3), cylinder (see subsection 2.4.1), solid sphere (see subsection 2.4.2), and cube (see section 2.5) as Sylvester equations with desired properties, we are able to derive optimal complexity spectral Poisson solvers.

Spectral methods with ADI-based solvers have been attempted previously. In 1979, Haidvogel and Zhang derived a Chebyshev-tau spectral method that discretizes (2.1.1) as a Sylvester equation of the form $AX + XA^T = F$ with the matrix A being pentadiagonal except for two rows. They then applied the ADI method after precomputing the LU decomposition of A [87]. However, they advocated against their ADI-based Poisson solver in favour of an $\mathcal{O}(n^3)$ algorithm, because their Sylvester equation does not possess favourable properties for the ADI method and the precomputation costs $\mathcal{O}(n^3)$ operations. In section 2.3, we employ a spectral discretization of (2.1.1) that is specifically designed for the ADI method and requires no precomputation, so that we have a provable algorithmic complexity of $\mathcal{O}(n^2(\log n)^2)$.

Many other fast Poisson solvers have found success with low-order methods, such as ones based on (i) cyclic reduction, (ii) multigrid, (iii) the fast multipole method, and (iv) the Fourier method with polynomial subtraction. However, extending these solvers to spectral methods while maintaining optimal complexity proved challenging for various reasons: (i) cyclic reduction is not readily applicable because spectral discretizations of (2.1.1) may not involve matrices with Toeplitz structure; (ii) multigrid convergence factors for spectral collocation discretizations often degrade as n increases [76]; (iii) the fast multipole method has a complexity that depends on the order of accuracy and is suboptimal in the spectral regime [85, 116]; and, (iv) pseudospectral Fourier with polynomial subtraction can be employed to derived an arbitrary-order Poisson solver [14, 34], but any approach based on uniform grids cannot be both numerically stable and spectrally accurate [140]. Thus, for the purpose of developing a fast spectral Poisson solver for (2.1.1), many of the approaches in the literature do not readily lend themselves.

Despite this observation, we stress that these fast Poisson solvers have advantages over global spectral methods; in particular, many of the methods are well-suited to adaptive discretizations, provide broad geometric flexibility beyond the simple domains considered here, and extend to other variable-coefficient elliptic PDEs. Though not considered in this work, our ADI-based solver may also extend to other strongly elliptic PDEs that preserve the necessary separated spectra property, such as the screened Poisson equation. In particular, it is possible that solving variable-coefficient PDEs may be possible using non-commutative ADI [175]. Finally, we do not advocate the use of global spectral methods over low-order or element-based methods for problems with non-smooth or discontinuous input data.

This chapter is structured as follows: In section 2.2, we review the ADI method for solving Sylvester equations. In section 2.3 we derive an optimal complexity spectral

Poisson solver for (2.1.1). In section 2.4, we use partial regularity to derive fast spectral methods for Poisson's equation on the cylinder and solid sphere before discussing how to do the cube in section 2.5. In section 2.6, we describe how our methods can be used to solve Poisson's equation with general boundary conditions.

For notational convenience, throughout this chapter we discretize using the same number of degrees of freedom in each variable, though our code and algorithms do not have this restriction. All code used in this chapter is publicly available [70]. The Poisson solver on the square (see section 2.3) is implemented in Chebfun [58, 164] and can be accessed via the command `chebfun2.poisson`. It is automatically executed in Chebop2 [163] when the user inputs Poisson's equation, and can handle rectangular domains and general Dirichlet boundary conditions (see section 2.6).

2.2 The alternating direction implicit method

The alternating direction implicit method is an iterative algorithm, originally devised by Peaceman and Rachford [136], which solves Sylvester equations of the following form [108]:

$$AX - XB = F, \qquad A, B, F \in \mathbb{C}^{n\times n} \tag{2.2.1}$$

where A, B, and F are known and $X \in \mathbb{C}^{n\times n}$ is the desired solution. In general, the ADI method is executed in an iterative fashion where iterates $X_0, X_1, \ldots,$ are computed in the hope that $\|X - X_j\|_2 \to 0$ as $j \to \infty$. Algorithm 2.2.1 summarizes the ADI method in this iterative form. At the start of the jth iteration, two shifts p_j and q_j are selected, and at the end of each iteration a test is performed to decide if the iterative method should be terminated. There are numerous strategies for selecting the shift parameters and determining when to terminate the iteration [149]. In practice, selecting good shifts for each iteration is of crucial importance for the ADI method to rapidly converge.

2.2.1 ADI as a direct solver

For an integer J, we would like to know upper bounds on $\|X - X_J\|_2$ so that we can determine a priori how many ADI iterations are required to achieve a relative accuracy of $0 < \epsilon < 1$. To develop error bounds on $\|X - X_J\|_2$, we desire (2.2.1) to satisfy three properties. Later, in section 2.3, we will design a spectral discretization of (2.1.1) as a Sylvester equation with these three properties.

Property 1: Normal matrices

This simplifies the error analysis of the ADI method:

Algorithm 2.2.1 The standard ADI method to solve $AX - XB = F$

Input: $A, B, F \in \mathbb{C}^{n\times n}$
Output: $X_j \in \mathbb{C}^{n\times n}$, an approximate solution to $AX - XB = F$
1: $X_0 := 0$
2: $j := 0$
3: **do**
4: Select ADI shifts p_j and q_j
5: Solve $X_{j+1/2}(B - p_j I) = F - (A - p_j I)X_j$ for $X_{j+1/2}$
6: Solve $(A - q_j I)X_{j+1} = F - X_{j+1/2}(B - q_j I)$ for X_{j+1}
7: $j := j + 1$
8: **while** not converged
9: **return** X_j

Figure 2.1: *Pseudocode for the ADI method described as an iterative algorithm for solving $AX - XB = F$. The convergence of X_j to X in the ADI method is particularly sensitive to the shifts $p_0, p_1, \ldots$ and $q_0, q_1, \ldots$. The convergence test at the end of each iteration can also be subtle [149, Sec. 2.2]. We do not use this general form of the ADI method as it does not lead to an algorithm with a provable computational complexity. Instead, we employ the ADI method on Sylvester equations that satisfy P1–P3, where a different variant of the ADI method can be employed (see Algorithm* 2.2.2*).*

P1. *The matrices A and B are normal matrices.*

In particular, when P1 holds there is a bound on the error $\|X - X_J\|_2$ that only depends on the eigenvalues of A and B and the shifts $p_0, \ldots, p_{J-1}$ and $q_0, \ldots, q_{J-1}$ [29]. Specifically,

$$\|X - X_J\|_2 \leq \frac{\sup_{z\in\sigma(A)} |r(z)|}{\inf_{z\in\sigma(B)} |r(z)|}\|X\|_2, \qquad r(z) = \frac{\prod_{j=0}^{J-1}(z - p_j)}{\prod_{j=0}^{J-1}(z - q_j)},$$

where $\sigma(A)$ and $\sigma(B)$ denote the spectra of A and B, respectively. To make the upper bound on $\|X - X_J\|_2$ as small as possible, one hopes to select shifts so that

$$\frac{\sup_{z\in\sigma(A)} |r(z)|}{\inf_{z\in\sigma(B)} |r(z)|} = \inf_{s\in\mathcal{R}_J} \frac{\sup_{z\in\sigma(A)} |s(z)|}{\inf_{z\in\sigma(B)} |s(z)|}, \tag{2.2.2}$$

where $\mathcal{R}_J$ denotes the space of degree (J, J) rational functions. In general, it is challenging to calculate explicit shifts so that $r(z)$ attains the infimum in (2.2.2). However, this problem is (approximately) solved if the next property holds.

Property 2: Real and disjoint spectra

The following property of (2.2.1) allows us to derive explicit expressions for the ADI shifts:

P2. *There are real disjoint non-empty intervals* $[a,b]$ *and* $[c,d]$ *such that* $\sigma(A) \subset [a,b]$ *and* $\sigma(B) \subset [c,d]$.

If P1 and P2 both hold, then we can relax (2.2.2) and select ADI shifts so that

$$\|X - X_J\|_2 \leq Z_J([a,b],[c,d])\|X\|_2, \quad Z_J([a,b],[c,d]) = \inf_{s\in\mathcal{R}_J} \frac{\sup_{z\in[a,b]}|s(z)|}{\inf_{z\in[c,d]}|s(z)|}, \tag{2.2.3}$$

where $Z_J = Z_J([a,b],[c,d])$ is referred to as a Zolotarev number. Since Zolotarev numbers have been extensively studied in the literature [28, 103, 108, 183], we are able to derive explicit expressions for the ADI shifts so that (2.2.3) holds. Moreover, we have an explicit upper bound on Z_J.

Theorem 2.2.1. *Let* J *be a fixed integer and let* X *satisfy* $AX - XB = F$*, where P1 and P2 hold. Run the ADI method with the shifts*

$$p_j = T\left(-\alpha\,\mathrm{dn}\left[\frac{2j+1}{2J}K(k),k\right]\right), \quad q_j = T\left(\alpha\,\mathrm{dn}\left[\frac{2j+1}{2J}K(k),k\right]\right), \tag{2.2.4}$$

for $0 \leq j \leq J-1$*, where* $k = \sqrt{1-1/\alpha^2}$*,* $K(k)$ *is the complete elliptic integral of the first kind [123, (19.2.8)], and* $\mathrm{dn}(z,k)$ *is the Jacobi elliptic function of the third kind [123, (22.2.6)]. Here,* α *is the real number given by* $\alpha = -1 + 2\gamma + 2\sqrt{\gamma^2-\gamma}$ *with* $\gamma = |c-a||d-b|/(|c-b||d-a|)$ *and* T *is the Möbius transformation*[2] *that maps* $\{-\alpha,-1,1,\alpha\}$ *to* $\{a,b,c,d\}$*. Then, the ADI iterate* X_J *satisfies*

$$\|X - X_J\|_2 \leq Z_J\|X\|_2, \qquad Z_J([a,b],[c,d]) \leq 4\left[\exp\left(\frac{\pi^2}{4\mu(1/\sqrt{\gamma})}\right)\right]^{-2J}, \tag{2.2.5}$$

where $\mu(k) = \frac{\pi}{2}K(\sqrt{1-k^2})/K(k)$ *is the Grötzsch ring function.*

Proof. For the α given in the statement of the theorem, there exists a Möbius transformation T that maps $\{-\alpha,-1,1,\alpha\}$ to $\{a,b,c,d\}$ because the two sets of collinear points have the same absolute cross-ratio. Since any Möbius transformation maps rational functions to rational functions, $Z_J([-\alpha,-1],[1,\alpha]) = Z_J([a,b],[c,d])$ with the zeros and poles of the associated rational functions (see (2.2.3)) related by the Möbius transformation T. For the equation $\tilde{A}\tilde{X} - \tilde{X}\tilde{B} = F$ where P1 holds with $\sigma(\tilde{A}) \subset [-\alpha,-1]$

[2]The Möbius transformation is given by $T(z) = (t_1z+t_2)/(t_3z+t_4)$, where $t_1 = a(-\alpha b + b + \alpha c + c) - 2bc$, $t_2 = a(\alpha(b+c) - b + c) - 2\alpha bc$, $t_3 = 2a - (\alpha+1)b + (\alpha-1)c$, and $t_4 = -\alpha(-2a+b+c) - b + c$.

and $\sigma(\tilde{B}) \subset [1, \alpha]$, the ADI shifts to ensure that $\|\tilde{X} - \tilde{X}_J\|_2 \leq Z_J([-\alpha, -1], [1, \alpha])\|\tilde{X}\|_2$ are given in [108, (2.18)] as

$$p_j = -\alpha \, \mathrm{dn}\left[\frac{2j+1}{2J} K\left(\sqrt{1-1/\alpha^2}\right), \sqrt{1-1/\alpha^2}\right], \quad q_j = -p_j, \quad 0 \leq j \leq J-1. \tag{2.2.6}$$

The formula (2.2.4) is immediately derived as $T(p_j)$ and $T(q_j)$, where p_j and q_j are given in (2.2.6). □

We often prefer to simplify the bound in (2.2.5) by removing the Grötzsch ring function from the bound on Z_J. For example, the bound in (2.2.5) remains valid, but is slightly weakened, if $4\mu(1/\sqrt{\gamma})$ is replaced by the upper bound $2\log(16\gamma)$ [28], i.e.,

$$\|X - X_J\|_2 \leq 4\left[\exp\left(\frac{\pi^2}{2\log(16\gamma)}\right)\right]^{-2J} \|X\|_2, \qquad \gamma = \frac{|c-a||d-b|}{|c-b||d-a|}. \tag{2.2.7}$$

Moreover, if $c = -b$ and $d = -a$ (which commonly occurs when $B = -A^T$), then—relabeling the spectra as $[-b, -a]$ and $[a, b]$—the bound simplifies even more as $4\mu(1/\sqrt{\gamma}) = 2\mu(a/b)$ and the bound remains valid if $\mu(a/b)$ is replaced by $\log(4b/a)$. That is,

$$\|X - X_J\|_2 \leq 4\left[\exp\left(\frac{\pi^2}{2\log(4b/a)}\right)\right]^{-2J} \|X\|_2. \tag{2.2.8}$$

Theorem 2.2.1 is very fruitful as it allows us to use the ADI method as a direct solver for $AX - XB = F$ when P1 and P2 hold, since it tells us the number of ADI iterations (and the associated shifts) required to achieve a desired relative error. For a relative accuracy of $0 < \epsilon < 1$, the simplified bound in (2.2.7) shows that $\|X - X_J\|_2 \leq \epsilon\|X\|_2$ if we take

$$J = \left\lceil \frac{\log(16\gamma)\log(4/\epsilon)}{\pi^2} \right\rceil \tag{2.2.9}$$

and we run the ADI method with the shifts given in (2.2.4). Algorithm 2.2.2 summarizes the ADI method on $AX - XB = F$ when P1 and P2 hold. This is the variant of the ADI method that we employ throughout this work.

We appreciate that it is awkward to calculate the shifts in (2.2.4) because they involve complete elliptic integrals and Jacobi elliptic functions. For the reader's convenience, we provide MATLAB code to compute the shifts in Appendix A. Note that computing the shifts can be done in $\mathcal{O}(1)$ operations, independent of n.

Property 3: Fast shifted linear solves

There is still one more important property of $AX - XB = F$. The shifted linear solves in Algorithm 2.2.2 need to be computationally cheap:

Algorithm 2.2.2 The ADI method to solve $AX - XB = F$ when P1 and P2 hold

Input: $A, B, F \in \mathbb{C}^{n\times n}$, $a, b, c, d \in \mathbb{R}$ satisfying P2, and a tolerance $0 < \epsilon < 1$
Output: $X_J \in \mathbb{C}^{n\times n}$ such that $\|X - X_J\|_2 \le \epsilon\|X\|_2$

```
γ := |c − a||d − b| / (|c − b||d − a|)
J := ⌈log(16γ) log(4/ε)/π²⌉
Set p_j and q_j for 0 ≤ j ≤ J − 1 as given in (2.2.4)
X_0 := 0
for j = 0, ..., J − 1 do
    Solve X_{j+1/2}(B − p_j I) = F − (A − p_j I)X_j for X_{j+1/2}
    Solve (A − q_j I)X_{j+1} = F − X_{j+1/2}(B − q_j I) for X_{j+1}
return X_J
```

Figure 2.2: *Pseudocode for the ADI method for solving $AX - XB = F$ when P1 and P2 hold. Here, for any relative accuracy $0 < \epsilon < 1$ the number of ADI iterations, J, and shifts $p_0, \ldots, p_{J-1}$ and $q_0, \ldots, q_{J-1}$ are known such that $\|X - X_J\|_2 \le \epsilon\|X\|_2$.*

P3. *For any $p, q \in \mathbb{C}$, the linear systems $(A - pI)x = b$ and $(B - qI)x = b$ can be solved in $\mathcal{O}(n)$ operations.*

If P3 holds, then each ADI iteration costs only $\mathcal{O}(n^2)$ operations and the overall cost of the ADI method with J iterations is $\mathcal{O}(Jn^2)$ operations.

In summary, properties P1, P2, and P3 are sufficient conditions on $AX - XB = F$ so that (i) we can determine the number of ADI iterations to attain a relative accuracy of $0 < \epsilon < 1$, (ii) we can derive explicit expressions for the ADI shifts, and (iii) we can compute each ADI iteration in $\mathcal{O}(n^2)$ operations.

2.2.2 An ADI-based fast Poisson solver for finite difference methods

We now describe the ADI-based fast Poisson solver with the second-order five-point FD stencil, though the approach easily extends to fourth- and sixth-order FD methods. Recall that the FD discretization of (2.1.1) with a five-point stencil on an $(n+1)\times(n+1)$ equispaced grid is given by the Sylvester equation $KX + XK^T = F$ (see (2.1.2)). We now verify that P1, P2, and P3 hold for $KX + XK^T = F$:

P1: $A = K$ and $B = -K^T$ are real and symmetric, so they are normal matrices.

P2: The eigenvalues of K are given by $-4/h^2 \sin^2(\pi k/(2n))$ for $1 \le k \le n-1$ with $h = 2/n$ [104, (2.23)]. Since $(2/\pi)x \le \sin x \le 1$ for $x \in [0, \pi/2]$ and $h = 2/n$, the eigenvalues of $A = K$ are contained in the interval $[-n^2, -1]$. The eigenvalues of $B = -K^T$ are contained in $[1, n^2]$.

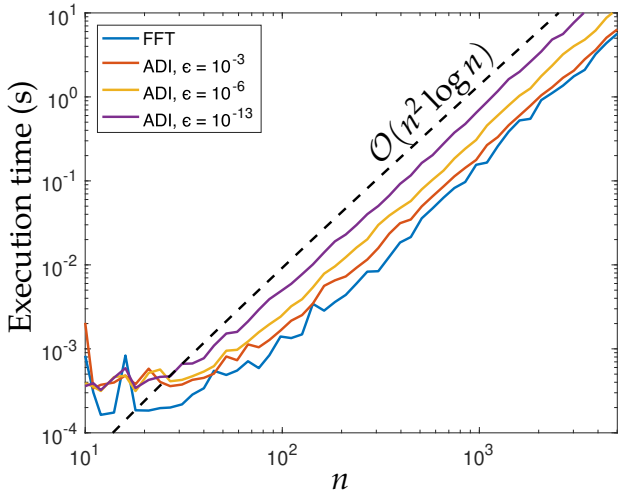


Figure 2.3: *Execution times for the ADI- and FFT-based fast Poisson solvers for a 5-point FD discretization with* $10 \leq n \leq 5000$. *The ADI-based solver is comparable to the FFT-based solver when* $\epsilon = 10^{-3}$. *While the ADI-based fast Poisson solver is computationally more expensive, it is applicable to a carefully designed spectral discretization. Since FFT-based fast Poisson solvers necessarily require uniform grids, they cannot provide a practical optimal complexity spectral method [140].*

P3: For any $p, q \in \mathbb{C}$, the linear systems $(A - pI)x = b$ and $(B - qI)x = b$ are tridiagonal and hence can be solved via the Thomas algorithm in $\mathcal{O}(n)$ operations [54, p. 162].

From the simplified bound in (2.2.8), we conclude that $J = \lceil \log(4n^2) \log(4/\epsilon)/\pi^2 \rceil$ ADI iterations are sufficient to ensure that $\|X - X_J\|_2 \leq \epsilon \|X\|_2$ for $0 < \epsilon < 1$, where the shifts are given in Theorem 2.2.1. Moreover, since P3 holds each ADI iteration only costs $\mathcal{O}(n^2)$ iterations. We conclude that the ADI method in Algorithm 2.2.2 solves $KX + XK^T = F$ in a total of $\mathcal{O}(n^2 \log n \log(1/\epsilon))$ operations. Figure 2.3 demonstrates the execution time[3] of this approach in comparison to the FFT-based fast Poisson solver for $10 \leq n \leq 5000$. While we are not advocating the use of the ADI-based fast Poisson solver for the five-point FD stencil, it does provide flexibility through the choice of an error tolerance ϵ and may be useful for higher-order FD methods and non-uniform grids. As we will show in the next section, ADI-based solvers extend to carefully designed spectral discretizations (see section 2.3).

We expect that one can also derive ADI-based fast Poisson solvers for any $(4w + 1)$-point FD stencil, $1 \leq w \leq \lfloor (n-1)/2 \rfloor$, that run in an optimal number of $\mathcal{O}(n^2 \log n \log(1/\epsilon))$ operations. Because FD discretization matrices have Toeplitz structure, one shifted linear solve only costs $\mathcal{O}(n \log n)$ operations using FFTs [115]. Unfortunately, for $w = \lfloor (n-1)/2 \rfloor$ the resulting spectral method must be numerically unstable because it is based on equispaced nodes [140].

[3] All timings in this chapter were performed in MATLAB R2017a on a 2017 MacBook Pro with no explicit parallelization.

2.3 A fast spectral Poisson solver on the square

Consider Poisson's equation on the square with homogeneous Dirichlet conditions:

$$u_{xx} + u_{yy} = f, \quad (x,y) \in [-1,1]^2, \qquad u(\pm 1, \cdot) = u(\cdot, \pm 1) = 0. \tag{2.3.1}$$

Since (2.3.1) has homogeneous Dirichlet conditions, we know that the solution can be written as $u(x,y) = (1-x^2)(1-y^2)v(x,y)$ for some function $v(x,y)$. To ensure that we are deriving a stable spectral method, we expand $v(x,y)$ in a standard orthogonal polynomial basis[4] [168]. That is,

$$u(x,y) \approx \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} X_{ij}(1-y^2)(1-x^2)\phi_i(y)\phi_j(x), \qquad (x,y) \in [-1,1]^2, \tag{2.3.2}$$

where $\phi_0, \phi_1, \ldots,$ are a sequence of orthogonal polynomials on $[-1,1]$ and the degree of ϕ_j is exactly j for $j \geq 0$. Here, $X \in \mathbb{C}^{n\times n}$ is the matrix of expansion coefficients of the solution[5] and we wish to find X so that the first $n \times n$ coefficients of $u_{xx} + u_{yy}$ match those of f. The choice of the orthogonal polynomial basis is critically important to derive our optimal complexity ADI-based fast Poisson solver. In particular, we want to construct a Sylvester equation for which P1, P2, and P3 hold. If, for example, the Chebyshev basis is selected, then the resulting Sylvester equation does not satisfy P1 from section 2.2.

2.3.1 An ultraspherical polynomial basis

To simplify the discretization of u_{xx} in (2.3.1), we select ϕ_j so that $\frac{d^2}{dx^2}\big[(1-x^2)\phi_j(x)\big]$ has a simple form in terms of $\phi_j(x)$. By the chain rule, we have

$$\frac{d^2}{dx^2}\Big[(1-x^2)\phi_j(x)\Big] = (1-x^2)\phi_j''(x) - 4x\phi_j'(x) - 2\phi_j(x), \tag{2.3.3}$$

[4] Additional benefits of choosing standard orthogonal polynomials include fast evaluation using Clenshaw's algorithm and fast transforms.

[5] More generally, the solution could be represented using a rectangular $m \times n$ discretization, with different discretization sizes in the x- and y-directions. Though we assume $m = n$ throughout this chapter, our results hold for $m \neq n$ and our code supports rectangular discretizations [70].

where a prime indicates one derivative in x. In [123, Chap. 18], one finds that the normalized ultraspherical polynomial,[6] denoted by $\tilde{C}_j^{(3/2)}(x)$, of degree j and parameter $3/2$ satisfies the second-order differential equation [123, Table 18.8.1]

$$(1-x^2)\tilde{C}_j^{(3/2)''}(x) - 4x\tilde{C}_j^{(3/2)'}(x) + j(j+3)\tilde{C}_j^{(3/2)}(x) = 0, \qquad x \in [-1,1]. \quad (2.3.4)$$

In particular, this means that $\tilde{C}_j^{(3/2)}(x)$ is an eigenfunction of the differential operator $u \mapsto \frac{d^2}{dx^2}\big[(1-x^2)u\big]$, i.e.,

$$\frac{d^2}{dx^2}\Big[(1-x^2)\tilde{C}_j^{(3/2)}(x)\Big] = -(j(j+3)+2)\tilde{C}_j^{(3/2)}(x), \qquad j \geq 0.$$

Encouraged by this simplification, we select $\phi_j = \tilde{C}_j^{(3/2)}$ in (2.3.2).

2.3.2 A spectral discretization of Poisson's equation

To construct a discretization of (2.3.1), we apply the Laplacian to the expansion in (2.3.2) to derive a set of equations that the matrix X must satisfy. The action of the Laplacian on each element of our basis is given by

$$\begin{aligned}&\nabla^2\Big[(1-y^2)(1-x^2)\tilde{C}_i^{(3/2)}(y)\tilde{C}_j^{(3/2)}(x)\Big] \\ &= -\Big[(j(j+3)+2)(1-y^2) + (i(i+3)+2)(1-x^2)\Big]\tilde{C}_i^{(3/2)}(y)\tilde{C}_j^{(3/2)}(x).\end{aligned} \quad (2.3.5)$$

Therefore, we can discretize (2.3.1) as a generalized Sylvester equation

$$MXD^T + DXM^T = F, \quad (2.3.6)$$

where X is the matrix of $(1-y^2)(1-x^2)\tilde{C}^{(3/2)}(y)\tilde{C}^{(3/2)}(x)$ expansion coefficients for the solution $u(x,y)$ in (2.3.2), F is the matrix of bivariate $\tilde{C}^{(3/2)}$ expansion coefficients for f (see subsection 2.3.4), D is a diagonal matrix with $D_{jj} = -(j(j+3)+2)$, and M is the $n \times n$ matrix that represents multiplication by $1-x^2$ in the $\tilde{C}^{(3/2)}$ basis. Since

[6] The ultraspherical polynomial of degree j and parameter $\lambda > 0$ is denoted by $C_j^{(\lambda)}$, where $C_0^{(\lambda)}, C_1^{(\lambda)}, \ldots$ are orthogonal on $[-1,1]$ with respect to the weight function $(1-x^2)^{\lambda-1/2}$. The normalized ultraspherical polynomials of parameter $3/2$, denoted by $\tilde{C}_j^{(3/2)}$, satisfy

$$\tilde{C}_j^{(3/2)}(x) = \sqrt{\frac{j+3/2}{(j+1)(j+2)}}C_j^{(3/2)}(x), \qquad j \geq 0,$$

so that $\int_{-1}^{1}(\tilde{C}_j^{(3/2)}(x))^2(1-x^2)dx = 1$.

the recurrence relation for the unnormalized ultraspherical polynomials, $C^{(3/2)}$, is given by [123, (18.9.7) & (18.9.8)]

$$(1-x^2)C_j^{(3/2)}(x) = -\frac{(j+1)(j+2)}{(2j+1)(2j+3)(2j+5)}\Big[(2j+1)C_{j+2}^{(3/2)}(x) - 2(2j+3)C_j^{(3/2)}(x) + (2j+5)C_{j-2}^{(3/2)}(x)\Big],$$

we find—after algebraic manipulations—that M is a symmetric pentadiagonal matrix with

$$M_{j,j} = \frac{2(j+1)(j+2)}{(2j+1)(2j+5)}, \quad M_{j,j+1} = 0, \quad M_{j,j+2} = \frac{-1}{(2j+3)(2j+5)}\sqrt{\frac{(j+4)!(2j+3)}{j!(2j+7)}}. \tag{2.3.7}$$

We can rearrange (2.3.6) by applying D^{-1} to obtain the standard Sylvester equation

$$AX - XB = D^{-1}FD^{-1}, \qquad A = D^{-1}M, \quad B = -M^TD^{-1}. \tag{2.3.8}$$

2.3.3 Verifying that P1, P2, and P3 hold

To guarantee that the ADI method for solving (2.3.8) has optimal complexity, we want the Sylvester equation to satisfy P1, P2, and P3 (see section 2.2). However, the matrices A and B in (2.3.8) are not normal matrices, so we do not solve (2.3.8) using the ADI method directly; we first transform them into the normal matrices $\tilde{A} = D^{1/2}AD^{-1/2}$ and $\tilde{B} = -\tilde{A}^T = -\tilde{A}$. Therefore, to solve (2.3.8) we solve the following Sylvester equation:

$$\tilde{A}Y - Y\tilde{B} = D^{-1/2}FD^{-1/2}, \qquad Y = D^{1/2}XD^{-1/2}, \tag{2.3.9}$$

and recover X via $X = D^{-1/2}YD^{1/2}$. We now verify that P1, P2, and P3 hold for (2.3.9):

- P1: $\tilde{A}$ and $\tilde{B}$ are real and symmetric so are normal matrices,
- P2: The eigenvalues of $\tilde{A}$ are contained in the interval $[-1/2, -1/(2n^4)]$ (see Appendix B). The eigenvalues of $\tilde{B} = -\tilde{A}^T$ are contained in $[1/(2n^4), 1/2]$.
- P3: For any $p, q \in \mathbb{C}$, the linear systems $(\tilde{A} - pI)x = b$ and $(\tilde{B} - qI)x = b$ are pentadiagonal matrices with zero sub- and super-diagonals. Hence, they can be solved in $\mathcal{O}(n)$ operations using the Thomas algorithm [54, p. 162].

By Theorem 2.2.1, we need at most

$$J = \lceil \log(4n^4)\log(4/\epsilon)/\pi^2 \rceil.$$

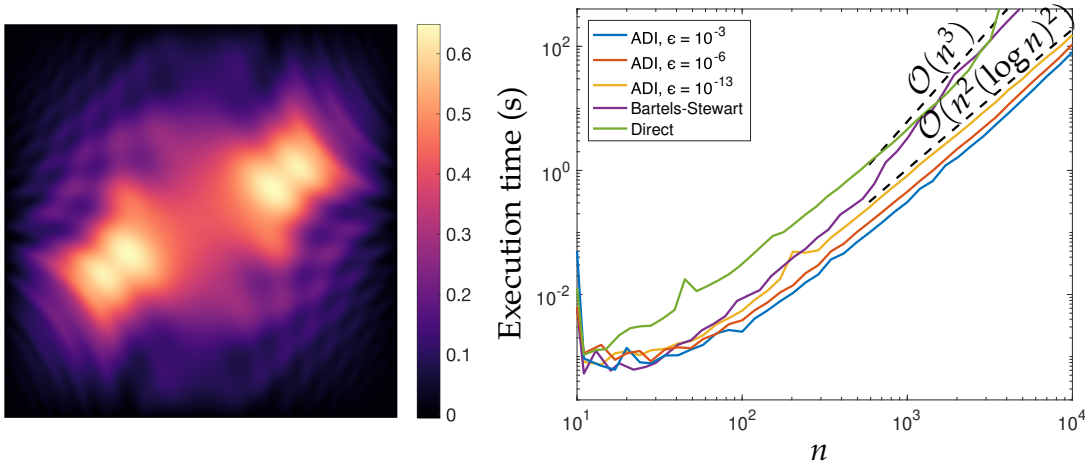


Figure 2.4: *Left: A computed solution to Poisson's equation on the square with right-hand side* $f(x,y) = -100x\sin(20\pi x^2 y)\cos(4\pi(x+y))$ *and* $n = 200$*, using an error tolerance of* $\epsilon = 10^{-13}$*. Right: Execution times for solving* $u_{xx} + u_{yy} = f$ *on* $[-1,1]^2$ *with homogeneous Dirichlet boundary conditions, using our ADI-based solver with various error tolerances, the Bartels–Stewart algorithm [23], and MATLAB's sparse direct solver. The Bartels–Stewart algorithm necessarily densifies its matrices and so cannot exploit the sparsity present in our discretization.*

ADI iterations to ensure that we solve (2.3.9) to within a relative accuracy of $0 < \epsilon < 1$. Since P3 holds, the ADI method solves (2.3.9) in $\mathcal{O}(n^2 \log n \log(1/\epsilon))$ operations, and an additional $\mathcal{O}(n^2)$ operations recovers X from Y.

It is worth reiterating that we use ADI as a direct solver, not an iterative method, here. In particular, it is only the polynomial degree n used to resolve the solution and righthand side that contributes a factor of $\log(n)$ to the number of ADI iterations required. If the rank of the righthand side is r, one can show that the rank of the solution is at most rJ [29]. For small r, the factored ADI (fADI) method can be employed to exploit this low rank structure, resulting in $\mathcal{O}(rn \log n \log(1/\epsilon))$ operations. Moreover, if the righthand side is a smooth function, this structure can also be exploited by factored-independent ADI (FI-ADI) [166]. Our discretizations can be immediately used with both fADI and FI-ADI.

The eigenvalue bounds provided by P2—while disjoint for any finite n—collide as $n \to \infty$, suggesting that for large n numerical stability may be an issue in practice. We have investigated the conditioning of (2.3.9) for a range of values of n using the condition number for Sylvester equations given in [92]. For $n = 100$, we observe a condition number of 10^6; for $n = 1000$, the condition number is 10^{10}. (These bounds are worst-case and are not observed to be sharp, however.) In practice, the convergence rate does not suffer due to the colliding intervals for practical values of n.

2.3.4 Computing the ultraspherical coefficients of a function

So far our Poisson solver assumes that (a) one is given the $\tilde{C}^{(3/2)}$ expansion coefficients for f in (2.3.1) and (b) one is satisfied with the solution returned in the form (2.3.2).

It is known how to compute the Legendre expansion coefficients F_{leg} from f in $\mathcal{O}(n^2(\log n)^2 \log(1/\epsilon))$ operations [165].[7] Using the fact that [123, (18.7.9) & (18.9.7)]

$$(j+\tfrac{1}{2})P_j(x) = \sqrt{\frac{(j+1)(j+2)}{(j+3/2)}}\tilde{C}_j^{(3/2)}(x) - \sqrt{\frac{j(j-1)}{(j-1/2)}}\tilde{C}_{j-2}^{(3/2)}(x), \qquad j \geq 2,$$

there is a sparse upper-triangular matrix S that converts Legendre coefficients to $\tilde{C}^{(3/2)}$ coefficients. Moreover, we can compute $F = S^{-1}F_{\text{leg}}S^{-T}$ in $\mathcal{O}(n^2)$ operations by backwards substitution.

Once the expansion coefficients X in (2.3.2) are known, one can convert the expansion coefficients to a Legendre or Chebyshev basis. The normalized ultraspherical coefficients are given by $X_{\text{ultra}} = MXM^T$ because of the $(1-y^2)(1-x^2)$ factor in (2.3.2). To obtain the Legendre coefficients for u, we note that $X_{\text{leg}} = SX_{\text{ultra}}S^T$. One can now construct a bivariate Chebyshev expansion of u.[8]

2.3.5 Numerical experiments

Table 2.1 summarizes our optimal complexity spectral Poisson solver. The overall complexity is $\mathcal{O}(n^2(\log n)^2 \log(1/\epsilon))$, after the coefficient transforms are taken into account.

Figure 2.4 shows our method compared to the Bartels–Stewart algorithm [23] (invoked via the `lyap` command in MATLAB) and MATLAB's sparse direct solver used to solve the Sylvester equation (2.3.8). The Bartels–Stewart algorithm costs $\mathcal{O}(n^3)$ operations and the direct solver costs an observed $\mathcal{O}(n^{2.5})$ operations[9]; as the timings demonstrate, our method is significantly faster once n is larger than a few hundred, and for large n the sparse direct solver quickly runs into memory constraints. In addition, there are important advantages of ADI in our setting: we are able to relax the tolerance ϵ according to the application, allowing the algorithm to exploit that parameter for a reduced computational cost. The solver can also easily be extended to any rectangular domain $[a,b] \times [c,d]$. Our Poisson solver on the rectangle can be accessed in [70] via the command `poisson_rectangle(F, lbc, rbc, dbc, ubc,`

[7] The Chebfun code to compute the $n \times n$ Legendre coefficients of f is `g = chebfun2(@(x,y) f(x,y)); Fleg = cheb2leg(cheb2leg(chebcoeffs2(g,n,n)).').';` [58].

[8] The Chebfun code to construct a bivariate Chebyshev expansion from a matrix of Legendre coefficients is `u = chebfun2( leg2cheb(leg2cheb(Xleg).').', 'coeffs' );` [58].

[9] The complexity is hard to tell from the timings directly. MATLAB's backslash command invokes UMFPACK with an approximate minimum degree reordering of the columns and rows. The number of nonzeros in the LU decomposition of the permuted linear system scales like $\mathcal{O}(n^{2.5})$, demonstrating that the solver costs at least $\mathcal{O}(n^{2.5})$ operations.

Table 2.1: *Summary of our optimal complexity spectral Poisson solver on the square with an $n \times n$ discretization. The algorithm costs $\mathcal{O}(n^2(\log n)^2 \log(1/\epsilon))$ operations for a working tolerance of $0 < \epsilon < 1$. For $n \leq 5000$, the dominating computational cost in practice is the ADI method.*

Algorithmic step	Cost
1. Compute the $\tilde{C}^{(3/2)}$ coefficients of f in (2.3.1) using [165]	$\mathcal{O}(n^2(\log n)^2 \log(1/\epsilon))$
2. Solve (2.3.9) via the ADI method	$\mathcal{O}(n^2 \log n \log(1/\epsilon))$
3. Compute the solution to (2.3.8) as $X = D^{-1/2} Y D^{1/2}$	$\mathcal{O}(n^2)$
4. Compute the Chebyshev coefficients of u using [165]	$\mathcal{O}(n^2(\log n)^2 \log(1/\epsilon))$

`[a b c d], tol)`, where `F` is the matrix of bivariate Chebyshev coefficients for the right-hand side, `lbc`, `rbc`, `dbc`, and `ubc` denote the left, right, bottom and top Dirichlet data, respectively, and `tol` is the error tolerance.

2.4 Fast spectral Poisson solvers on cylindrical and spherical geometries

We now describe how to extend our fast Poisson solver to cylindrical and spherical geometries. We exploit the fact that both the cylindrical and spherical Laplacians decouple in the azimuthal variable, allowing us to reduce the full three-dimensional problem into n independent two-dimensional problems that can be solved by ADI. On both geometries, we employ a variant of the double Fourier sphere method [117] (see subsection 2.4.1.1) and impose partial regularity on the solution to ensure smoothness.

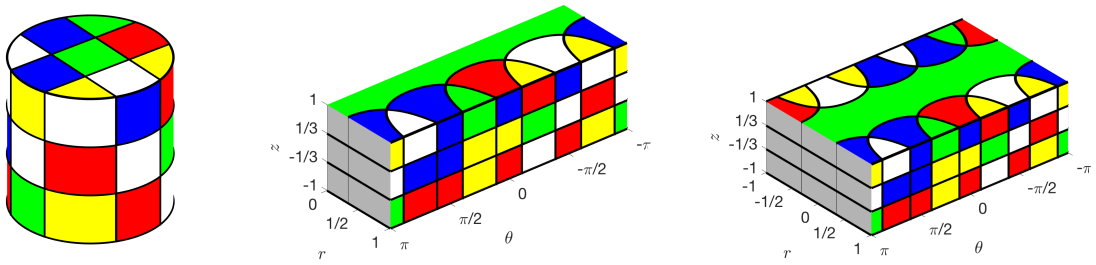
2.4.1 A fast spectral Poisson solver on the cylinder

Here, we consider solving Poisson's equation on the cylinder, i.e., $u_{xx} + u_{yy} + u_{zz} = f$ on $x^2 + y^2 \in [0, 1]$ and $z \in [-1, 1]$ with homogeneous Dirichlet conditions. Our first step is to change to the cylindrical coordinate system, i.e., $(x, y, z) = (r\cos\theta, r\sin\theta, z)$, where $r \in [0, 1]$ is the radial variable and $\theta \in [-\pi, \pi]$ is the angular variable. This change-of-variables transforms Poisson's equation to

$$\frac{\partial^2 u}{\partial r^2} + \frac{1}{r}\frac{\partial u}{\partial r} + \frac{1}{r^2}\frac{\partial^2 u}{\partial \theta^2} + \frac{\partial^2 u}{\partial z^2} = f, \qquad (r, \theta, z) \in [0, 1] \times [-\pi, \pi] \times [-1, 1], \qquad (2.4.1)$$

where $u(1, \theta, z) = 0$ for $(\theta, z) \in [-\pi, \pi] \times [-1, 1]$ and $u(r, \theta, \pm 1) = 0$ for $(r, \theta) \in [0, 1] \times [-\pi, \pi]$.

The coordinate transform has simplified the domain of the differential equation to a rectangle, but has several issues: (1) Any point of the form $(0, \theta, z)$ with $\theta \in [-\pi, \pi]$



(a) (b) (c)

Figure 2.5: *Illustration of the DFS method for a Rubik's cube-colored cylinder. (a) The Rubik's cube-colored cylinder. (b) The Rubik's cube-colored cylinder projected into cylindrical coordinates. (c) The Rubik's cube-colored cylinder after applying the DFS method. The DFS method represents a smooth function $f(x,y,z)$ on the cylinder with a function $f(r,\theta,z)$ on $[-1,1]\times[-\pi,\pi]\times[-1,1]$ that is 2π-periodic in θ and $f(0,\theta,z)$ is a constant for each $\theta\in[-\pi,\pi]$ and $z\in[-1,1]$.*

and $z \in [-1,1]$ maps to $(0,0,z)$ in Cartesian coordinates, introducing an artificial singularity along the center line $r = 0$, (2) The differential equation in (2.4.1) is second-order in the r-variable, but we do not have a natural boundary condition to impose at $r = 0$, and (3) Not every function in the variables (r,θ,z) is a well-defined function on the cylinder, so additional constraints must be satisfied by $u = u(r,\theta,z)$ in (2.4.1).

2.4.1.1 The double Fourier sphere method for the cylinder

The double Fourier sphere (DFS) method, originally proposed for computations on the surface of the sphere [66, 117, 167], is a simple technique that alleviates many of the concerns with cylindrical coordinate transforms. Instead of solving (2.4.1), we "double-up" u and f to $\tilde{u}$ and $\tilde{f}$ and solve

$$\frac{\partial^2\tilde{u}}{\partial r^2}+\frac{1}{r}\frac{\partial\tilde{u}}{\partial r}+\frac{1}{r^2}\frac{\partial^2\tilde{u}}{\partial\theta^2}+\frac{\partial^2\tilde{u}}{\partial z^2}=\tilde{f},\qquad (r,\theta,z)\in[-1,1]\times[-\pi,\pi]\times[-1,1],\quad (2.4.2)$$

where the r-variable is now over $[-1,1]$, instead of $[0,1]$. Here, the solution u (resp. f) is doubled-up as follows:

$$\tilde{u}(r,\theta,z)=\begin{cases}u(r,\theta,z), & (r,\theta,z)\in[0,1]\times[-\pi,\pi]\times[-1,1],\\ u(-r,\theta+\pi,z), & (r,\theta,z)\in[-1,0]\times[-\pi,\pi]\times[-1,1]\end{cases}\qquad (2.4.3)$$

and the homogeneous Dirichlet conditions become $\tilde{u}(\pm1,\theta,z) = 0$ for $(\theta,z) \in [-\pi,\pi]\times[-1,1]$ and $\tilde{u}(r,\theta,\pm1) = 0$ for $(r,\theta) \in [-1,1]\times[-\pi,\pi]$. Figure 2.5 illustrates the DFS method when applied to a Rubik's cube-colored cylinder. Although doubling up leads us to overresolve the solution on the cylinder by a factor of two, this modest cost is outweighed by the optimal complexity of the proposed solver.

The doubled-up functions $\tilde{u}$ and $\tilde{f}$ are non-periodic in the r- and z-variables, and 2π-periodic in the θ-variable. Therefore, we seek the coefficients for $\tilde{u}$ in a Chebyshev–Fourier–Chebyshev expansion:

$$\tilde{u}(r,\theta,z) \approx \sum_{k=-n/2}^{n/2-1} \tilde{u}_k(r,z) e^{\mathrm{i}k\theta}, \qquad \tilde{u}_k(r,z) = \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} X_{ij}^{(k)} T_i(r) T_j(z), \tag{2.4.4}$$

where we assume that n is an even integer and $\tilde{u}_k(r,z)$ denotes the kth Fourier mode of $\tilde{u}(r,\cdot,z)$. We have written the Chebyshev–Fourier–Chebyshev expansion in this form because it turns out that each Fourier mode can be solved for separately. Since $\tilde{f}(r,\theta,z) \approx \sum_{k=-n/2}^{n/2-1} \tilde{f}_k(r,z) e^{\mathrm{i}k\theta}$, we can plug (2.4.4) into (2.4.2) to find that

$$\frac{\partial^2 \tilde{u}_k}{\partial r^2} + \frac{1}{r}\frac{\partial \tilde{u}_k}{\partial r} - \frac{k^2}{r^2}\tilde{u}_k + \frac{\partial^2 \tilde{u}_k}{\partial z^2} = \tilde{f}_k, \qquad (r,z) \in [-1,1]\times[-1,1], \tag{2.4.5}$$

for each $-n/2 \le k \le n/2-1$. This allows us to solve the trivariate PDE in (2.4.2) with a system of n independent bivariate PDEs for each $u_k(r,z)$.

2.4.1.2 Imposing partial regularity on the solution

The issue with (2.4.4) is that a Chebyshev–Fourier–Chebyshev expansion in (r,θ,z) does not necessarily represent a smooth function in (x,y,z) on the cylinder. For instance, $\tilde{u}(0,\theta,z)$ must be a function of the z-variable only for the corresponding function on the cylinder to be continuous. Since we have $x = r\cos\theta$ and $y = r\sin\theta$, we know that the kth Fourier mode $\tilde{u}_k(r,z)$ must decay like $\mathcal{O}(r^{|k|})$ as $r \to 0$. By the uniqueness of Fourier expansions, we also know that $\tilde{u}_k(\pm 1, z) = 0$ and $\tilde{u}_k(r,\pm 1) = 0$ for $-n/2 \le k \le n/2-1$. Therefore, we know that there must be a function[10] $\tilde{v}_k(r,z)$ such that

$$\tilde{u}_k(r,z) = (1-r^2)(1-z^2)r^{|k|}\tilde{v}_k(r,z), \qquad -\frac{n}{2} \le k \le \frac{n}{2}-1. \tag{2.4.6}$$

Ideally, we would like to numerically compute for a bivariate Chebyshev expansion for $\tilde{v}_k(r,z)$ and then recover $\tilde{u}_k(r,z)$ from (2.4.6). This would ensure that the solution $\tilde{u}(r,\theta,z)$ corresponds to a smooth function on the cylinder.

Unfortunately, imposing full regularity on $\tilde{u}_k(r,z)$ is numerically problematic because the regularity condition involves high-order monomial powers. The idea of imposing *partial regularity* on $\tilde{u}_k(r,z)$ avoids the high degree monomial terms [162], and instead $\tilde{u}_k(r,z)$ is written as:

$$\tilde{u}_k(r,z) = (1-r^2)(1-z^2)r^{\min(|k|,2)}\tilde{\omega}_k(r,z), \qquad -\frac{n}{2} \le k \le \frac{n}{2}-1, \tag{2.4.7}$$

[10] One can also show that $\tilde{v}_k(r,z)$ must be an even (odd) function of r if k is even (odd).

where the regularity requirements from (2.4.6) are relaxed. If the functions $\tilde{\omega}_k(r,z)$ are additionally imposed to be even (odd) in r if k is even (odd), then the function $\tilde{u}(r,\theta,z)$ corresponds to at least a continuously differentiable function on the cylinder.

It is worth noting that the imposition of partial regularity on the solution does not seem to affect convergence of our solver to the solutions on either the cylinder or the solid sphere. Our experiments exhibit convergence rates commensurate with the smoothness of the solution, which agrees with previous observations from the disk [162].

2.4.1.3 A solution method for each Fourier mode

The partial regularity conditions in (2.4.7) naturally split into three cases that we treat separately: $|k| \geq 2$ (Case 1), $|k| = 1$ (Case 2), and $k = 0$ (Case 3) . In terms of developing a fast Poisson solver for (2.4.1), it is only important that the PDEs in (2.4.5) for $|k| \geq 2$ are solved in optimal complexity.

Case 1: $|k| \geq 2$**.** The idea is to solve for the function $\tilde{\omega}_k(r,z)$, where $\tilde{u}_k(r,z) = r^2(1-r^2)(1-z^2)\tilde{\omega}_k(r,z)$ and afterwards to recover $\tilde{u}_k(r,z)$. To achieve this, we find the differential equation that $\tilde{\omega}_k(r,z)$ satisfies by substituting (2.4.7) into (2.4.5). After simplifying, we obtain the following equation:

$$\left[\underbrace{r^2(1-r^2)\frac{\partial^2\tilde{\omega}_k}{\partial r^2} + (5-9r^2)r\frac{\partial\tilde{\omega}_k}{\partial r} + 4(1-4r^2)\tilde{\omega}_k}_{=\mathcal{L}_1} - k^2(1-r^2)\tilde{\omega}_k\right](1-z^2) + r^2(1-r^2)\underbrace{\left[(1-z^2)\frac{\partial^2\tilde{\omega}_k}{\partial z^2} - 4z\frac{\partial\tilde{\omega}_k}{\partial z} - 2\tilde{\omega}_k\right]}_{=\mathcal{L}_2} = \tilde{f}_k, \tag{2.4.8}$$

where no boundary conditions are required. Focusing on the z-variable, we observe that $\mathcal{L}_2$ is identical to the differential equation in subsection 2.3.1. Therefore, we represent the z-variable of $\tilde{\omega}_k(r,z)$ in an ultraspherical expansion because $\tilde{C}_j^{(3/2)}$ is an eigenfunction of $\mathcal{L}_2$. For the r-variable, we also use the $\tilde{C}^{(3/2)}$ basis because the multiplication matrix for $(1-r^2)$ is a normal matrix (see (2.3.7)).

Since the $k^2(1-r^2)\tilde{\omega}_k$ term dominates $\mathcal{L}_1$ when k is large, the discretization of $\mathcal{L}_1 - k^2(1-r^2)\tilde{\omega}_k$ in the $\tilde{C}^{(3/2)}$ basis is a near-normal[11] matrix; the matrix tends to a normal matrix as $k \to \infty$. Therefore, we represent $\tilde{\omega}_k(r,z)$ as

$$\tilde{\omega}_k(r,z) \approx \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} Y_{ij}^{(k)}\tilde{C}_i^{(3/2)}(r)\tilde{C}_j^{(3/2)}(z). \tag{2.4.9}$$

[11] A matrix is *near-normal* if the condition number of its eigenvector matrix is close to one.

One can show that an $n \times n$ discretization of $\mathcal{L}_1$ is given by

$$L_1 = M_{r^2} D + 5 M_r M_{1-r^2} D_1 + 14 M_{1-r^2} - 6I,$$

where D is given in (2.3.6), $M_{1-r^2} = M$ (see (2.3.7)), I is the $n \times n$ identity matrix, $M_{r^2} = I - M_{1-r^2}$, M_r is multiplication by r in the $\tilde{C}^{(3/2)}$ basis and D_1 is the first-order differentiation matrix. While D_1 is a upper-triangular dense matrix, we note that $M_{1-r^2} D_1$ is a tridiagonal matrix from [123, (18.9.8) & (18.9.19)]. Moreover, M_r is a tridiagonal matrix [123, Tab. 18.9.1] and hence, L_1 is a pentadiagonal matrix.

Looking at (2.4.8), we find that the coefficient matrix $Y^{(k)}$ in (2.4.9) satisfies

$$(L_1 - k^2 M_{1-r^2}) Y^{(k)} M_{1-r^2}^T + M_{r^2} M_{1-r^2} Y^{(k)} D = F_k,$$

which after rearranging becomes the following Sylvester equation:

$$A Y^{(k)} - Y^{(k)} B = (L_1 - k^2 M_{1-r^2})^{-1} F_k D^{-1}, \tag{2.4.10}$$

where $A = (L_1 - k^2 M_{1-r^2})^{-1} M_{1-r^2}$ and $B = -M_{1-r^2}^T D^{-1}$. Here, B is a normal pentadiagonal matrix after a diagonal similarity transform and A is a near-normal matrix which tends to a normal matrix as k gets large. Moreover, we observe that A has real eigenvalues that are well-separated from the eigenvalues of B and we can solve linear systems of the form $(A - pI)x = b$ in $\mathcal{O}(n)$ operations as $(M_{1-r^2} - p(L_1 - k^2 M_{1-r^2}))x = (L_1 - k^2 M_{1-r^2})b$. Therefore, we can apply ADI to (2.4.10) to solve for each $Y^{(k)}$ in $\mathcal{O}(n^2 (\log n)^2 \log(1/\epsilon))$ operations. Since there are $\mathcal{O}(n)$ such $Y^{(k)}$, the total complexity is $\mathcal{O}(n^3 (\log n)^2 \log(1/\epsilon))$. We recover $\tilde{u}_k(r, z)$ via the relation $\tilde{u}_k(r, z) = r^2(1 - r^2)(1 - z^2)\tilde{\omega}_k(r, z)$.

Case 2: $|k| = 1$**.** We continue to represent $\tilde{\omega}_k(r, z)$ in the expansion (2.4.9). When $|k| = 1$, we find that $\tilde{\omega}_k(r, z)$ satisfies the following partial differential equation:

$$\underbrace{\left[r(1 - r^2) \frac{\partial^2 \tilde{\omega}_k}{\partial r^2} + (3 - 7r^2) \frac{\partial \tilde{\omega}_k}{\partial r} - 8r\tilde{\omega}_k \right]}_{=\mathcal{L}_3} (1 - z^2)$$
$$+ r(1 - r^2) \left[(1 - z^2) \frac{\partial^2 \tilde{\omega}_k}{\partial z^2} - 4z \frac{\partial \tilde{\omega}_k}{\partial z} - 2\tilde{\omega}_k \right] = \tilde{f}_k.$$

We can discretize this as

$$L_3 Y^{(k)} M_{1-r^2}^T + M_r M_{1-r^2} Y^{(k)} D = F_k$$

and solve with the Bartels–Stewart algorithm, costing $\mathcal{O}(n^3)$ operations. Since there are only two Fourier modes with $|k| = 1$, this does not dominate the overall

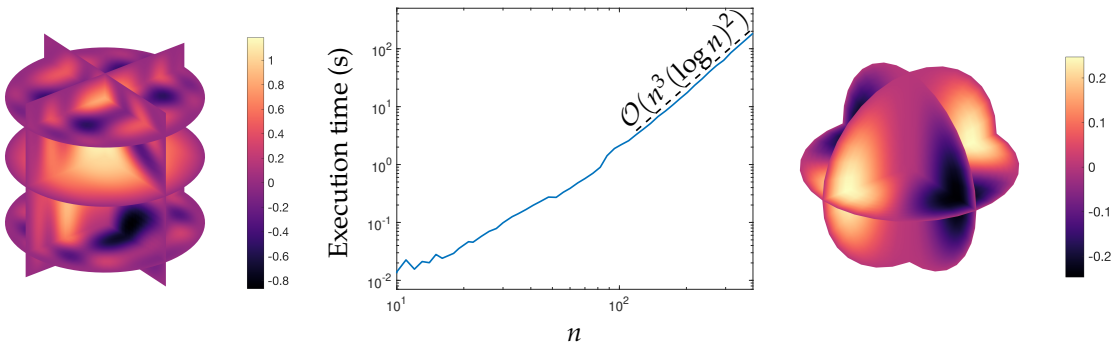


Figure 2.6: *Left: A computed solution to Poisson's equation on the cylinder, shown on various slices through the cylinder. The right-hand side f is such that the exact solution is $u(x,y,z) = (1-x^2-y^2)(1-z^2)(z\cos 4\pi x^2 + \cos 4\pi yz)$. Middle: Execution times for the Poisson solver on the cylinder with an error tolerance of $\epsilon = 10^{-13}$. Right: A computed solution to Poisson's equation on the solid sphere, shown on various slices through the sphere. The right-hand side f is such that the exact solution is $u(r,\theta,\phi) = (1-r^2)(r\sin\phi)^2 e^{i2\theta}$.*

computational complexity of the Poisson solver when n is large. We recover $\tilde{u}_k(r,z)$ via the relation $\tilde{u}_k(r,z) = r(1-r^2)(1-z^2)\tilde{\omega}_k(r,z)$.

Case 3: $k = 0$**.** Finally, the zero Fourier mode satisfies $\tilde{u}_0(r,z) = (1-r^2)(1-z^2)\tilde{\omega}_0(r,z)$ where

$$\underbrace{\left[r^2(1-r^2)\frac{\partial^2\tilde{\omega}_0}{\partial r^2} + (1-5r^2)r\frac{\partial\tilde{\omega}_0}{\partial r} - 4r^2\tilde{\omega}_0\right]}_{=\mathcal{L}_4}(1-z^2)$$
$$+ r^2(1-r^2)\left[(1-z^2)\frac{\partial^2\tilde{\omega}_0}{\partial z^2} - 4z\frac{\partial\tilde{\omega}_0}{\partial z} - 2\tilde{\omega}_0\right] = r^2\tilde{f}_0.$$

We can discretize this as $L_4 Y^{(0)} M_{1-r^2}^T + M_{r^2} M_{1-r^2} Y^{(0)} D = M_{r^2} F_0$ and solve using the Bartels–Stewart algorithm, costing $\mathcal{O}(n^3)$ operations. Again, this cost is negligible for large n since there is only one Fourier mode with $k = 0$.

Figure 2.6 shows a computed solution to Poisson's equation on the cylinder using this algorithm and confirms the optimal complexity of the resulting solver. Our Poisson solver on the cylinder can be accessed in [70] via the command `poisson_cylinder(F, tol)`, where `F` is the tensor of trivariate Chebyshev–Fourier–Chebyshev coefficients for the doubled-up right-hand side and `tol` is the error tolerance.

2.4.2 A fast spectral Poisson solver on the solid sphere

Consider Poisson's equation on the unit ball, i.e., $u_{xx} + u_{yy} + u_{zz} = f$ on $x^2 + y^2 + z^2 \in [0,1]$ with homogeneous Dirichlet conditions. Our first step is to change to the spherical coordinate system, i.e., $(x,y,z) = (r\cos\theta\sin\phi, r\sin\theta\sin\phi, r\cos\phi)$ where $r \in [0,1]$ is

the radial variable, $\theta \in [-\pi, \pi]$ is the azimuthal variable, and $\phi \in [0, \pi]$ is the polar variable. This change of variables transforms Poisson's equation to

$$\frac{\partial^2 u}{\partial r^2} + \frac{2}{r}\frac{\partial u}{\partial r} + \frac{1}{r^2}\frac{\partial^2 u}{\partial \phi^2} + \frac{\cos\phi}{r^2 \sin\phi}\frac{\partial u}{\partial \phi} + \frac{1}{r^2 \sin^2\phi}\frac{\partial^2 u}{\partial \theta^2} = f \tag{2.4.11}$$

for $(r, \theta, \phi) \in [0, 1] \times [-\pi, \pi] \times [0, \pi]$, where $u(1, \theta, \phi) = 0$ for $(\theta, \phi) \in [-\pi, \pi] \times [0, \pi]$.

Similar to the cylinder, we use the DFS method to double-up u and f in both the r- and ϕ-variables and solve for $\tilde{u}$ over the domain $(r, \theta, \phi) \in [-1, 1] \times [-\pi, \pi] \times [-\pi, \pi]$. The doubled-up functions are non-periodic in the r-variable and 2π-periodic in the θ- and ϕ-variables, leading us to seek the coefficients for $\tilde{u}$ in a Chebyshev–Fourier–Fourier expansion:

$$\tilde{u}(r, \theta, \phi) \approx \sum_{k=-n/2}^{n/2-1} \tilde{u}_k(r, \phi) e^{\mathrm{i}k\theta}, \qquad \tilde{u}_k(r, \phi) = \sum_{j=0}^{n-1} \sum_{\ell=-n/2}^{n/2-1} X_{j\ell}^{(k)} T_j(r) e^{\mathrm{i}\ell\phi},$$

where again we have written the expansion in this form because each Fourier mode in θ can be solved for separately. In this case, doubling up leads us to overresolve the solution on the solid sphere by a factor of four, which is again outweighed by the optimal complexity of the proposed solver.

As in the cylinder case, to ensure smoothness in (x, y, z) on the solid sphere we will impose partial regularity on $\tilde{u}_k(r, \phi)$. Since we have $x = r\cos\theta\sin\phi$ and $y = r\sin\theta\sin\phi$, we know that the kth θ-Fourier mode $\tilde{u}_k(r, \phi)$ must decay like $\mathcal{O}((r\sin\phi)^{|k|})$ as $r\sin\phi \to 0$. Therefore, we impose the partial regularity condition:

$$\tilde{u}_k(r, \phi) = (1 - r^2)(r\sin\phi)^{\min(|k|,2)} \tilde{\omega}_k(r, \phi), \qquad -\frac{n}{2} \le k \le \frac{n}{2} - 1,$$

and solve for $\tilde{\omega}_k(r, \phi)$. Again, the partial regularity requirement naturally splits into three cases that we treat separately: $|k| \ge 2$, $|k| = 1$, and $k = 0$. If we represent the r-variable of $\tilde{\omega}_k(r, \phi)$ using the $\tilde{C}_i^{(3/2)}$ basis in r, then for $|k| \ge 2$ we obtain n decoupled sparse Sylvester equations with near-normal matrices which we can solve using ADI in $\mathcal{O}(n^2(\log n)^2 \log(1/\epsilon))$ operations. For $k = -1, 0, 1$, we use the Bartels–Stewart algorithm to solve the Sylvester equation directly in $\mathcal{O}(n^3)$ operations.

Figure 2.6 shows a computed solution to Poisson's equation on the solid sphere using this algorithm. Our Poisson solver on the solid sphere can be accessed in [70] via the command `poisson_solid_sphere(F, tol)`, where `F` is the tensor of trivariate Chebyshev–Fourier–Fourier coefficients for the doubled-up right-hand side and `tol` is the error tolerance.

2.5 A fast spectral Poisson solver on the cube

Consider Poisson's equation on the cube with homogeneous Dirichlet conditions:

$$u_{xx} + u_{yy} + u_{zz} = f, \quad (x, y, z) \in [-1, 1]^3, \quad u(\pm 1, \cdot, \cdot) = u(\cdot, \pm 1, \cdot) = u(\cdot, \cdot, \pm 1) = 0 \tag{2.5.1}$$

From section 2.3, we can discretize (2.5.1) as

$$\left(D_{xx} + D_{yy} + D_{zz}\right) \operatorname{vec}(X) = \operatorname{vec}(F), \tag{2.5.2}$$

where $X, F \in \mathbb{C}^{n \times n \times n}$, $D_{xx} = A \otimes A \otimes I$, $D_{yy} = A \otimes I \otimes A$, and $D_{zz} = I \otimes A \otimes A$. Here, $A = D^{-1}M$ is the pentadiagonal matrix from section 2.3, I is the $n \times n$ identity matrix, '$\otimes$' is the Kronecker product, and $\operatorname{vec}(\cdot)$ is the vectorization operator.

Unlike for the cylinder and sphere, there is no decoupling that allows us to reduce the three-term equation into n two-term equations. Therefore, we would like to solve (2.5.2) using a generalization of the ADI method without constructing the large Kronecker product matrices; however, it is unclear how to generalize ADI to handle more than two terms at a time [173, p. 31]. Instead, we employ the nested ADI method described in [172]. This simply involves grouping the first two terms together and performing the ADI-like iteration given by

$$(D_{zz} - p_{i,1} I) \operatorname{vec}(X_{i+1/2}) = \operatorname{vec}(F) - ((D_{xx} + D_{yy}) - p_{i,1} I) \operatorname{vec}(X_i) \tag{2.5.3}$$

$$((D_{xx} + D_{yy}) - q_{i,1} I) \operatorname{vec}(X_{i+1}) = \operatorname{vec}(F) - (D_{zz} - q_{i,1} I) \operatorname{vec}(X_{i+1/2}) \tag{2.5.4}$$

for suitable choices of the shift parameters $p_{i,1}$ and $q_{i,1}$. Since the matrices D_{xx}, D_{yy}, and D_{zz} are Kronecker products involving two copies of A and the identity matrix, it can be shown that the eigenvalue bounds on D_{xx}, D_{yy}, and D_{zz} are the same as in section 2.3, but squared. Thus, we require $\mathcal{O}(\log n)$ iterations of (2.5.3)–(2.5.4).

To solve the two-term equation (2.5.4), we can apply a nested ADI iteration to the matrices $D_{xx} - \frac{q_{i,1}}{2} I$ and $D_{yy} - \frac{q_{i,1}}{2} I$ as follows:

$$\left(\left(D_{xx} - \tfrac{q_{i,1}}{2} I\right) - p_{j,2} I\right) \operatorname{vec}(Y_{j+1/2}) = F_i - \left(\left(D_{yy} - \tfrac{q_{i,1}}{2} I\right) - p_{j,2} I\right) \operatorname{vec}(Y_j) \tag{2.5.5}$$

$$\left(\left(D_{yy} - \tfrac{q_{i,1}}{2} I\right) - q_{j,2} I\right) \operatorname{vec}(Y_{j+1}) = F_i - \left(\left(D_{xx} - \tfrac{q_{i,1}}{2} I\right) - q_{j,2} I\right) \operatorname{vec}(Y_{j+1/2}) \tag{2.5.6}$$

where $F_i = \operatorname{vec}(F) - (D_{zz} - q_{i,1} I) \operatorname{vec}(X_{i+1/2})$. After the iteration converges, the solution to (2.5.4) is obtained as $X_{i+1} := Y_{j+1}$. For the optimal choices of $p_{j,2}$ and $q_{j,2}$ (see section 2.2) we expect (2.5.5)–(2.5.6) to converge in $\mathcal{O}(\log n)$ iterations.

Finally, we are left with solving the three linear systems (2.5.3), (2.5.5), and (2.5.6), which each involve a shifted Kronecker system. Each Kronecker system is actually degenerate in one dimension, due to the presence of the identity matrix. Thus, we

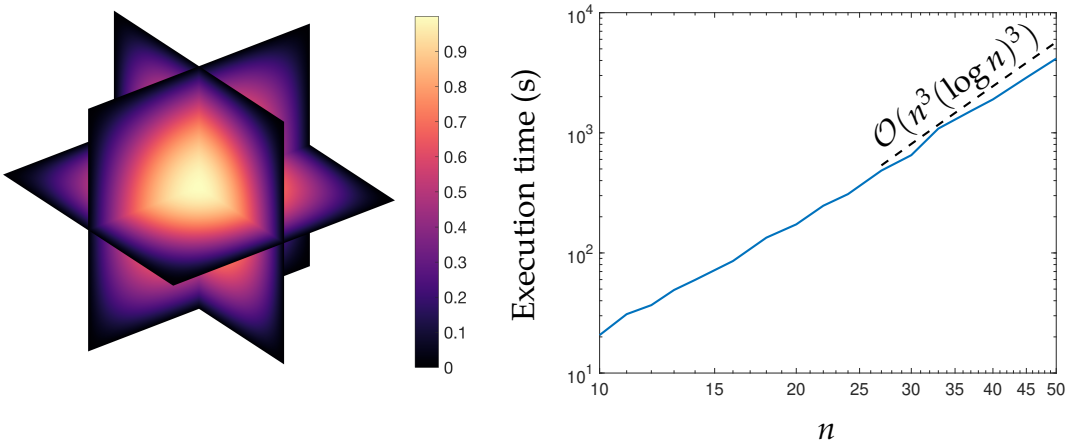


Figure 2.7: *Left: A computed solution to Poisson's equation on the cube, shown on various slices through the cube. The right-hand side f is such that the exact solution is $u(x,y,z) = (1-x^2)(1-y^2)(1-z^2)\cos(xyz^2)$. Right: Execution times for the Poisson solver on the cube with an error tolerance of $\epsilon = 10^{-13}$.*

can decouple (2.5.3), (2.5.5), and (2.5.6) along that degenerate dimension and solve n decoupled systems independently. For example, to solve (2.5.3) for $X_{i+1/2}$ we solve

$$AX_{i+1/2}(:,:,k)A^T - p_{i,1}X_{i+1/2}(:,:,k) = F_i(:,:,k), \qquad 1 \le k \le n, \tag{2.5.7}$$

where $X(:,:,k)$ denotes the kth slice of the tensor X in the z-dimension and $F_i = \text{vec}(F) - ((D_{xx}+D_{yy}) - p_{i,1}I)\,\text{vec}(X_i)$. To solve each of the decoupled systems (2.5.7), we can perform yet another nested ADI iteration. If we rewrite (2.5.7) in the form

$$p_{i,1}A^{-1}X_{i+1/2}(:,:,k) - X_{i+1/2}(:,:,k)A^T = A^{-1}F_i(:,:,k)$$

then the iteration for each k becomes

$$Z_{\ell+1/2}(A^T - p_{\ell,3}I) = A^{-1}F_i(:,:,k) - (p_{i,1}A^{-1} - p_{\ell,3}I)Z_\ell \tag{2.5.8}$$

$$(p_{i,1}I - q_{\ell,3}A)Z_{\ell+1} = AF_i(:,:,k) - AZ_{\ell+1/2}(A^T - q_{\ell,3}I). \tag{2.5.9}$$

After the iteration converges, the solution to (2.5.3) is obtained for each k as $X_{i+1/2}(:,:,k) := Z_{\ell+1}$. Note that we have multiplied (2.5.9) by A so that (2.5.8)–(2.5.9) can be solved fast. For suitable choices of $p_{\ell,3}$ and $q_{\ell,3}$, this will converge in $\mathcal{O}(\log n)$ iterations. Thus, as in section 2.3, each of the n decoupled equations can be solved in $\mathcal{O}(n^2 \log n)$ operations, allowing (2.5.3), (2.5.5), and (2.5.6) to be solved in $\mathcal{O}(n^3 \log n)$ operations. Since there are two levels of nested ADI iterations above this inner computation, the solution to (2.5.1) requires $\mathcal{O}(n^3(\log n)^3 \log(1/\epsilon))$ operations.

Figure 2.7 shows a computed solution to Poisson's equation on the cube using this algorithm and confirms the optimal complexity of the resulting solver. We stress that though this is observed to be an optimal complexity spectral method to solve (2.5.1), it is far from a practical algorithm; the inner ADI iterations must be solved to

machine precision to assure that the outer iterations will converge, resulting in large algorithmic constants that dominate for realistic choices of n. Proper tensor-based approaches [110, 133] based on, e.g., the tensor-train decomposition [132], may be the right choice to develop a practical optimal complexity spectrally-accurate solver for (2.5.1). As in section 2.3, the solver can also be extended to general box-shaped domains. Our Poisson solver on the cube can be accessed in [70] via the command `poisson_cube(F, tol)`, where `F` is the tensor of trivariate Chebyshev coefficients for the right-hand side and `tol` is the error tolerance.

2.6 Nontrivial boundary conditions

So far we have assumed homogeneous Dirichlet boundary conditions. We now describe how to extend our method to handle more general boundary conditions.

2.6.1 Nonhomogeneous Dirichlet conditions

To extend our solver to handle nonhomogeneous Dirichlet conditions, we convert the nonhomogeneous problem into a homogeneous one by moving the boundary conditions to the right-hand side. That is,

1. Compute the coefficients X_{bc} of a function u_{bc} satisfying the Dirichlet data but not necessarily satisfying Poisson's equation (see Appendix C).
2. Compute the Laplacian of u_{bc}.
3. Solve the modified equation $\nabla^2 u_{\text{rhs}} = f - \nabla^2 u_{\text{bc}}$ with homogeneous Dirichlet boundary conditions for the coefficients X_{rhs}.
4. The original solution is then obtained as $X = X_{\text{rhs}} + X_{\text{bc}}$.

Note that the above steps are in coefficient space and can be done fast. This treatment of Dirichlet conditions works for any of the domains discussed in this work.

2.6.2 Neumann and Robin

For Neumann or Robin boundary conditions we must abandon bases containing $(1 - x^2)$ factors and employ a more general discretization scheme. The ultraspherical spectral method [126, 163] discretizes linear PDEs by generalized Sylvester equations with sparse, well-conditioned matrices and can handle boundary conditions in the form of general linear constraints. For Poisson's equation with Neumann or Robin boundary conditions, the method results in a two-term Sylvester equation with

pentadiagonal matrices except for a few dense rows. Experiments indicate that the eigenvalues of the matrices lie within disjoint intervals similar to those in section 2.3, but this is not theoretically justified. However, in practice, we observe that applying the ADI method to these Sylvester equations computes a solution in an optimal number of operations, with accuracy and timings similar to the theoretically-justified methods previously presented for Dirichlet problems.

Chapter 3

The ultraspherical spectral element method[†]

3.1 Introduction

Traditional approaches for solving PDEs on meshed geometries include finite element methods (FEMs) [93], discontinuous Galerkin (DG) methods [50], and spectral element methods (SEMs) [134]. Each approach typically represents the solution of the PDE as a piecewise polynomial, with continuity or jump conditions weakly or strongly imposed between elements. Convergence is achieved by either refining the mesh (h-refinement) or increasing the polynomial degree on the elements (p-refinement). In theory, super-algebraic convergence can be observed—even for solutions with singularities—by optimally selecting a refinement strategy (hp-adaptivity) [18]. However, hp-adaptivity theory can require high polynomial degrees, which are rarely used in practice as traditional methods can have prohibitive computational costs and numerical stability issues in this regime.

In particular, constructing efficient solvers for traditional high-order nodal element methods can be challenging. Direct solvers can become computationally intractable even for relatively small polynomial degrees as nodal discretizations result in dense linear algebra; in d dimensions, the computational complexity for a direct solver naïvely scales as $\mathcal{O}(p^{3d})$. Iterative solvers may require an increasing number of iterations as p increases because of the difficulties in designing robust preconditioners in the high p regime [129]. Because of these challenges, traditional element methods are typically restricted to low polynomial degrees, and h-refinement is generically preferred over p-refinement irrespective of local error estimators [171]. Even when hp-adaptivity theory—based on the regularity of the PDE solution—indicates that high p should be used, this advice is often ignored due to the computational cost of the high-p regime.

Much work has gone toward reducing the computational costs associated with high-order element methods. For discretizations that possess tensor-product structure (e.g., standard nodal bases on quadrilateral elements or certain bases on triangular elements [155]), sum factorization [129] reduces the cost of operator assembly from

[†]This chapter is a modified version of the following jointly authored publication: D. Fortunato, N. Hale, and A. Townsend, *The ultraspherical spectral element method*, Jun 2020, https://arxiv.org/abs/2006.08756.

$\mathcal{O}(p^{3d})$ to $\mathcal{O}(p^{2d+1})$, and matrix-free evaluation reduces the cost of matrix-vector multiplication from $\mathcal{O}(p^{2d})$ to $\mathcal{O}(p^{d+1})$ [3, Tab. 1]. Solvers for the resulting linear systems are often based on iterative methods coupled with sufficient preconditioning. Low-order FEM discretizations on a mesh constructed from the high-order SEM nodes can be shown to be spectrally equivalent to the SEM discretizations [43], and matrix-free preconditioners based on this equivalence can perform well when coupled with a multigrid method using specialized smoothers [135]. Multigrid methods applied to high-order DG discretizations can perform well if the discrete operators are coarsened according to the flux formulation of the PDE [69]. Spectral element multigrid methods have proven effective when applied to nodal discretizations of Poisson's equation in one dimension, though multigrid convergence factors can weakly depend on both h and p [111, 144]. Modal discretizations for p-FEM based on integrated Jacobi polynomials can yield sparse stiffness matrices that contain an optimal number of nonzeros, but developing optimal solvers for such discretizations remains a challenge [30]. Many open-source software libraries exist for high-order element computation, including MFEM [3], Firedrake [142], Nektar++ [42, 121], and Nek5000 [9].

Though solvers for element methods are commonly based on preconditioned iterative methods, fast direct solvers for high-order methods have become an active area of research in recent years [114]. Direct solvers are robust, require no convergence analysis or preconditioners based on problem parameters, and can be repeatedly applied to update a solution cheaply in a design loop. The hierarchical Poincaré–Steklov (HPS) scheme [15, 79, 80, 112–114] is a direct solver for multidomain spectral collocation, based on a recursive domain decomposition approach similar to classical nested dissection. The formulation hierarchically merges Dirichlet-to-Neumann operators and results in an in-memory solution operator, which can be reapplied fast to multiple righthand sides on static meshes. The ability to reuse computed solution operators allows for efficient implicit time-stepping for parabolic problems [16, 17]. The method has been extended to handle mesh adaptivity [74], local geometry deformation [182], three-dimensional problems [88], and boundary integral equations [78]. The HPS scheme based on spectral collocation has an overall complexity of $\mathcal{O}(Np^4 + N^{3/2})$ in two dimensions, where $N \approx (p/h)^2$ is the total number of degrees of freedom, p is the polynomial degree on each element, and h is the minimum mesh element size.

In this chapter, we take advantage of recent advances in sparse spectral methods to propose an SEM in two dimensions with a computational complexity of

$$\underbrace{\frac{p^4}{h^2}}_{\text{initialization stage}} + \underbrace{\frac{p^3}{h^3}}_{\text{build stage}} + \underbrace{\frac{p^3}{h^2} + \frac{p^2}{h^2}\log\frac{1}{h^2}}_{\text{solve stage}} \approx \frac{p^4}{h^2} + \frac{p^3}{h^3} \approx Np^2 + N^{3/2}.$$

Specifically, we propose a variant of the HPS scheme that employs the ultraspherical spectral method [125, 163] instead of spectral collocation for element-wise discretization. The method retains sparsity in the high-p regime by carefully selecting bases

to be specific families of orthogonal polynomials and employing sparse recurrence relations between them. The discretization is not nodal, but modal; that is, the unknowns are not values on a grid, but coefficients in a polynomial expansion.

In this work, we are interested in solving linear PDEs on two-dimensional meshed geometries with Dirichlet boundary conditions,[12] i.e.,

$$\begin{aligned} \mathcal{L}u(x,y) &= f(x,y) \quad \text{in } \Omega, \\ u(x,y) &= g(x,y) \quad \text{on } \partial\Omega. \end{aligned} \tag{3.1.1}$$

Here, Ω is a domain in $\mathbb{R}^2$, f and g are given functions defined on Ω and its boundary, and $\mathcal{L}$ is a variable-coefficient, second-order, elliptic partial differential operator (PDO) of the form

$$\mathcal{L}u = \nabla \cdot (A(x,y)\nabla u) + \nabla \cdot (b(x,y)u) + c(x,y)u, \tag{3.1.2}$$

with $A(x,y) \in \mathbb{C}^{2\times 2}$, $b(x,y) \in \mathbb{C}^2$, and $c(x,y) \in \mathbb{C}$.

This chapter is structured as follows. In section 3.2, we review the ultraspherical spectral method, a sparse and spectrally-accurate method for solving linear ODEs and PDEs on rectangular domains, and discuss its application to quadrilateral and triangular domains. In section 3.3, we extend this spectral method to the non-overlapping domain decomposition setting, highlighting the differences from traditional collocation-based patching approaches. We describe how the hierarchical merging of Poincaré–Steklov operators efficiently performs domain decomposition on meshes with many elements. In section 3.4, we present an implementation of the ultraspherical SEM in the software package `ultraSEM`, and briefly describe its syntax and design. In section 3.5, we present numerical results and applications of the method.

3.2 Background material

3.2.1 The ultraspherical spectral method

First, we review the fundamental ideas in the ultraspherical spectral method [125], which in one dimension solves linear ordinary differential equations (ODEs) with variable coefficients of the form

$$\sum_{\lambda=0}^{M} a_\lambda(x) \frac{d^\lambda u}{dx^\lambda} = f(x), \qquad x \in [-1,1], \tag{3.2.1}$$

[12] Robin boundary conditions can be converted to equivalent Dirichlet boundary conditions using the Dirichlet-to-Neumann operators constructed by the HPS scheme, and so we focus on Dirichlet boundary conditions throughout this work.

along with general linear boundary conditions $\mathcal{B}u = \mathbf{g} \in \mathbb{C}^M$ to ensure that there is a unique solution. For an integer p, the method seeks to approximate the first $p+1$ Chebyshev expansion coefficients $\{u_j\}_{j=0}^{p}$ of the solution u, where

$$u(x) = \sum_{j=0}^{\infty} u_j T_j(x), \qquad x \in [-1, 1],$$

and $T_j(x) = \cos(j \cos^{-1} x)$ is the degree-j Chebyshev polynomial of the first kind.

Classical spectral methods represent differentiation as a dense operator [32, 168], but the ultraspherical spectral method employs the "sparse" recurrence relations

$$\frac{d^\lambda T_j}{dx^\lambda} = \begin{cases} 2^{\lambda-1} j (\lambda-1)!\, C_{j-\lambda}^{(\lambda)}, & j \geq \lambda, \\ 0, & 0 \leq j \leq \lambda - 1, \end{cases} \tag{3.2.2}$$

where $C_j^{(\lambda)}$ is the degree-j ultraspherical polynomial of parameter $\lambda > 0$ [123, Sec. 18.3]. This results in a sparse representation of differentiation operators. In particular, the differentiation operator for the λth derivative is given by

$$\mathcal{D}_\lambda = 2^{\lambda-1}(\lambda-1)! \begin{pmatrix} \overbrace{0 \quad \cdots \quad 0}^{\lambda \text{ times}} & \lambda & & & \\ & & \lambda+1 & & \\ & & & \lambda+2 & \\ & & & & \ddots \end{pmatrix}, \qquad \lambda \in \mathbb{N} \setminus \{0\}.$$

For $\lambda \geq 1$, the matrix $\mathcal{D}_\lambda$ maps a vector of Chebyshev coefficients to a vector of $C^{(\lambda)}$ coefficients of the λth derivative. For convenience, we use $\mathcal{D}_0$ to denote the identity operator.

Since $\mathcal{D}_\lambda$ returns a vector of ultraspherical coefficients for $\lambda \geq 1$, operators to convert between the Chebyshev and ultraspherical bases are required. Let $\mathcal{S}_0$ be the operator that converts a vector of Chebyshev coefficients to a vector of $C^{(1)}$ coefficients, and let $\mathcal{S}_\lambda$, for $\lambda \geq 1$, be the operator that converts a vector of $C^{(\lambda)}$ coefficients to a vector of $C^{(\lambda+1)}$ coefficients. Using the recurrence relations [123, (18.9.7) & (18.9.9)]

$$T_j = \begin{cases} \frac{1}{2}\left(C_j^{(1)} - C_{j-2}^{(1)}\right), & j \geq 2, \\ \frac{1}{2} C_1^{(1)}, & j = 1, \\ C_0^{(1)}, & j = 0, \end{cases} \qquad C_j^{(\lambda)} = \begin{cases} \frac{\lambda}{\lambda+j}\left(C_j^{(\lambda+1)} - C_{j-2}^{(\lambda+1)}\right), & j \geq 2, \\ \frac{\lambda}{\lambda+1} C_1^{(\lambda+1)}, & j = 1, \\ C_0^{(\lambda+1)}, & j = 0, \end{cases}$$

it can be shown that the conversion operators $\mathcal{S}_0$ and $\mathcal{S}_\lambda$ are sparse and given by [125]

$$\mathcal{S}_0 = \begin{pmatrix} 1 & 0 & -\frac{1}{2} & & \\ & \frac{1}{2} & 0 & -\frac{1}{2} & \\ & & \frac{1}{2} & 0 & \ddots \\ & & & \frac{1}{2} & \ddots \\ & & & & \ddots \end{pmatrix}, \quad \mathcal{S}_\lambda = \begin{pmatrix} 1 & 0 & -\frac{\lambda}{\lambda+2} & & \\ & \frac{\lambda}{\lambda+1} & 0 & -\frac{\lambda}{\lambda+3} & \\ & & \frac{\lambda}{\lambda+2} & 0 & \ddots \\ & & & \frac{\lambda}{\lambda+2} & \ddots \\ & & & & \ddots \end{pmatrix}, \quad \lambda \geq 1.$$

To represent multiplication by the variable coefficients $a_\lambda(x)$ in (3.2.1), multiplication operators $\mathcal{M}_\lambda[a_\lambda]$ for $C^{(\lambda)}$ coefficients[13] can be explicitly constructed. If $a_\lambda(x)$ is approximated by a degree-m_λ polynomial, then the operator $\mathcal{M}_\lambda[a_\lambda]$ is m_λ-banded [125].

Discretizing (3.2.1) using these operators to represent differentiation, conversion between bases, and multiplication by variable coefficients results in a banded $(p+1) \times (p+1)$ linear system given by

$$\left(\mathcal{M}_M[a_M]\mathcal{D}_M + \sum_{\lambda=0}^{M-1} \mathcal{S}_{M-1} \cdots \mathcal{S}_\lambda \mathcal{M}_\lambda[a_\lambda]\mathcal{D}_\lambda \right) \mathbf{u} = \mathcal{S}_{M-1} \cdots \mathcal{S}_0 \, \mathbf{f}, \tag{3.2.3}$$

where $\mathbf{u}$ and $\mathbf{f}$ are vectors of Chebyshev coefficients of u and f, respectively. Note that since the order-M differential operator in (3.2.3) maps the vector of Chebyshev coefficients $\mathbf{u}$ to $C^{(M)}$ coefficients, the vector of Chebyshev coefficients $\mathbf{f}$ must also be converted to $C^{(M)}$ coefficients. The bandwidth of the linear system in (3.2.3) scales as $\mathcal{O}(\max_\lambda m_\lambda)$, independent of the polynomial order p. If the variable coefficients $a_\lambda(x)$ can be approximated by polynomials such that $m_\lambda \ll p$, then (3.2.3) is a sparse linear system.

To impose the boundary constraints given by $\mathcal{B}$, we must encode $\mathcal{B}$ in terms of its action on a vector of Chebyshev coefficients. For Dirichlet boundary conditions on $[-1, 1]$, such action is given by

$$\mathcal{B} = \begin{pmatrix} T_0(-1) & T_1(-1) & \cdots & T_p(-1) \\ T_0(1) & T_1(1) & \cdots & T_p(1) \end{pmatrix} = \begin{pmatrix} 1 & -1 & \cdots & (-1)^p \\ 1 & 1 & \cdots & 1 \end{pmatrix}, \tag{3.2.4}$$

because $\mathcal{B}\mathbf{u} \approx (u(-1), u(1))^T$. Neumann, Robin, and more general boundary constraints can be similarly encoded. To impose the M boundary conditions $\mathcal{B}\mathbf{u} = \mathbf{g}$ on the linear system (3.2.3), the ultraspherical spectral method uses boundary bordering [32], wherein the last M rows of the linear system are replaced by dense rows that impose constraints on the Chebyshev coefficients of the solution (e.g., (3.2.4) for Dirichlet boundary conditions). The resulting $(p+1) \times (p+1)$ linear system has a distinctive

[13] The multiplication operator for $\lambda = 0$, $\mathcal{M}_0[a_0]$, acts on a vector of Chebyshev coefficients.

almost banded[14] structure and can be solved in $\mathcal{O}((\max_\lambda m_\lambda)^2 p)$ operations using the adaptive QR algorithm [125] or the Woodbury formula. Figure 3.1 (left) shows the almost banded structure typical of the linear systems in the ultraspherical spectral method.

The ultraspherical spectral method can be extended to solve PDEs in two dimensions on rectangular domains [163]. For the PDE given in (3.1.1) and for a polynomial order p, the method computes modes $X \in \mathbb{C}^{(p+1)\times(p+1)}$ of the solution $u(x,y)$ in a bivariate tensor-product Chebyshev basis, such that

$$u(x,y) = \sum_{i=0}^{\infty}\sum_{j=0}^{\infty} X_{ij} T_i(y) T_j(x), \qquad (x,y) \in [-1,1]^2.$$

Discretization of the PDE is based on separable models of linear partial differential operators. For example, the elliptic PDO $\mathcal{L}$ given by (3.1.2) can be decomposed into a sum of tensor products of one-dimensional differential operators

$$\mathcal{L} = \sum_{j=1}^{K} \left(\mathcal{L}_j^y \otimes \mathcal{L}_j^x \right), \tag{3.2.5}$$

where $\mathcal{L}_1^y, \ldots, \mathcal{L}_K^y$ are operators associated with ODEs in y, $\mathcal{L}_1^x, \ldots, \mathcal{L}_K^x$ are operators associated with ODEs in x. In (3.2.5), the tensor product operator '$\otimes$' is defined such that if $u(x,y) = v(y)w(x)$, then

$$(\mathcal{L}^y \otimes \mathcal{L}^x)\, u(x,y) = (\mathcal{L}^y v(y))\, (\mathcal{L}^x w(x))$$

for some operators $\mathcal{L}^y$ and $\mathcal{L}^x$. Such separable representations of PDOs can be automatically computed [163]. The univariate differential operators $\mathcal{L}_1^y, \ldots, \mathcal{L}_K^y, \mathcal{L}_1^x, \ldots, \mathcal{L}_K^x$ can each be discretized using the ultraspherical spectral method in one dimension, and boundary conditions in x and y can be imposed on the rows and columns of X, thus giving us a scheme for discretizing the PDE. The resulting linear system of size $(p+1)^2 \times (p+1)^2$ is almost block-banded with a bandwidth of $\mathcal{O}(p)$ and $\mathcal{O}(p)$ dense rows, and can be solved in $\mathcal{O}(p^4)$ operations. In special cases, e.g., where $K = 1$ or $K = 2$, further structure can be exploited to arrive at faster solvers [71, 163].

3.2.2 Spectral methods on quadrilaterals and triangles

Global spectral methods defined on rectangles can be used on other polygons through coordinate transformation. Let $\mathcal{Q}_{\text{ref}} = [-1,1]^2$ be the reference square with vertices given by $(r_0, s_0) = (-1,-1)$, $(r_1, s_1) = (1,-1)$, $(r_2, s_2) = (1,1)$, $(r_3, s_3) = (-1,1)$. Denote by (r,s) the coordinates in reference space and by (x,y) the coordinates in real space, and suppose we have a mapping from reference space to real space,

[14] A matrix is *almost banded* if it is banded except for a small number of columns or rows.

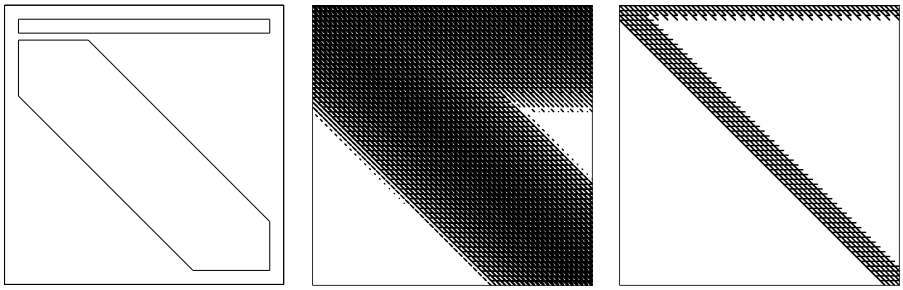


Figure 3.1: *(Left) Typical structure of the almost banded matrices constructed by the ultraspherical spectral method, i.e., banded matrices except for a small number of dense rows. In one dimension, ODEs are discretized as almost banded* $(p+1) \times (p+1)$ *linear systems with bandwidth independent of* p *and* $\mathcal{O}(1)$ *dense rows; such systems can be solved in* $\mathcal{O}(p)$ *operations. In two dimensions on* $[-1,1]^2$*, PDEs are discretized as almost block-banded* $(p+1)^2 \times (p+1)^2$ *linear systems, with a bandwidth of* $\mathcal{O}(p)$ *and* $\mathcal{O}(p)$ *dense rows; such systems can be solved in* $\mathcal{O}(p^4)$ *operations. (Center) In two dimensions on a quadrilateral, PDEs are transformed to* $[-1,1]^2$ *and then discretized. When Jacobian factors are kept as rational functions (see subsection* 3.2.2*), the discrete differential operator has a large bandwidth. (Right) By scaling the transformed PDE twice by the determinant of the Jacobian, the discrete differential operator remains sparse.*

$(r,s) \mapsto (x,y)$. To apply a global spectral method on $\mathcal{Q}_{\text{ref}}$ to a PDE defined in real space, the differential operator $\mathcal{L}$ and righthand side $f(x,y)$ are transformed into reference space. The coordinate transformation alters the differential operator via the chain rule. For a function $u(r,s)$ defined on $\mathcal{Q}_{\text{ref}}$, first- and second-order derivatives in x and y are given by

$$\begin{aligned}
u_x &= r_x u_r + s_x u_s, \\
u_y &= r_y u_r + s_y u_s, \\
u_{xx} &= (r_x)^2 u_{rr} + 2 r_x s_x u_{rs} + (s_x)^2 u_{ss} + r_{xx} u_r + s_{xx} u_s, \\
u_{xy} &= r_x r_y u_{rr} + (r_x s_y + r_y s_x) u_{rs} + s_x s_y u_{ss} + r_{xy} u_r + s_{xy} u_s, \\
u_{yy} &= (r_y)^2 u_{rr} + 2 r_y s_y u_{rs} + (s_y)^2 u_{ss} + r_{yy} u_r + s_{yy} u_s,
\end{aligned}$$

where the Jacobian factors $r_x, r_{xx}, \ldots$ depend on the coordinate mapping. In this work, we are interested in mappings from $\mathcal{Q}_{\text{ref}}$ to quadrilaterals or triangles.

For a quadrilateral domain $\mathcal{Q}$ with vertices $(x_0,y_0), \ldots, (x_3,y_3)$, a bilinear mapping from $(r,s) \in \mathcal{Q}_{\text{ref}}$ to $(x,y) \in \mathcal{Q}$ is given by

$$\begin{bmatrix} r \\ s \end{bmatrix} \mapsto \begin{bmatrix} a_0^x + a_1^x r + a_2^x s + a_3^x rs \\ a_0^y + a_1^y r + a_2^y s + a_3^y rs \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix},$$

where the coefficients $a_0^x, \ldots, a_3^x$ and $a_0^y, \ldots, a_3^y$ satisfy the linear system

$$\begin{bmatrix} 1 & r_0 & s_0 & r_0 s_0 \\ 1 & r_1 & s_1 & r_1 s_1 \\ 1 & r_2 & s_2 & r_2 s_2 \\ 1 & r_3 & s_3 & r_3 s_3 \end{bmatrix} \begin{bmatrix} a_0^x & a_0^y \\ a_1^x & a_1^y \\ a_2^x & a_2^y \\ a_3^x & a_3^y \end{bmatrix} = \begin{bmatrix} x_0 & y_0 \\ x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \end{bmatrix}.$$

While the mapping from (r, s) to (x, y) is bilinear, the mapping from (x, y) to (r, s) is more complicated and in particular is not polynomial, and so we would like to avoid directly computing the inverse maps $r(x, y)$ and $s(x, y)$. Therefore, to compute the first-order Jacobian factors r_x, s_x, r_y, and s_y, we apply the inverse function theorem to the Jacobian matrix $J_{rs} = \partial(r, s)/\partial(x, y)$, which states that $J_{rs} = \left(J_{xy}\right)^{-1}$ with $J_{xy} = \partial(x, y)/\partial(r, s)$. Writing out the Jacobians explicitly, we obtain the following formulae for the first-order factors r_x, s_x, r_y, and s_y:

$$\begin{bmatrix} r_x & r_y \\ s_x & s_y \end{bmatrix} = \begin{bmatrix} x_r & x_s \\ y_r & y_s \end{bmatrix}^{-1} = \frac{1}{\det(J_{xy})} \begin{bmatrix} y_s & -x_s \\ -y_r & x_r \end{bmatrix},$$

where $\det(J_{xy}) = x_r y_s - x_s y_r$. Applying the chain rule to these definitions yields formulae for the second-order factors $r_{xx}, r_{xy}, r_{yy}, s_{xx}, s_{xy}$, and s_{yy}.

However, note that the Jacobian factors are rational functions, due to factors of $\det(J_{xy})$ and $\det(J_{xy})^2$ in the denominators of the first- and second-order terms, respectively. Thus, the coordinate transformation from $\mathcal{Q}$ to $\mathcal{Q}_{\text{ref}}$ introduces rational variable coefficients into the differential operator, and the discretization of the transformed operator by the ultraspherical spectral method results in a linear system with large bandwidth (see Figure 3.1 (center)). To recover sparsity, we scale the transformed differential operator $\mathcal{L}_{rs}$ and righthand side $f(r, s)$ by the factor $\det(J_{xy})^2$ [180], and discretize the scaled PDE

$$\underbrace{\left(\det(J_{xy})^2 \mathcal{L}_{rs}\right)}_{\widehat{\mathcal{L}}_{rs}} u(r, s) = \underbrace{\det(J_{xy})^2 f(r, s)}_{\widehat{f}}.$$

As all Jacobian factors can be written with denominator $\det(J_{xy})^2$, this scaling turns the rational variable coefficients induced by the transformation into polynomial variable coefficients of degree ≤ 2 (see Figure 3.1 (right)). Thus, PDEs on $\mathcal{Q}$ with degree-m variable coefficients are transformed into PDEs on $\mathcal{Q}_{\text{ref}}$ with degree-$(m + 2)$ variable coefficients.

For a triangular domain $\mathcal{T}$, the Duffy transformation [59, 154] may be used to define a mapping from $\mathcal{Q}_{\text{ref}}$ to $\mathcal{T}$ by collapsing one side of $\mathcal{Q}_{\text{ref}}$ to a point. Let $\mathcal{T}_{\text{ref}}$ be the reference triangle with vertices $(x_0, y_0) = (0, 0)$, $(x_1, y_1) = (1, 0)$, and $(x_2, y_2) = (0, 1)$. A mapping from $(r, s) \in \mathcal{Q}_{\text{ref}}$ to $(x, y) \in \mathcal{T}_{\text{ref}}$ can be defined by

$$\begin{bmatrix} r \\ s \end{bmatrix} \mapsto \begin{bmatrix} \frac{1}{4}(1 + r)(1 - s) \\ \frac{1}{2}(1 - s) \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix},$$

which maps the line segment between $(-1,1)$ and $(1,1)$ in $\mathcal{Q}_{\text{ref}}$ to the point $(0,1)$ in $\mathcal{T}_{\text{ref}}$. The inverse of this transformation, mapping from $(x,y) \in \mathcal{T}_{\text{ref}}$ to $(r,s) \in \mathcal{Q}_{\text{ref}}$, possesses a singularity at the point $(0,1)$, i.e.,

$$\begin{bmatrix} x \\ y \end{bmatrix} \mapsto \begin{bmatrix} 2x/(1-y) - 1 \\ 2y - 1 \end{bmatrix} = \begin{bmatrix} r \\ s \end{bmatrix}.$$

If discretized directly, Jacobian factors based on this transformation introduce singular variable coefficients into the differential operator when the operator is transformed to $\mathcal{T}_{\text{ref}}$. However, the singularity induced by the Duffy transformation may be removed by scaling the PDE by powers of $1-y$. For a general triangular domain $\mathcal{T}$ with vertices $(x_0,y_0),\ldots,(x_2,y_2)$, the Duffy transformation may be composed with an affine transformation of the form

$$\begin{bmatrix} x \\ y \end{bmatrix} \mapsto \begin{bmatrix} x_0 + (x_1 - x_0)\, x + (x_2 - x_0)\, y \\ y_0 + (y_1 - y_0)\, x + (y_2 - y_0)\, y \end{bmatrix}$$

to yield a mapping from $\mathcal{Q}_{\text{ref}}$ to $\mathcal{T}$.

We focus our attention on straight-sided quadrilateral elements in the remainder of this work. However, the algorithms presented below can be applied to triangular elements through simple modifications. The `ultraSEM` software supports both triangular and quadrilateral elements.

3.3 The ultraspherical spectral element method

We now describe how to adapt the ultraspherical spectral method into an SEM, focusing on key implementation aspects. Our method is based on the hierarchical Poincaré–Steklov scheme, an efficient non-overlapping domain decomposition approach [15, 79, 80, 112–114]. We employ a variant of the HPS scheme to handle irregular, non-tensor-product meshes (see subsection 3.3.4). Broadly, our method is the following:

1. The method takes as input a second-order elliptic PDO $\mathcal{L}$, a righthand side f, Dirichlet data g, and a mesh with elements $\{\mathcal{E}_i\}_{i=1}^{n_{\text{elem}}}$.

2. On each element, two local operators are constructed: (i) a solution operator, which computes the local solution to the PDE on the element when given Dirichlet data, and (ii) a Dirichlet-to-Neumann operator, which computes the outward flux of the local solution when given Dirichlet data (see subsection 3.3.3.1).

3. Local elemental operators are merged pairwise in a hierarchical fashion, yielding solution operators and Dirichlet-to-Neumann operators, which act on the interfaces between elements or groups of elements. Merging continues until a single global solution operator is computed for the entire mesh (see subsection 3.3.3.2).

4. The given Dirichlet data g is passed in at the top level. Solution operators are applied down the tree, providing the solution at unknown interfaces between elements (see subsection 3.3.3.3).

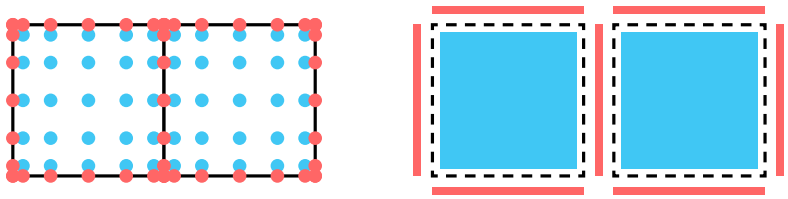
5. Once the solution is known at all the interfaces, local solution operators are applied on each element to determine the interior solution over the entire mesh.

The method lends itself to parallelization. Specifically, steps 2 and 5 can be performed independently on each element as the computations involved are entirely decoupled. Moreover, step 2 is often the bottleneck when p is large, and so significant speedups may be gained if parallelism is exploited (see subsection 3.3.4). The hierarchical steps 3 and 4 may also be parallelized, as the operations taking place on two branches in the hierarchy are decoupled until the two branches are merged. Thus, a careful, load-balanced strategy for parallelizing across branches in the hierarchy may lead to further speedups.

3.3.1 Domain decomposition for modal discretizations

Adapting a domain decomposition approach such as the HPS scheme—originally formulated around a spectral collocation method [112, 113]—to a modal discretization such as the ultraspherical spectral method gives rise to a few subtleties. In the nodal setting, values along interfaces are inherently shared between elements, allowing for an intuitive way to separate the nodes in each element into "interior" and "interface" degrees of freedom and solve for them accordingly (see Figure 3.2a). Cross point conditions (e.g., at a point where the corners of four quadrilaterals meet) can then be avoided by removing the degrees of freedom located at cross points [16]. In the modal setting, on the other hand, the coefficients in a bivariate Chebyshev expansion are not spatially localized, and therefore do not intuitively separate into such categories. To regain a decoupling for Chebyshev coefficients, it is helpful to think about bivariate functions on each element communicating not with each other directly, but with univariate functions on each interface (see Figure 3.2b). Using a modal discretization for these bivariate interior functions and univariate interface functions then allows Chebyshev coefficients to be separated as before. Cross point conditions must then be imposed directly for the resulting linear systems to be nonsingular (see subsection 3.3.3.2).

An alternative remedy to localize modal discretizations is to use a basis that has intrinsic spatial separation between interior and interface, such as a basis consisting of bubble functions (functions that are zero on the edges of an element) and edge functions (functions that are nonzero on the edges of an element) [154]. However, such a basis may not yield a sparse discretization of the PDE. We choose to use the ultraspherical basis to obtain sparse linear algebra, which affords our method a lower computational complexity with respect to p.



(a) Nodal discretization (b) Modal discretization

Figure 3.2: *Two interpretations of non-overlapping domain decomposition for nodal and modal discretizations, with interface data (red) and interior data (blue). (a) In a nodal discretization, neighboring elements communicate directly through degrees of freedom at nodes, which can be partitioned into shared interface nodes and local interior nodes. (b) In a modal discretization, neighboring elements communicate indirectly through unshared interface functions, allowing for coefficients in a modal discretization to be spatially separated.*

3.3.2 Model problem: two "glued" squares

To begin, we consider the simple domain decomposition setting of two square-shaped elements that are "glued" together. That is, we wish to use the ultraspherical spectral method to solve the patching problem[15]

$$\begin{aligned} \nabla^2 u_1 &= f_1 && \text{in } \mathcal{E}_1, \\ \nabla^2 u_2 &= f_2 && \text{in } \mathcal{E}_2, \\ u_1 &= g_1 && \text{on } \partial\mathcal{E}_1 \cap \partial\Omega, \\ u_2 &= g_2 && \text{on } \partial\mathcal{E}_2 \cap \partial\Omega, \\ u_1 &= u_2 && \text{on } \Gamma, \\ \tfrac{\partial u_1}{\partial n_1} + \tfrac{\partial u_2}{\partial n_2} &= 0 && \text{on } \Gamma, \end{aligned} \tag{3.3.1}$$

where $\mathcal{E}$ is a mesh of the domain $\Omega = [-2,2] \times [-1,1]$ with elements $\mathcal{E}_1 = [-2,0] \times [-1,1]$ and $\mathcal{E}_2 = [0,2] \times [-1,1]$, Γ is the interface between the two elements, f and g are given functions, and $f_i = f(\mathcal{E}_i)$ for any function f. This model problem of a pairwise merge serves as a building block in the HPS scheme. The problem setup is depicted in Figure 3.3.

The patching problem (3.3.1) couples two three-sided Dirichlet problems via continuity conditions across the interface Γ. Equivalently, (3.3.1) can be regarded as two decoupled, four-sided Dirichlet problems when given a suitable piece of Dirichlet data along Γ. That is, there exists an interface function φ such that (3.3.1) is equivalent to

$$\begin{aligned} \nabla^2 u_1 &= f_1 && \text{in } \mathcal{E}_1, \\ u_1 &= g_1 && \text{on } \partial\mathcal{E}_1 \cap \partial\Omega, \\ u_1 &= \varphi && \text{on } \Gamma, \end{aligned} \qquad \begin{aligned} \nabla^2 u_2 &= f_2 && \text{in } \mathcal{E}_2, \\ u_2 &= g_2 && \text{on } \partial\mathcal{E}_2 \cap \partial\Omega, \\ u_2 &= \varphi && \text{on } \Gamma. \end{aligned} \tag{3.3.2}$$

[15] It is worth noting that this formulation is equivalent to the global problem $\nabla^2 u = f$ in Ω, $u = g$ on $\partial\Omega$. This holds for any second-order linear elliptic boundary value problem [43].

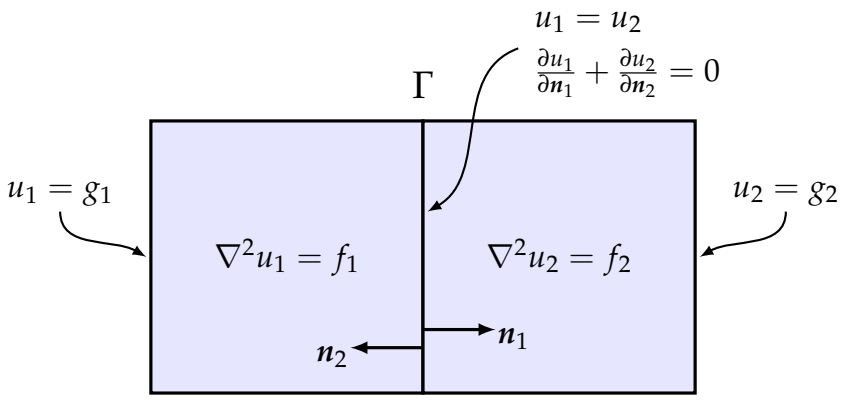


Figure 3.3: *The canonical problem setup for two "glued" squares.*

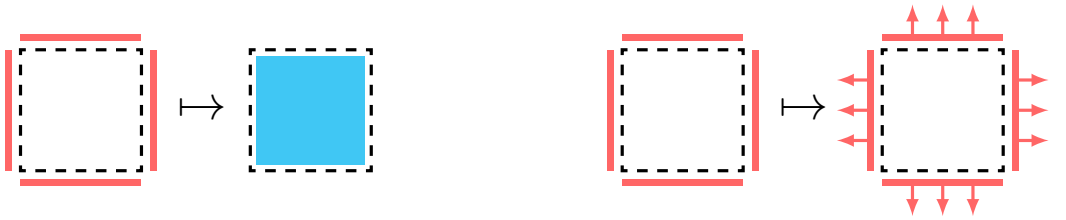
To determine this unknown interface function φ, we aim to build a direct solver—an operator S_Γ such that $\varphi = S_\Gamma g$—using ingredients from local operators on each element. In particular, we construct local direct solvers on $\mathcal{E}_1$ and $\mathcal{E}_2$, and then use pieces of these operators to construct the interfacial solution operator S_Γ. Once the interface function φ is found, the two subproblems in (3.3.2) decouple and can be solved independently by applying local direct solvers on $\mathcal{E}_1$ and $\mathcal{E}_2$. By building a direct solver for the global interface problem based on direct solvers for the subproblems in (3.3.2), the generalization to multiple elements follows naturally.

3.3.2.1 Constructing local operators

To construct a direct solver for (3.3.2), we first build operators that encode how to solve the PDE locally on elements $\mathcal{E}_1$ and $\mathcal{E}_2$. Such operators, called solution operators, take in Dirichlet data and return the corresponding solution to the PDE on an element. For a quadrilateral domain, the solution operator takes in four univariate functions—representing four sides of Dirichlet data—and returns a bivariate function that satisfies the PDE (see Figure 3.4a).

We use the ultraspherical spectral method for solving PDEs on quadrilaterals (see subsection 3.2.2). If on each element we employ a $(p+1) \times (p+1)$ coefficient discretization for the solution so that the solution is at most a degree-(p,p) polynomial, then the solution operator $S_\mathcal{E} \in \mathbb{C}^{(p+1)^2 \times 4(p+1)+1}$ on element $\mathcal{E}$ is a dense matrix. For a column vector $\boldsymbol{c} \in \mathbb{C}^{4(p+1)}$ and scalar $\alpha \in \mathbb{C}$, the product $S_\mathcal{E} \begin{bmatrix} \boldsymbol{c} \\ \alpha \end{bmatrix} \in \mathbb{C}^{(p+1)^2}$ represents the $(p+1) \times (p+1)$ Chebyshev coefficients of the solution to the PDE on $\mathcal{E}$ with Dirichlet data $\boldsymbol{c}$ and righthand side αf.[16] Here, $\boldsymbol{c} = [\boldsymbol{c}_1, \boldsymbol{c}_2, \boldsymbol{c}_3, \boldsymbol{c}_4]^T$ represents the Chebyshev coefficients of four univariate functions of Dirichlet data on the left, right, bottom, and top of $\mathcal{E}$, respectively, each discretized with $p+1$ coefficients. For

[16] In practice, we always take $\alpha = 1$.



(a) Solution operator (b) Dirichlet-to-Neumann operator

Figure 3.4: *A visualization of the local operators computed for each element. (a) The solution operator on a quadrilateral takes in four univariate functions of Dirichlet data and returns a bivariate function that solves the PDE using the given boundary conditions. (b) The Dirichlet-to-Neumann operator on a quadrilateral takes in four univariate functions of Dirichlet data and returns four univariate functions of Neumann data, representing the normal derivative of the solution to the PDE on the four sides.*

example, on the left side, the coefficients $\boldsymbol{c}_1 \in \mathbb{C}^{p+1}$ define the degree-p boundary function $h_1(y)$ as

$$h_1(y) = \sum_{j=0}^{p} (\boldsymbol{c}_1)_j T_j(y).$$

Similarly, $\boldsymbol{c}_2$, $\boldsymbol{c}_3$, and $\boldsymbol{c}_4$ define functions on the other three sides.

The solution operator on $\mathcal{E}$ can be decomposed into four operators $S_{\mathcal{E}}^1, S_{\mathcal{E}}^2, S_{\mathcal{E}}^3, S_{\mathcal{E}}^4 \in \mathbb{C}^{(p+1)^2 \times (p+1)}$ that account for the homogeneous part of the solution and one column vector $S_{\mathcal{E}}^{\text{rhs}} \in \mathbb{C}^{(p+1)^2 \times 1}$ that accounts for the particular part of the solution. That is,

$$S_{\mathcal{E}} = \left[\begin{array}{cccc|c} S_{\mathcal{E}}^1 & S_{\mathcal{E}}^2 & S_{\mathcal{E}}^3 & S_{\mathcal{E}}^4 & S_{\mathcal{E}}^{\text{rhs}} \end{array}\right],$$

where the vector $S_{\mathcal{E}}^{\text{rhs}}$ is defined by

$$S_{\mathcal{E}}^{\text{rhs}} = \text{vec}(X), \qquad u_{\text{rhs}}(x,y) = \sum_{i=0}^{p}\sum_{j=0}^{p} X_{ij} T_i(y) T_j(x),$$

and $\text{vec}(\cdot)$ is the column-wise vectorization operator. Here, u_{rhs} satisfies the PDE on $\mathcal{E}$ with homogeneous boundary conditions, i.e.,

$$\nabla^2 u_{\text{rhs}} = f|_{\mathcal{E}}, \qquad u_{\text{rhs}}|_{\partial\mathcal{E}} = 0.$$

The products $S_{\mathcal{E}}^i \boldsymbol{c}_i$ represent the $(p+1) \times (p+1)$ Chebyshev coefficients of the approximate solution to the homogeneous problem (i.e., $f = 0$) with Dirichlet data on side i given by the Chebyshev coefficients $\boldsymbol{c}_i$ and zero Dirichlet data on the other

three sides. We construct the matrices $S^i_{\mathcal{E}}$ column-by-column. To construct the jth column of $S^i_{\mathcal{E}}$, we set the jth Dirichlet coefficient to one and the rest to zero, i.e.,

$$(\boldsymbol{c}_i)_k = \begin{cases} 1, & \text{if } k = j, \\ 0, & \text{otherwise,} \end{cases} \tag{3.3.3}$$

for $0 \leq k \leq p$. We wish to solve the PDE using this Dirichlet data for each $0 \leq j \leq p$ to obtain the $(p+1) \times (p+1)$ coefficients of the solution, which are reshaped and placed as a column into $S^i_{\mathcal{E}}$. That is, the jth column of the solution operator for the ith side of the element $\mathcal{E}$, i.e., $(S^i_{\mathcal{E}})_{:,j}$, is constructed as

$$(S^i_{\mathcal{E}})_{:,j} = \operatorname{vec}(X), \qquad v_j(x,y) = \sum_{k=0}^{p} \sum_{\ell=0}^{p} X_{k\ell} T_k(y) T_\ell(x),$$

where v_j approximately satisfies the following homogeneous PDE:

$$\nabla^2 v_j = 0, \qquad v_j|_{\partial\mathcal{E}} = \begin{cases} T_j, & \text{on side } i, \\ 0, & \text{otherwise.} \end{cases}$$

Unfortunately, the Dirichlet data used in this construction process may have discontinuities at the corners of the domain, leading to incompatible boundary conditions. To ensure compatibility is satisfied, we orthogonally project each function $\boldsymbol{c}_i$ onto the space of functions that are continuous at the corners before solving the PDE. The compatibility conditions at the four corners of the quadrilateral can be encoded into a matrix $B \in \mathbb{C}^{4\times 4(p+1)}$ given by

$$B = \begin{bmatrix} B_{-1} & 0 & -B_{-1} & 0 \\ B_{+1} & 0 & 0 & -B_{-1} \\ 0 & B_{-1} & -B_{+1} & 0 \\ 0 & B_{+1} & 0 & -B_{+1} \end{bmatrix} \begin{matrix} \} \text{ bottom left corner} \\ \} \text{ top left corner} \\ \} \text{ bottom right corner} \\ \} \text{ top right corner} \end{matrix}$$

with

$$B_{\pm 1} = \begin{bmatrix} T_0(\pm 1) & T_1(\pm 1) & \cdots & T_p(\pm 1) \end{bmatrix},$$

where $T_j(\pm 1) = (\pm 1)^j$. The matrix $B_{\pm 1}$ is an evaluation operator at the endpoints of the interval $[-1, 1]$. So, for the functions h_i defined above, $B_{\pm 1}\boldsymbol{c}_i = h_i(\pm 1)$. A given piece of boundary data defined by the coefficients $\boldsymbol{c} = [\boldsymbol{c}_1, \boldsymbol{c}_2, \boldsymbol{c}_3, \boldsymbol{c}_4]$ is compatible at the corners if and only if $B\boldsymbol{c} = 0$. To project the boundary data so that it satisfies compatibility, we build a basis for $\operatorname{null}(B)$, which is of rank $4(p+1) - 4$. Taking the singular value decomposition $B = U\Sigma V^*$ and letting $\tilde{V}$ be the last $4(p+1) - 4$ columns of V, we construct a projection matrix $P = \tilde{V}\tilde{V}^*$. Since this projection matrix depends only on p, it can be precomputed and stored. The product $\tilde{\boldsymbol{c}} = P\boldsymbol{c}$ orthogonally projects the functions defined by $\boldsymbol{c}_1$, $\boldsymbol{c}_2$, $\boldsymbol{c}_3$, and $\boldsymbol{c}_4$ onto the space of

compatible boundary conditions, so that $\tilde{\boldsymbol{c}}_1$, $\tilde{\boldsymbol{c}}_2$, $\tilde{\boldsymbol{c}}_3$, and $\tilde{\boldsymbol{c}}_4$ are continuous at the four corners of the quadrilateral. We apply this projection during the construction process to the Dirichlet data $\boldsymbol{c}$ in (3.3.3) to obtain compatible Dirichlet data $\tilde{\boldsymbol{c}}$. It is this Dirichlet data that we use to construct the columns of the solution operator $S^i_{\mathcal{E}}$.

Continuity conditions between elements are communicated locally via the Dirichlet-to-Neumann operator, or Poincaré–Steklov operator. The Dirichlet-to-Neumann operator on an element $\mathcal{E}$, denoted by $\Sigma_{\mathcal{E}}$, computes the local solution to the PDE on $\mathcal{E}$ when given Dirichlet data, and then evaluates the outward fluxes of the solution on each side of the element (see Figure 3.4b). Because the local solution is computed as an intermediate step in its calculation, the Dirichlet-to-Neumann operator can be written as a product of the normal derivative operator and the solution operator. That is, $\Sigma_{\mathcal{E}} = D_{\mathcal{E}} S_{\mathcal{E}}$, where $D_{\mathcal{E}}$ computes the outward fluxes of a bivariate function on each side of the element $\mathcal{E}$ when given its $(p+1)^2$ Chebyshev coefficients. On the reference square $[-1,1]^2$, $D_{[-1,1]^2} \in \mathbb{C}^{4(p+1)\times(p+1)^2}$ is given by

$$D_{[-1,1]^2} = \begin{bmatrix} I \otimes D_{-1} \\ I \otimes D_{+1} \\ D_{-1} \otimes I \\ D_{+1} \otimes I \end{bmatrix} \begin{matrix} \} \text{ left normal derivative} \\ \} \text{ right normal derivative} \\ \} \text{ bottom normal derivative} \\ \} \text{ top normal derivative} \end{matrix}$$

where '$\otimes$' denotes the Kronecker product operator for matrices, I is the $(p+1)\times(p+1)$ identity matrix, and

$$D_{\pm 1} = \pm \begin{bmatrix} T_0'(\pm 1) & T_1'(\pm 1) & \cdots & T_p'(\pm 1) \end{bmatrix}, \qquad T_j'(\pm 1) = (\pm 1)^j j^2.$$

On quadrilaterals and triangles, the normal derivative operator is transformed according to the Jacobian factors described in subsection 3.2.2. Hence, the Dirichlet-to-Neumann operator $\Sigma_{\mathcal{E}} \in \mathbb{C}^{4(p+1)\times(4(p+1)+1)}$ is a dense matrix. The product $\Sigma_{\mathcal{E}} \left[\begin{smallmatrix} \boldsymbol{c} \\ \alpha \end{smallmatrix}\right] \in \mathbb{C}^{4(p+1)}$ represents the four normal derivatives of the solution to the PDE on the element $\mathcal{E}$ with Dirichlet data $\boldsymbol{c}$ and righthand side αf, each discretized with $p+1$ Chebyshev coefficients. In the context of the model problem (3.3.2), the Dirichlet-to-Neumann operators $\Sigma_{\mathcal{E}_1}$ and $\Sigma_{\mathcal{E}_2}$ on the elements $\mathcal{E}_1$ and $\mathcal{E}_2$, respectively, are merged to make the interfacial solution operator S_Γ, allowing for the direct solution of the unknown interface function φ.

3.3.2.2 Merging two operators

With local operators constructed on each element $\mathcal{E}_1$ and $\mathcal{E}_2$, we now aim to build a global solution operator, S_Γ, from the local operators $S_{\mathcal{E}_1}$, $S_{\mathcal{E}_2}$, $\Sigma_{\mathcal{E}_1}$, and $\Sigma_{\mathcal{E}_2}$, to solve for the unknown interface function φ. Mathematically, this decomposition mimics the classical Schur complement method for domain decomposition, keeping the physical interpretation for modal discretizations from subsection 3.3.1 in mind.

For elements $\mathcal{E}_1$ and $\mathcal{E}_2$, let Γ_1 and Γ_2 denote the indices of the local Dirichlet data corresponding to the shared boundary Γ. For $\mathcal{E}_1$, since the shared interface Γ is on the right side and the boundary data $\boldsymbol{c} = [\boldsymbol{c}_1, \boldsymbol{c}_2, \boldsymbol{c}_3, \boldsymbol{c}_4]$ is ordered as left, right, bottom, and top, the indices corresponding to the $p+1$ Chebyshev coefficients of the right-side Dirichlet data $\boldsymbol{c}_2$ are given by the set $\Gamma_1 = \{(p+1)+1, \ldots, 2(p+1)\}$. Similarly, since the interface Γ is on the left side of $\mathcal{E}_2$, the indices of the local Dirichlet data on the shared boundary of $\mathcal{E}_2$ are given by $\Gamma_2 = \{1, \ldots, p+1\}$. Finally, denote by L_1 and L_2 the sets containing the indices corresponding to the coefficients of the unshared Dirichlet data on each element, so that $\mathrm{L}_1 = \{1, \ldots, 4(p+1)\} \setminus \Gamma_1$ and $\mathrm{L}_2 = \{1, \ldots, 4(p+1)\} \setminus \Gamma_2$.

With these indices defined for $\mathcal{E}_1$ and $\mathcal{E}_2$ based on interaction with the Dirichlet data on Γ, the rows and columns of the local operators $\Sigma_{\mathcal{E}_1}$ and $\Sigma_{\mathcal{E}_2}$ can be partitioned into "interior" and "interface" blocks. The pieces of $\Sigma_{\mathcal{E}_1}$ and $\Sigma_{\mathcal{E}_2}$ that affect the shared interface naturally separate, and a Schur complement may be performed to write down the following $(p+1) \times (p+1)$ linear system for the solution operator on the interface:

$$-\left(\Sigma_{\mathcal{E}_1}^{\Gamma_1,\Gamma_1} + \Sigma_{\mathcal{E}_2}^{\Gamma_2,\Gamma_2}\right) S_\Gamma = \left[\begin{array}{cc|c} \Sigma_{\mathcal{E}_1}^{\Gamma_1,\mathrm{L}_1} & \Sigma_{\mathcal{E}_2}^{\Gamma_2,\mathrm{L}_2} & \Sigma_{\mathcal{E}_1}^{\Gamma_1,\mathrm{end}} + \Sigma_{\mathcal{E}_2}^{\Gamma_2,\mathrm{end}} \end{array}\right], \tag{3.3.4}$$

where the last column of the righthand side of (3.3.4) encodes the contribution from the particular solution. Here, "end" denotes the index of the last column of a matrix. The linear system in (3.3.4) has a clear interpretation: the matrix $\Sigma_{\mathcal{E}_1}^{\Gamma_1,\Gamma_1} + \Sigma_{\mathcal{E}_2}^{\Gamma_2,\Gamma_2}$ computes the jump in the normal derivative across the shared interface Γ and enforces this jump to be offset by the contributions from the unshared sides and particular solution, resulting in a discrete analogue of the original continuity condition in (3.3.1). As before, the merged solution operator $S_\Gamma \in \mathbb{C}^{(p+1)\times(6(p+1)+1)}$ is a dense matrix. For a column vector $\boldsymbol{c} \in \mathbb{C}^{6(p+1)}$ and scalar $\alpha \in \mathbb{C}$, the product $S_\Gamma \left[\begin{smallmatrix} \boldsymbol{c} \\ \alpha \end{smallmatrix}\right] \in \mathbb{C}^{p+1}$ represents the $p+1$ Chebyshev coefficients of the solution to the PDE on Γ with Dirichlet data $\boldsymbol{c} = [\boldsymbol{c}_1, \ldots, \boldsymbol{c}_6]^T$ and righthand side αf, where now the Dirichlet data $\boldsymbol{c}$ is specified on the six sides of the merged domain Ω.

The Schur complement also allows us to write down the Dirichlet-to-Neumann operator for the merged domain. Using the new solution operator S_Γ, we can construct a new Dirichlet-to-Neumann operator on Ω as

$$\Sigma_\Omega = \left[\begin{array}{cc|c} \Sigma_{\mathcal{E}_1}^{\Gamma_1,\mathrm{L}_1} & 0 & \Sigma_{\mathcal{E}_1}^{\mathrm{L}_1,\mathrm{end}} \\ 0 & \Sigma_{\mathcal{E}_2}^{\Gamma_2,\mathrm{L}_2} & \Sigma_{\mathcal{E}_2}^{\mathrm{L}_2,\mathrm{end}} \end{array}\right] + \left[\begin{array}{c} \Sigma_{\mathcal{E}_1}^{\mathrm{L}_1,\Gamma_1} \\ \Sigma_{\mathcal{E}_2}^{\mathrm{L}_2,\Gamma_2} \end{array}\right] S_\Gamma, \tag{3.3.5}$$

where $\Sigma_\Omega \in \mathbb{C}^{6(p+1)\times(6(p+1)+1)}$. The vector $\Sigma_\Omega \left[\begin{smallmatrix} \boldsymbol{c} \\ \alpha \end{smallmatrix}\right]$ represents normal derivatives on the six sides of Ω of the solution to the PDE on Ω with Dirichlet data $\boldsymbol{c}$ and righthand side αf, each discretized with $p+1$ Chebyshev coefficients.

3.3.2.3 Computing the solution

We now have all the ingredients we need to compute the solution to (3.3.1). We begin by converting the given boundary functions, g_1 and g_2, into Chebyshev coefficients. On each of the three sides of $\mathcal{E}_1$ and $\mathcal{E}_2$ where g_1 and g_2 are known, we construct the degree-p Chebyshev approximant to the boundary data and compile the coefficients into vectors $\boldsymbol{g}_1$ and $\boldsymbol{g}_2$ of length $3(p+1)$. Next, to solve for the interface function φ that makes (3.3.2) equivalent to (3.3.1), we simply compute the matrix-vector product $S_\Gamma \begin{bmatrix} \boldsymbol{g} \\ 1 \end{bmatrix}$, where $\boldsymbol{g} = [\boldsymbol{g}_1, \boldsymbol{g}_2]^T$, which yields the $p+1$ Chebyshev coefficients of φ. With the Dirichlet data now known on all four sides of each of the elements $\mathcal{E}_1$ and $\mathcal{E}_2$, the local solution operators $S_{\mathcal{E}_1}$ and $S_{\mathcal{E}_2}$ can finally be applied. Defining vectors $\widehat{\boldsymbol{g}}_i$ such that $\widehat{\boldsymbol{g}}_i^{\Gamma_i} = \varphi$ and $\widehat{\boldsymbol{g}}_i^{\mathrm{L}_i} = \boldsymbol{g}_i$ for $i = 1, 2$, the matrix-vector products $S_{\mathcal{E}_1} \begin{bmatrix} \widehat{\boldsymbol{g}}_1 \\ 1 \end{bmatrix}$ and $S_{\mathcal{E}_2} \begin{bmatrix} \widehat{\boldsymbol{g}}_2 \\ 1 \end{bmatrix}$ contain the $(p+1) \times (p+1)$ coefficients of the solutions u_1 and u_2, respectively, satisfying (3.3.1).

3.3.3 The hierarchical scheme

At the end of merge process for the model problem of two "glued" squares, we are left with two operators acting on Ω: (1) a solution operator, S_Γ, to solve for the unknown interface inside Ω, and (2) a Dirichlet-to-Neumann operator, Σ_Ω, to map boundary data to outward fluxes on Ω. These operators encode everything we need to know to solve the PDE on Ω. In effect, Ω is now no different from the original elements $\mathcal{E}_1$ or $\mathcal{E}_2$, and so it can be treated as just another element, ready to be merged again with a new domain. After another merge, we are once again in the same situation, with access to local operators that allow us to treat the merged domain as a black box. This is the hierarchical Poincaré–Steklov scheme.

3.3.3.1 Initialization stage

For a mesh $\mathcal{E} = \{\mathcal{E}_i\}_{i=1}^{n_{\text{elem}}}$ of a domain Ω, the scheme begins with an initialization stage, wherein local solution operators $S_{\mathcal{E}_i}$ and Dirichlet-to-Neumann operators $\Sigma_{\mathcal{E}_i}$ are constructed on each element $\mathcal{E}_i$ according to subsection 3.3.2.1. The initialization process is outlined in Algorithm 3.3.1. As the operations performed in the initalization stage are local to each element, Algorithm 3.3.1 can be parallelized across elements.

3.3.3.2 Build stage

Once local operators have been computed for each element, the scheme enters the build stage, where a hierarchy of merged operators is constructed in an upward pass. Given a set of indices $\mathcal{I}$ that define a sequence of pairwise merges between elements, operators are merged as in (3.3.4) and (3.3.5) in the order $\mathcal{I}$ until the entire mesh has been merged into one large conglomerate. Along the way, merged elements store their newly computed solution operators and Dirichlet-to-Neumann operators. The

Algorithm 3.3.1 Initialization stage: `initialize`$(\mathcal{E}, \mathcal{L}, f)$

Input: Mesh $\mathcal{E} = \{\mathcal{E}_i\}_{i=1}^{n_{\text{elem}}}$, partial differential operator $\mathcal{L}$, righthand side f
Output: Solution operators for every element, $\{S_{\mathcal{E}_i}\}_{i=1}^{n_{\text{elem}}}$

1: **for** each element $\mathcal{E}_i$ in mesh **do**
2: Transform $(\mathcal{L}, f) \mapsto (\widehat{\mathcal{L}}, \widehat{f})$ into reference space (see subsection 3.2.2).
3: Discretize $\widehat{\mathcal{L}}$ and $\widehat{f}$ using the ultraspherical spectral method.
4: Construct the solution operator $S_{\mathcal{E}_i}$ (see subsection 3.3.2.1).
5: Construct the Dirichlet-to-Neumann operator $\Sigma_{\mathcal{E}_i} := D_{\mathcal{E}_i} S_{\mathcal{E}_i}$ (see subsection 3.3.2.1).

build stage ends with a solution operator that acts on the entire mesh, taking in Dirichlet data on every boundary of Ω and returning the solution to the PDE along the penultimate merged interface. The build stage is outlined in Algorithm 3.3.2.

When the mesh contains cross points (i.e., points in the interior of the mesh where corners of multiple elements meet), the linear system defining the solution operator,

$$-\left(\Sigma_{\mathcal{E}_i}^{\Gamma_i,\Gamma_i} + \Sigma_{\mathcal{E}_j}^{\Gamma_j,\Gamma_j}\right) S_{\Gamma_{ij}} = \left[\begin{array}{cc|c} \Sigma_{\mathcal{E}_i}^{\Gamma_i,\mathrm{L}_i} & \Sigma_{\mathcal{E}_j}^{\Gamma_j,\mathrm{L}_j} & \Sigma_{\mathcal{E}_i}^{\Gamma_i,\text{end}} + \Sigma_{\mathcal{E}_j}^{\Gamma_j,\text{end}} \end{array}\right], \tag{3.3.6}$$

may be rank deficient, as a continuity condition on the sum of the normal fluxes around the cross point has not been imposed [43]. Rather than imposing this condition directly, we solve the rank-deficient system by finding the minimum-norm solution to (3.3.6) in the least-squares sense [73].

Algorithm 3.3.2 Build stage (upward pass): `build`$(\mathcal{E}, \Sigma_{\mathcal{E}}, \mathcal{I})$

Input: Mesh $\mathcal{E} = \{\mathcal{E}_i\}_{i=1}^{n_{\text{elem}}}$, local operators $\Sigma_{\mathcal{E}} = \{\Sigma_{\mathcal{E}_i}\}_{i=1}^{n_{\text{elem}}}$, merge indices $\mathcal{I}$
Output: Solution operators for every merge, $\{S_{\Gamma_{ij}}\}_{(i,j)\in\mathcal{I}}$

1: **for** each pair in $(i, j) \in \mathcal{I}$ **do**
2: Define the merged domain $\mathcal{E}_{ij} := \mathcal{E}_i \cup \mathcal{E}_j$.
3: Define the shared interface $\Gamma_{ij} := \mathcal{E}_i \cap \mathcal{E}_j$.
4: Define indices Γ_i, Γ_j for the shared boundary Γ_{ij} on $\mathcal{E}_i$, $\mathcal{E}_j$.
5: Define indices $\mathrm{L}_i := \overline{\Gamma_i}$, $\mathrm{L}_j := \overline{\Gamma_j}$ for the unshared boundaries on $\mathcal{E}_i$, $\mathcal{E}_j$.
6: Solve the linear system

$$-\left(\Sigma_{\mathcal{E}_i}^{\Gamma_i,\Gamma_i} + \Sigma_{\mathcal{E}_j}^{\Gamma_j,\Gamma_j}\right) S_{\Gamma_{ij}} = \left[\begin{array}{cc|c} \Sigma_{\mathcal{E}_i}^{\Gamma_i,\mathrm{L}_i} & \Sigma_{\mathcal{E}_j}^{\Gamma_j,\mathrm{L}_j} & \Sigma_{\mathcal{E}_i}^{\Gamma_i,\text{end}} + \Sigma_{\mathcal{E}_j}^{\Gamma_j,\text{end}} \end{array}\right]$$

for the merged solution operator $S_{\Gamma_{ij}}$.
7: Define the merged Dirichlet-to-Neumann operator,

$$\Sigma_{\mathcal{E}_{ij}} := \left[\begin{array}{cc|c} \Sigma_{\mathcal{E}_i}^{\Gamma_i,\mathrm{L}_i} & 0 & \Sigma_{\mathcal{E}_i}^{\mathrm{L}_i,\text{end}} \\ 0 & \Sigma_{\mathcal{E}_j}^{\Gamma_j,\mathrm{L}_j} & \Sigma_{\mathcal{E}_j}^{\mathrm{L}_j,\text{end}} \end{array}\right] + \left[\begin{array}{c} \Sigma_{\mathcal{E}_i}^{\mathrm{L}_i,\Gamma_i} \\ \Sigma_{\mathcal{E}_j}^{\mathrm{L}_j,\Gamma_j} \end{array}\right] S_{\Gamma_{ij}}.$$

8: **return** $\{S_{\Gamma_{ij}}\}_{(i,j)\in\mathcal{I}}$

3.3.3.3 Solve stage

The final stage of the scheme is the solve stage, which uses the merged solution operators to recover the unknown interface data in a downward pass through the hierarchy. Beginning at the top of the hierarchy, the solution operator acting on the entire mesh is applied to the known Dirichlet data g, returning the Chebyshev coefficients of the solution on the top-level merged interface. These coefficients are then used as Dirichlet data on the next level, where solution operators are again applied to compute the unknown interface data on subdomains. Finally, at the bottom level of the hierarchy—where the solution is now known at each interface between elements—the local solution operators $S_{\mathcal{E}_i}$ are applied to compute the bivariate solution in the interior of each element $\mathcal{E}_i$. The solve stage is outlined in Algorithm 3.3.3.

The solve stage may be executed multiple times using different boundary data without recomputing the operators constructed in the initialization and build stages. The stored operators may also be efficiently updated to solve (3.1.1) with a different righthand side. Recall that the last column of every solution operator and Dirichlet-to-Neumann operator in the hierarchy corresponds to the contribution from the particular solution. Using a new righthand side, an updated particular solution can be constructed on each element $\mathcal{E}_i$ as in subsection 3.3.2.1, replacing the last columns of $S_{\mathcal{E}_i}$ and $\Sigma_{\mathcal{E}_i}$. A modified build stage may be then be executed, where the last column of each interfacial solution and Dirichlet-to-Neumann operator is updated by solving the linear system (3.3.6) in an upward pass.

Algorithm 3.3.3 Solve stage (downward pass): $\texttt{solve}(\mathcal{E}, g)$

Input: Element (or merged element) $\mathcal{E}$, Dirichlet data g
Output: Solutions $\{u_i\}_{i=1}^{n_{\text{elem}}}$

if $\mathcal{E}$ is the entire domain **then**
  Get all boundary faces $(\partial\mathcal{E})_i$.
  Evaluate Dirichlet data $g_i := g((\partial\mathcal{E})_i)$ and convert to Chebyshev coefficients.
if $\mathcal{E}$ is a leaf **then**
  Compute the local solution $u := S_{\mathcal{E}} \begin{bmatrix} g \\ 1 \end{bmatrix}$.
  return u
else
  Look up the elements $\mathcal{E}_i$, $\mathcal{E}_j$ that were merged to make $\mathcal{E}$.
  Define the shared interface $\Gamma_{ij} := \mathcal{E}_i \cap \mathcal{E}_j$.
  Recover the missing interface data $\varphi := S_{\Gamma_{ij}} \begin{bmatrix} g \\ 1 \end{bmatrix}$.
  Define vectors $\widehat{\boldsymbol{g}}_i$, $\widehat{\boldsymbol{g}}_j$ such that $\widehat{\boldsymbol{g}}_i^{\Gamma_i} = \varphi$, $\widehat{\boldsymbol{g}}_i^{\mathrm{L}_i} = g_i$ and $\widehat{\boldsymbol{g}}_j^{\Gamma_j} = \varphi$, $\widehat{\boldsymbol{g}}_j^{\mathrm{L}_j} = g_j$.
  Compute the solution on $\mathcal{E}_i$, $\{u_i\} := \texttt{solve}(\mathcal{E}_i, \widehat{\boldsymbol{g}}_i)$.
  Compute the solution on $\mathcal{E}_j$, $\{u_j\} := \texttt{solve}(\mathcal{E}_j, \widehat{\boldsymbol{g}}_j)$.
  return $\{u_i\} \cup \{u_j\}$

3.3.4 Computational complexity

We now determine the computational complexity of the initialization, build, and solve stages in terms of the number of degrees of freedom, $N \approx (p/h)^2$, where h is the minimum mesh size and p is the polynomial order. Here, we assume that the number of elements in the mesh, n_{elem}, scales as $\mathcal{O}(1/h^2)$, which is valid for a mesh that is approximately uniformly refined. For a mesh that is adaptively refined, the number of elements is typically much less than this estimate.

We begin with the initialization stage. On each element $\mathcal{E}_i$, we approximate the solution as a degree-(p,p) polynomial using $(p+1)^2$ degrees of freedom. After transforming the PDE into the local coordinate system of the element, we discretize $\widehat{\mathcal{L}}$ and $\widehat{f}$ using the ultraspherical spectral method. The bivariate Chebyshev coefficients of $\widehat{f}$ can be computed in $\mathcal{O}(p^2 \log p)$ operations via a discrete cosine transform [169]. A separable representation of $\widehat{\mathcal{L}}$ can be computed in $\mathcal{O}(p^3)$ operations using the singular value decomposition, and differentiation, conversion, and multiplication matrices can be constructed for each separable piece in $\mathcal{O}(p)$ operations. The $(p+1)^2 \times (p+1)^2$ discrete PDO can then be assembled using Kronecker products in $\mathcal{O}(p^4)$ operations. The discrete PDO L is almost block-banded, with a bandwidth of $\mathcal{O}(p)$ and $\mathcal{O}(p)$ dense rows. To compute the solution operator $S_{\mathcal{E}_i}$, we must solve a linear system with $\mathcal{O}(p)$ righthand sides. That is, we must solve a system of the form $LX = B$, where L is $\mathcal{O}(p^2) \times \mathcal{O}(p^2)$ and B is $\mathcal{O}(p^2) \times \mathcal{O}(p)$. The almost-banded matrix L may be written as the sum of an $\mathcal{O}(p)$-banded matrix A and a rank-$\mathcal{O}(p)$ correction, $L = A + UCV^T$, where U and V are $\mathcal{O}(p^2) \times \mathcal{O}(p)$ and C is $\mathcal{O}(p) \times \mathcal{O}(p)$. Using the Woodbury formula, the solution to $LX = B$ becomes

$$X = L^{-1}B = \left(A + UCV^T\right)^{-1} B = \left(I \; - \; A^{-1}U\left(C^{-1} + V^T A^{-1} U\right)^{-1} V^T\right) A^{-1}B.$$

The banded matrix A can be inverted in $\mathcal{O}(p^3)$ operations and its inverse applied to $\mathcal{O}(p)$ righthand sides in $\mathcal{O}(p^4)$ operations. The matrix $C^{-1} + V^T A^{-1} U$ is $\mathcal{O}(p) \times \mathcal{O}(p)$ and so its inverse can be applied to $\mathcal{O}(p)$ righthand sides in $\mathcal{O}(p^3)$ operations. Therefore, the solution operator $S_{\mathcal{E}_i}$ on an element can be constructed in $\mathcal{O}(p^4)$ operations. The Dirichlet-to-Neumann operator $\Sigma_{\mathcal{E}_i}$ can be computed as a matrix product in $\mathcal{O}(p^4)$ operations. As these operators are computed once for each element, the overall cost of the initialization stage scales as

$$\frac{p^4}{h^2} \approx Np^2.$$

The cost of the build and solve stages depends on the merge scheme defined by the indices $\mathcal{I}$. If the mesh $\mathcal{E} = \{\mathcal{E}_i\}_{i=1}^{n_{\text{elem}}}$ is approximately tensor-product, the merge indices $\mathcal{I}$ can be defined so that the hierarchy is approximately a binary tree (i.e., a binary tree with $\mathcal{O}(1)$ additional merges). If the mesh is unstructured, a hierarchical partitioning of the mesh may be computed by conversion to a graph partitioning

problem [99]. The partitioning should be as balanced as possible, so that the indices $\mathcal{I}$ define a balanced tree. If the user specifies merge indices that correspond to an unbalanced tree, then the tree may be automatically rebalanced. We assume that the merge indices $\mathcal{I}$ have been given so that the hierarchy in the build and solve stages approximately forms a binary tree with $\mathcal{O}(\log n_{\text{elem}})$ levels.

Let level $\ell = 0$ denote the bottom level of the hierarchy, where no elements have been merged. For a merge between $\mathcal{E}_i$ and $\mathcal{E}_j$ on level ℓ of the build stage, the solution operator $S_{\Gamma_{ij}}$ is computed by solving the linear system (3.3.6). The agglomerates $\mathcal{E}_i$ and $\mathcal{E}_j$ each contain $\mathcal{O}(2^\ell)$ mesh elements, with the interface between them, Γ_{ij}, containing $\mathcal{O}(2^{\ell/2})$ boundaries. Hence, the linear system in (3.3.6) is $\mathcal{O}(2^{\ell/2}p) \times \mathcal{O}(2^{\ell/2}p)$ and can be solved in $\mathcal{O}((2^{\ell/2}p)^3)$ operations. As level ℓ has $\mathcal{O}(2^{-\ell}n_{\text{elem}})$ elements, the cost of processing all merges on level ℓ scales as

$$\left(2^{-\ell}n_{\text{elem}}\right) \cdot \left(2^{\ell/2}p\right)^3 \;=\; n_{\text{elem}}2^{\ell/2}p^3.$$

The total cost for the build stage then scales as

$$p^3 n_{\text{elem}} \sum_{\ell=0}^{\mathcal{O}(\log n_{\text{elem}})} 2^{\ell/2} \;\approx\; p^3 (n_{\text{elem}})^{3/2} \;\approx\; \frac{p^3}{h^3} \;\approx\; N^{3/2}$$

as $N \to \infty$.

At level $\ell > 0$ of the solve stage, the unknown interface data is computed via a matrix-vector multiply with an $\mathcal{O}(2^{\ell/2}p) \times \mathcal{O}(2^{\ell/2}p)$ matrix. As level ℓ has $\mathcal{O}(2^{-\ell}n_{\text{elem}})$ elements, the cost of computing the solution on all interfaces scales as

$$\left(2^{-\ell}n_{\text{elem}}\right) \cdot \left(2^{\ell/2}p\right)^2 \;=\; p^2 n_{\text{elem}}.$$

The total cost for all levels $\ell > 0$ is then

$$\sum_{\ell=1}^{\mathcal{O}(\log n_{\text{elem}})} p^2 n_{\text{elem}} \;\approx\; p^2 n_{\text{elem}} \log n_{\text{elem}}.$$

At the bottom level, $\ell = 0$, the solution is computed on each element through matrix-vector multiplication with local solution operators of size $(p+1)^2 \times (4(p+1)+1)$, which requires $\mathcal{O}(p^3)$ operations. Therefore, the total cost for the solve stage scales as

$$p^2 n_{\text{elem}} \log n_{\text{elem}} + p^3 n_{\text{elem}} \;\approx\; \frac{p^2}{h^2} \log \frac{1}{h^2} + \frac{p^3}{h^2} \;\approx\; N \log \tfrac{1}{h^2} + Np.$$

The overall computational complexity of the method is therefore

$$\underbrace{Np^2}_{\text{initialization stage}} \;+\; \underbrace{N^{3/2}}_{\text{build stage}} \;+\; \underbrace{N \log \tfrac{1}{h^2} + Np}_{\text{solve stage}} \;\approx\; Np^2 + N^{3/2}.$$

As the method stores dense solution operators and Dirichlet-to-Neumann operators on every level of the hierarchy, the total storage cost is analogous to the computational cost of the solve stage. The amount of storage required by the method scales as

$$N \log \tfrac{1}{h^2} + Np.$$

The storage cost can become prohibitive when p is large, as the local solution operators on each element require $\mathcal{O}(p^3 n_{\text{elem}})$ storage. However, these operators need not be constructed and stored. In the initialization stage, local Dirichlet-to-Neumann operators can be constructed directly by locally solving the PDE, evaluating the outward flux, and then discarding the solution. In the solve stage, the solution on the interior of each element can be computed by locally solving the PDE on the fly. This reduces the storage cost to $\mathcal{O}(N \log \frac{1}{h^2})$ while increasing the computational cost of the solve stage to $\mathcal{O}(N \log \frac{1}{h^2} + Np^2)$ operations, but does not change the overall computational complexity of the method.

3.4 Software

We have implemented the ultraspherical SEM in an open-source software package, `ultraSEM`, written in MATLAB without parallelization [67]. An outline of the workflow is depicted in Figure 3.5, and a simple example is shown in Figure 3.6.

The user constructs each element as an `ultraSEM.Domain`, which encodes the coordinate transformations and merge indices local to each element. Convenient functions for constructing rectangles, quadrilaterals, triangles, and polygons are available via the commands `ultraSEM.rectangle`, `ultraSEM.quad`, `ultraSEM.triangle`, and `ultraSEM.polygon`, respectively (see Figure 3.6 (left)), which automatically encode the suitable transformations and merge indices. Elements can be combined to form larger domains by merging them with the '`&`' operator; the merge indices $\mathcal{I}$ will then correspond to the order induced by the sequence of '`&`' operations. More general meshes can be constructed using the `refine(dom)` method (see Figure 3.6 (center)), which performs uniform h-refinement on a given domain `dom`, or the `refinePoint(dom, [x,y])` method, which performs adaptive h-refinement on `dom` around the point (x, y).

A PDO is specified by its coefficients for each derivative, in the form `{{uxx, uxy, uyy}, {ux, uy}, b}`, where each term `uxx`, `uxy`, ... can be a scalar (constant coefficient) or function handle (variable coefficient). The domain and PDO are then passed—along with a righthand side and polynomial order—to construct an `ultraSEM` object (see Figure 3.6 (right)). The `ultraSEM` constructor initializes the local operators on each element (see Algorithm 3.3.1), which are represented as `ultraSEM.Leaf` objects in the hierarchy. The hierarchy of merged operators may then be built in an upward pass via the `build()` command (see Algorithm 3.3.2), which creates a tree of `ultraSEM.Parent` objects (if `build()` is not explicitly called, the build stage is automatically performed when the user requests a solve to be executed). The solve

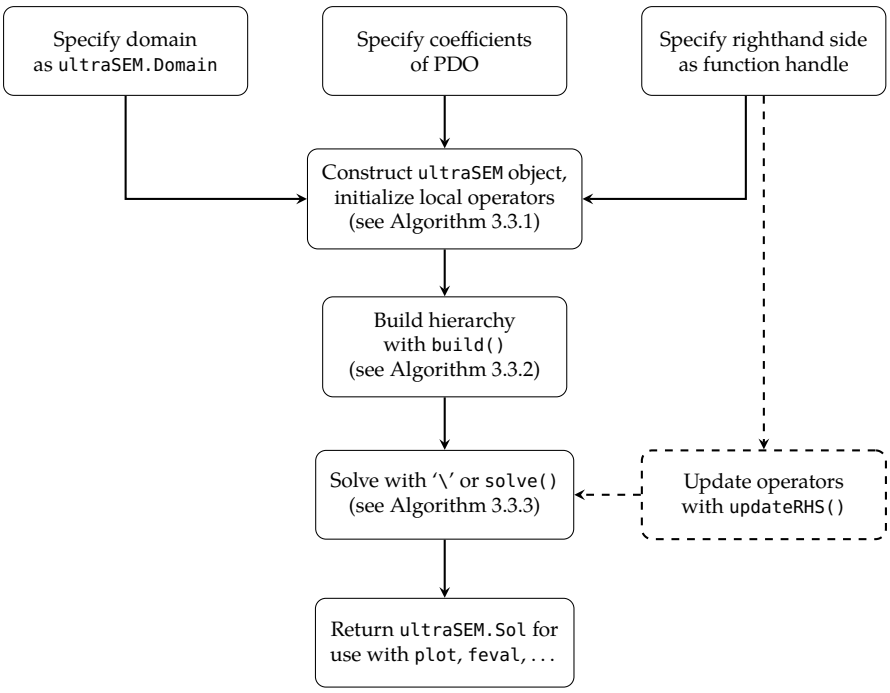


Figure 3.5: *A diagram of the code workflow in* `ultraSEM`*. The code is designed to mirror the steps of the hierarchical Poincaré–Steklov scheme.*

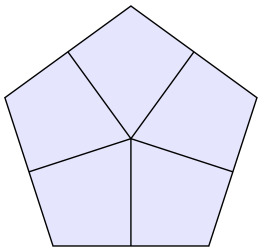
stage is invoked via the `solve()` command (or equivalently, the '\' operator), which computes the solution by applying the hierarchy of operators in a downward pass (see Algorithm 3.3.3). The solution is returned as an `ultraSEM.Sol` object, which overloads a host of functions for plotting (e.g., `plot`, `contour`) and evaluation (e.g., `feval`, `norm`).

An `ultraSEM` object that has been initialized and built can be repeatedly applied to new boundary conditions by invoking `solve()` multiple times. The object can also be cheaply updated to solve with a new righthand side by calling `updateRHS()`, which alters the last column of each operator in the hierarchy to correspond to a new particular solution.

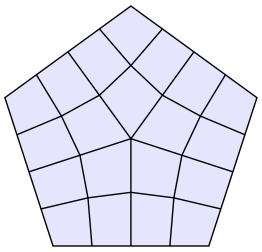
3.5 Numerical results

3.5.1 Computational complexity

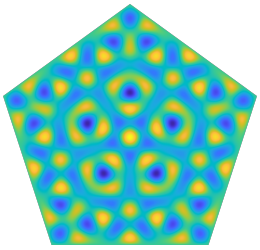
To illustrate the computational complexity of `ultraSEM`, we measure the execution times of the initialization, build, and solve stages of the method under uniform h- and p-refinement. Figure 3.7 shows the recorded timings for solving the variable coefficient PDE $\nabla^2 u + \sin(xy)u = f$ on the domain $\Omega = [0,1]^2$ with a spatially varying righthand and spatially varying Dirichlet boundary conditions.



```
dom = ultraSEM.polygon(5);
plot(dom)
```



```
dom = refine(dom);
plot(dom)
```



```
p = 20; rhs = -1; bc = 0;
pdo = {{1,0,1}, {0,0}, 1000};
S = ultraSEM(dom, pdo, rhs, p);
u = S \ bc;
plot(u)
```

Figure 3.6: *A simple example of the syntax in* `ultraSEM`*. A pentagonal domain (with side length 1.2) is meshed into five quadrilaterals (left) and uniformly refined (center). The Helmholtz equation* $\nabla^2 u + 1000u = -1$ *with zero Dirichlet boundary conditions is then solved on the mesh using polynomials of degree 20 on each element (right).*

In Figure 3.7 (left), the polynomial order is fixed at $p = 4$ and a Cartesian mesh with $\mathcal{O}(1/h^2)$ elements is successively refined. The initialization and build stages both exhibit $\mathcal{O}(1/h^2)$ scaling as $h \to 0$, while the solve stage scales as $\mathcal{O}(1/h^2 \log(1/h^2))$. The timings for the build stage do not exhibit the expected $\mathcal{O}(1/h^3)$ scaling. This is likely due to the fact that the build stage relies on dense linear algebra routines that have been heavily optimized for the relatively small $\mathcal{O}(1/h) \times \mathcal{O}(1/h)$ matrices tested here.

In Figure 3.7 (right), the Cartesian mesh is fixed to have 4×4 elements and the polynomial order p is successively increased. The cost of the initialization stage dominates, exhibiting close to the expected $\mathcal{O}(p^4)$ scaling as $p \to \infty$. The build and solve stages perform better than expected, both exhibiting $\mathcal{O}(p^2)$ scaling. Again, this can likely be attributed to the performance of dense linear algebra routines in the regime of p tested.

3.5.2 Convergence and hp-adaptivity

We now investigate the convergence properties of `ultraSEM` with respect to the mesh size h and polynomial order p. As a test problem, we consider solving the Helmholtz equation,

$$\nabla^2 u + (\sqrt{2}\omega)^2 u = 0, \qquad u \in [-1, 1]^2, \tag{3.5.1}$$

with $\omega \in \mathbb{R}$ and Dirichlet boundary conditions given so that the exact solution is $u(x, y) = \cos(\omega x)\cos(\omega y)$. To measure convergence over a range of polynomial orders, we set $\omega = p$ so that the number of degrees of freedom per wavelength remains fixed independent of p. We then solve (3.5.1) under uniform h-refinement. Figure 3.8 shows the relative error in the L^2 norm as $h \to 0$ for polynomial orders

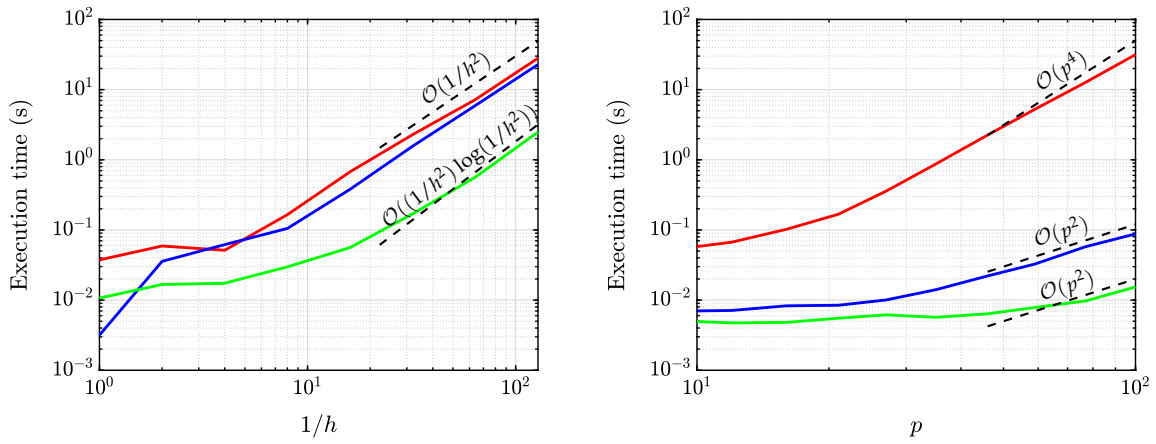


Figure 3.7: *Execution time (in seconds) for* `ultraSEM` *over a range of mesh sizes (left) and polynomial orders (right). Timings are depicted for the initialization stage (red), build stage (blue), and solve stage (green), when solving the PDE* $\nabla^2 u + \sin(xy)u = f$ *on the domain* $\Omega = [0,1]^2$ *with spatially varying righthand side and spatially varying Dirichlet boundary conditions. On the left, we successively refine a Cartesian mesh while keeping the polynomial order fixed at* $p = 4$*. On the right, we use a* 4×4 *Cartesian mesh while successively increasing the polynomial order.*

$p = 5$, $p = 10$, and $p = 30$. The convergence rate is observed to be $\mathcal{O}(h^{p-1})$. If error is measured in the H^1 or H^2 norm, where H^k denotes the Sobolev space of functions whose weak derivatives up to order k are in L^2, then the convergence rate is similarly $\mathcal{O}(h^{p-1})$. Since our method is sparse with respect to p, the exact rate of convergence is not so important, as a degree-p discretization may easily be replaced by a degree-$(p+1)$ discretization with minimal increase in computational cost.

In general, the mesh size h and polynomial order p need not be the same on each element. Adaptive h-refinement can be performed on each element locally; however, subdividing an element may give rise to meshes with hanging nodes (i.e., nodes of the mesh which occur in the middle of an element's face). While hanging nodes may be handled in the hierarchical Poincare–Steklov scheme through the use of interpolation operators [74], we choose to avoid them here. To avoid hanging nodes, `ultraSEM` performs h-refinement in a conforming way around specified corners or points, by subdividing a quadrilateral element into three or five children, respectively (see Figure 3.9).

The ultraspherical spectral element method can naturally perform p-adaptivity by applying local interpolation and restriction operators to the elemental matrices involved in each merge operation. Since each unknown interface function is represented by a vector of Chebyshev coefficients, interpolation to and restriction from an interface can be performed simply by zero-padding or truncating the interface data. The polynomial order on an interface can be defined in a variety of ways. Popular choices include the minimum rule and maximum rule [55]; we employ the minimum rule here, which sets the polynomial order on an interface to be the minimum of the polynomial orders on the adjacent elements.

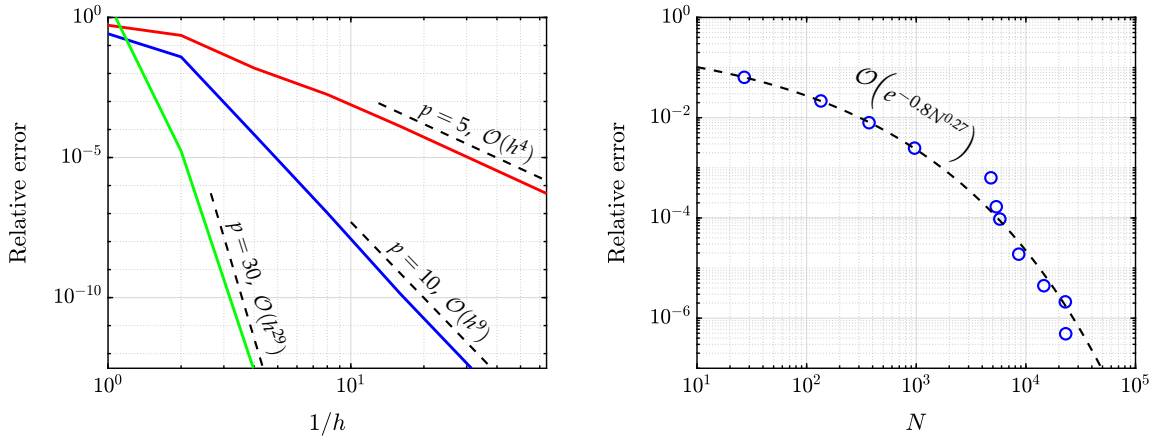


Figure 3.8: *Convergence of* `ultraSEM` *with respect to h and p. (Left) Relative error in the* L^2 *norm when solving* (3.5.1) *under uniform h-refinement, for* $p = 5$ *(red),* $p = 10$ *(blue), and* $p = 30$ *(green). To measure convergence over a range of polynomial orders, we set* $\omega = p$ *so that the number of degrees of freedom per wavelength remains fixed independent of p. In each case,* $\mathcal{O}(h^{p-1})$ *convergence is observed. (Right) An* a priori hp*-adaptivity strategy is applied to the L-shape problem in* (3.5.2)*. The relative error decays super-algebraically in the total number of degrees of freedom N.*

Figure 3.9: *To avoid hanging nodes,* `ultraSEM` *performs h-refinement in a conforming way. (Left) A square is successively refined into a corner by subdividing into three children per refinement level. (Right) A square is successively refined around a point by subdividing into five children per refinement level.*

We now consider the application of an hp-adaptivity strategy to the classical L-shape domain problem [118],

$$\nabla^2 u = 0, \qquad u \in [-1,1]^2 \setminus [0,1] \times [-1,0], \tag{3.5.2}$$

with Dirichlet boundary conditions given so that the exact solution is $u(r,\theta) = r^{2/3}\sin(2\theta/3)$, where $r = \sqrt{x^2+y^2}$ and $\theta = \tan^{-1}(y/x)$. The reentrant corner of the domain induces a singularity in the solution so that $u \in H^{1+2/3}$ near the origin, where $\alpha = 1 + 2/3$ is the largest α such that $u \in H^\alpha$. Therefore, any strategy based on uniform h- or p-refinement is necessarily restricted to algebraic convergence[17] [19].

[17] For this Laplace problem, alternative methods may provide higher accuracy per degree of freedom than element methods. For instance, root-exponential convergence in the supremum norm can be achieved by representing the solution as the real part of a rational function with poles exponentially clustered near each corner [81].

That is, for a numerical solution u_{hp} based on uniform refinement, the error can be bounded *a priori* by

$$\|u - u_{hp}\|_{L^2} \leq \|u - u_{hp}\|_{H^1} \leq C \left(\frac{h}{p}\right)^{2/3} \|u\|_{H^{1+2/3}},$$

for some constant $C > 0$. However, by employing a suitable hp-adaptivity strategy, super-algebraic convergence in the number of degrees of freedom N can be achieved [18], i.e.,

$$\|u - u_{hp}\|_{L^2} \leq \|u - u_{hp}\|_{H^1} \leq C_1 e^{-C_2 N^{1/3}},$$

for some constants $C_1, C_2 > 0$. Here we employ an *a priori* adaptivity strategy, where h-refinement is performed into the reentrant corner on elements adjacent to the origin and p-refinement is performed on all other elements [2]. Given a desired relative error tolerance and an initial coarse hp-mesh, an automatic hp-adaptivity loop is run that successively refines or coarsens each element in h or p based on an *a posteriori* error indicator [1, 119]. Here, we compute the element-wise error from the exact solution as a surrogate for a true *a posteriori* error indicator. Figure 3.8 (right) shows the relative error in the L^2 norm versus the total number of degrees of freedom N in the adaptive hp-mesh for a sequence of error tolerances. Super-algebraic convergence to the solution is observed as the number of degrees of freedom increases. A least-squares fit to the data gives an approximate convergence rate of $\mathcal{O}(e^{-0.8N^{0.27}})$. To illustrate the range of h and p used on a given mesh, for a relative error tolerance of 10^{-6} the final mesh contains 15 levels of corner h-refinement (see Figure 3.9 (left)) and polynomial orders ranging from 3 (near the reentrant corner) to 13 (away from the reentrant corner).

As a practical example of hp-adaptivity, we consider using `ultraSEM` on a domain with small-scale geometric features along its boundary. The domain Ω is a snowflake shape created by a fractal-like Penrose tiling (see Figure 3.10 (left)). We construct a mesh of 4,568 quadrilaterals over Ω using the meshing software Gmsh [75], with the element size constrained to be smaller near the boundary and larger in the interior. To specify a p-adaptive discretization, we define a function that varies smoothly from $p = 40$ in the center of Ω to $p = 7$ near the boundary, indicating that coarse elements in the interior of Ω employ a high-p discretization while fine elements close to the boundary of Ω employ a lower p. The total number of degrees of freedom for this hp-mesh is $N = 333{,}627$. We locate the domain Ω such that $y < 0$ for all $(x, y) \in \Omega$, and solve the gravity Helmholtz equation

$$\nabla^2 u + 100(1 - y)u = -1, \qquad u \in \Omega, \tag{3.5.3}$$

with zero Dirichlet boundary conditions. The computation in `ultraSEM` takes about 90 seconds. The computed solution is shown in Figure 3.10 (right). The h-adaptive nature

of the discretization allows for the small-scale geometry of the domain boundary to be resolved without using a prohibitive number of elements, while the p-adaptive nature of the discretization allows for the high-degree approximation of smooth functions on coarse elements.

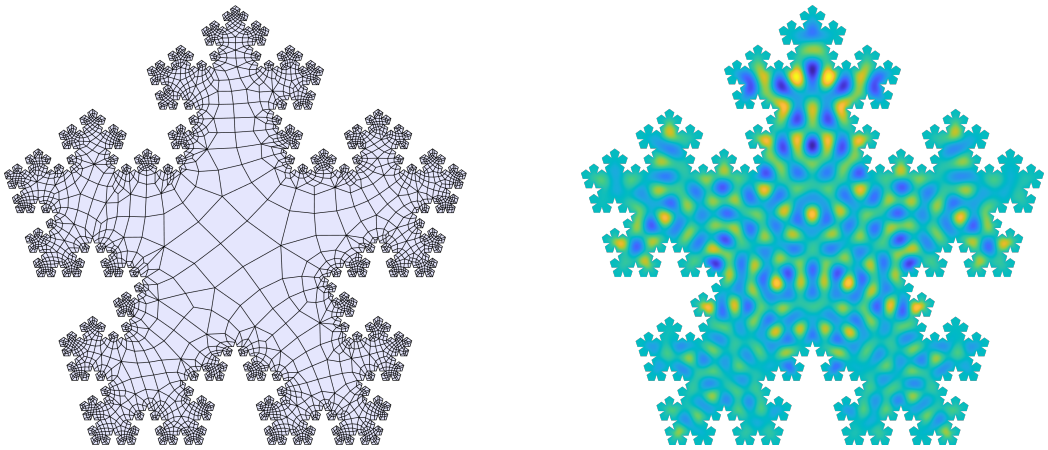


Figure 3.10: *(Left) A snowflake-shaped domain created by a Penrose tiling (with side length 0.066) is adaptively meshed with 4,658 elements, with p varying from $p = 7$ on small elements to $p = 40$ on large elements. (Right) The gravity Helmholtz equation* (3.5.3) *is solved on this domain. The solution is represented by $N = 333{,}627$ degrees of freedom.*

3.5.3 Implicit time-stepping for parabolic problems

The ability to reuse precomputed solution operators allows for efficient implicit time-stepping for parabolic problems. To demonstrate, we consider solving the variable-coefficient convection-diffusion equation on the domain $\Omega = [0, 10] \times [-1, 1]$ over the time span $[0, T]$,

$$\frac{\partial u}{\partial t} = \kappa \nabla^2 u - \nabla \cdot (\boldsymbol{b}(x, y) u), \qquad u \in \Omega \times [0, T], \tag{3.5.4}$$

for $T > 0$, with initial condition $u(x, y, 0) = e^{-4(x-1)^2 - 4y^2}$ and zero Dirichlet boundary conditions. This equation models the transport of a contaminant concentration in a flow. We define the diffusivity $\kappa = 0.01$ and the convective velocity $\boldsymbol{b}(x, y) = (1 - e^{\gamma x} \cos 2\pi y, \frac{\gamma}{2\pi} e^{\gamma x} \sin 2\pi y)$. Here, the velocity field $\boldsymbol{b}(x, y)$ is the analytical solution to the Kovasznay flow [101], where $\gamma = \mathrm{Re}/2 - \sqrt{\mathrm{Re}^2/4 - 4\pi^2}$ and $\mathrm{Re} = 100$ is the Reynolds number.

Define the time step $\Delta t = 0.1$ and time points $t_n = n\Delta t$ for integers $n \geq 0$, and let u^n denote the approximate solution to (3.5.4) at time t_n. Discretizing in time using the backward Euler method yields a steady-state PDE in u^{n+1},

$$u^{n+1} - \Delta t \, \kappa \nabla^2 u^{n+1} + \Delta t \, \nabla \cdot \left(\boldsymbol{b} u^{n+1} \right) = u^n, \tag{3.5.5}$$

which must be solved once per time step to compute u^{n+1} from u^n. We use `ultraSEM` to solve (3.5.5) on a 2×10 Cartesian mesh of Ω with polynomial order $p = 10$ on each element. Figure 3.11 (left) shows snapshots of the computed solution at times $t = 0$, $t = 1$, and $t = 5$. As the righthand side of (3.5.5) depends on n, the operators in `ultraSEM` must be updated at each time step. If the operators are reconstructed from scratch at each time step, simulating to time $T = 5$ completes in roughly 9 seconds (see Figure 3.11 (right, red)). If instead only the particular solution is reconstructed using `updateRHS()`, then the same simulation completes in less than a second (see Figure 3.11 (right, blue)). Figure 3.11 (right) compares the execution times required to simulate (3.5.4) over the time span $[0, T]$ using these two methods. It is clear that when many time steps are taken, `updateRHS()` should always be used.

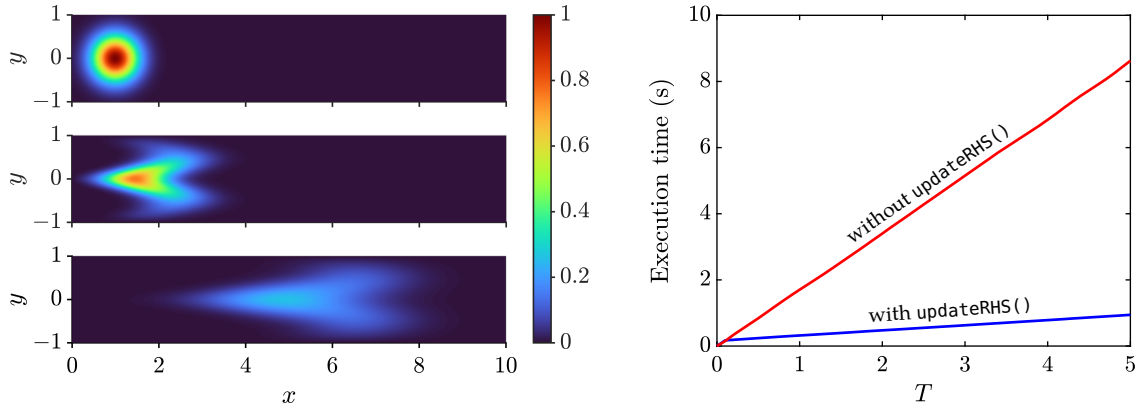


Figure 3.11: *(Left) Snapshots of the solution to the convection-diffusion equation* (3.5.4) *at times* $t = 0$, $t = 1$, *and* $t = 5$, *computed using* `ultraSEM` *in space and backward Euler in time. (Right) The execution time required to simulate* (3.5.4) *over the time span* $[0, T]$, *by either reconstructing the operators from scratch at each time step (red) or updating the particular solution using* `updateRHS()` *(blue).*

Chapter 4

Efficient operator-coarsening multigrid schemes for local discontinuous Galerkin methods†

4.1 Introduction

Discontinuous Galerkin (DG) methods have gained broad popularity in recent years. They are well-suited to hp-adaptivity, provide high-order accuracy, and can be applied to a wide range of problems on complex geometries with unstructured meshes. Although DG methods were first applied to the discretization of hyperbolic conservation laws, they have been extended to handle elliptic problems and diffusive operators in a unified framework [11]. Such methods include the symmetric interior penalty (SIP) method [10,57], the Bassi–Rebay (BR1, BR2) methods [25,26], the local discontinuous Galerkin (LDG) method [51], the compact discontinuous Galerkin (CDG) method [137], the line-based discontinuous Galerkin method [138], and the hybridizable discontinuous Galerkin (HDG) method [48]. In particular, the development of efficient solvers for DG discretizations of elliptic problems is an active area of research.

Among the panoply of DG methods for elliptic problems, the LDG method is a popular choice: it is accurate, stable, simple to implement, and extendable to higher-order derivatives [179]. Additionally, on Cartesian grids it has been shown to be superconvergent [49]. The LDG method results in symmetric positive (semi)definite discretizations which are well-suited to solution by efficient iterative methods. In particular, the multigrid method has emerged as a natural candidate due to its success in the continuous finite element and finite difference communities, both as a standalone solver and as a preconditioner for the conjugate gradient (PCG) method. However, direct application of standard multigrid techniques to DG discretizations of elliptic problems can result in suboptimal performance when inherited bilinear forms

†This chapter is a reformatted version of the following jointly authored publication: D. Fortunato, C. H. Rycroft, and R. Saye, *Efficient operator-coarsening multigrid schemes for local discontinuous Galerkin methods*, SIAM J. Sci. Comput., 41 (2019), pp. A3913–A3937, https://doi.org/10.1137/18M1206357.

are employed [5, 82], and much work has gone into developing specialized smoothers and coarse-correction methods to remedy this issue, even on Cartesian grids [63, 97].

When a mesh hierarchy is available, geometric h-multigrid is a natural choice of solver. Error estimates have been derived for a multilevel interior penalty (IP) method on unstructured meshes, yielding convergence factors in the range $\rho \approx 0.3$–0.5 for Poisson's equation [35, 82]; the method has also been applied to adaptively refined Cartesian grids with similar results [98]. Subsequent work describes how the multilevel IP method [82] can be used as a preconditioner for the LDG method—both for the Schur complement system (using conjugate gradient) and for the saddle-point system (using GMRES)—resulting in a bounded condition number with respect to mesh size h [97]. More recent work for LDG and IP uses a multigrid W-cycle on nested [5] and agglomerated [6] unstructured meshes; however, these results indicate poor convergence factors of $\rho \approx 0.8$–0.9 even with many smoothing steps. On non-nested polygonal meshes, h-independent iteration counts with convergence factors $\rho \approx 0.2$–0.3 are achieved for SIP using an additive Schwarz smoother with 3–8 smoothing steps per V-cycle [8].

A popular choice for high-order DG methods is p- or hp-multigrid [24, 65, 89, 109], where p refers to coarsening the polynomial degree p in the multilevel hierarchy, and hp refers to some combined strategy of coarsening the mesh size h as well as the polynomial degree p of the underlying discretization. Using factor-of-two coarsening in p with an element Jacobi smoother, convergence factors of $\rho \approx 0.5$ were achieved for Laplace's equation with $p \leq 4$ [89]. Fidkowski et al. [65] used a line smoother with sequential coarsening in p that gave similar results for convection–diffusion problems, though the performance degraded as $h \to 0$. A method employing an overlapping Schwarz smoother with factor-of-two coarsening was used with PCG on LDG discretizations up to $p = 32$ [158]; this method exhibited good convergence factors of $\rho \leq 0.1$ on high-aspect-ratio Cartesian grids, at the cost of an expensive smoother.

Algebraic multigrid methods have also been applied to DG discretizations of elliptic problems. A hierarchy of operators can be defined by agglomerating neighboring unknowns based on smoothed aggregation (SA), resulting in average convergence factors of $\rho \approx 0.4$ and $\rho \approx 0.2$ for the bilinear BR2 and SIP methods, respectively [141]. An SA method employing energy minimization was used with PCG to achieve h-independent convergence factors of $\rho \approx 0.2$ for LDG discretizations, but performance degraded with increasing p [122]. A method based on unsmoothed aggregation was developed for the IP method using a coarse space consisting of continuous linear basis functions [27]; this method proved robust for multi-phase problems with large jumps in ellipticity coefficient, but efficiency weakly degraded with mesh size. A related approach based on smoothed aggregation and low-order coarse grid correction yielded similar results [156].

Independent of the type of multigrid method, a particular fact to note—and something we believe underpins the difficulties in applying multigrid to DG—is that coarsening a fine-grid operator is not always the same as constructing that operator

directly from the coarse grid. Indeed, it was noted by Antonietti et al. [5] that for all stable and strongly consistent DG methods, "convergence cannot be independent of the number of levels if inherited bilinear forms are considered (i.e., the coarse solvers are the restriction of the stiffness matrix constructed on the finest grid)." Furthermore they noted that non-inherited forms must be employed for the multigrid method to be scalable. In the context of two-level methods, the reason for this loss of scalability is known [4]. In this chapter, for the LDG method, we present a simple modification to traditional multigrid operator coarsening that yields optimal multigrid convergence and can be extended to other DG discretizations of elliptic problems. We confirm that traditional coarsening of the fine-mesh elliptic operator results in poor performance, and show that the coarsening of the saddle-point flux formation restores optimal multigrid efficiency. Our approach is equivalent to pure geometric multigrid but avoids the need to explicitly build the coarse mesh and its associated components, such as quadrature rules, Jacobian mappings, lifting operators, and face-to-element enumerations—as discussed, this holds benefit for a variety of intricate DG implementations where building the coarse mesh can be problematic. Nevertheless we point out that in the pure geometric multigrid setting, quadrature-free DG methods [7] have recently been proposed which avoid the construction of coarse mesh quadrature rules.

This chapter is structured as follows. In section 4.2, we formulate a general DG discretization of Poisson's equation and derive the LDG method through the appropriate choice of numerical flux. In section 4.3, we describe the construction of geometric hp-multigrid methods in the corresponding DG setting. In particular, we show that traditional operator coarsening can fail to create the coarse operator resulting from faithful rediscretization in a pure geometric multigrid setting for LDG methods, and present a modified coarsening strategy that remedies this. In section 4.4, we present numerical results for the standard and modified multigrid methods on uniform and adaptively-refined Cartesian grids in 2D and 3D. We conclude with some examples of multi-phase elliptic interface problems on implicitly defined meshes, which demonstrate good multigrid performance even on meshes with long and thin filaments as well as tiny and dispersed phase components.

4.2 Discontinuous Galerkin formulation

4.2.1 Model problem

The model elliptic problem considered in this work is the Poisson problem

$$\begin{aligned} -\nabla^2 u &= f \quad \text{in } \Omega, \\ u &= g \quad \text{on } \Gamma_D, \\ \nabla u \cdot \boldsymbol{n} &= h \quad \text{on } \Gamma_N, \end{aligned} \tag{4.2.1}$$

where Ω is a domain in $\mathbb{R}^d$, Γ_D and Γ_N denote the components of $\partial\Omega$ on which Dirichlet and Neumann boundary conditions are imposed, $\boldsymbol{n}$ is the outward unit normal to the boundary, and f, g, and h are given functions defined on Ω and its boundary.

4.2.2 DG for elliptic problems

In order to apply a DG method to (4.2.1), we rewrite it as a first-order system by introducing the auxiliary variable $\boldsymbol{q} = \nabla u$ and writing the Laplacian as the divergence of $\boldsymbol{q}$ [11]:

$$\begin{aligned} \boldsymbol{q} &= \nabla u \quad && \text{in } \Omega, \\ -\nabla \cdot \boldsymbol{q} &= f \quad && \text{in } \Omega, \\ u &= g \quad && \text{on } \Gamma_D, \\ \boldsymbol{q} \cdot \boldsymbol{n} &= h \quad && \text{on } \Gamma_N. \end{aligned} \tag{4.2.2}$$

In this work, we mainly consider discretizations of (4.2.2) wherein the corresponding meshes arise from Cartesian grids, quad/octrees, or implicitly defined meshes of more complex curved domains (see subsections 4.4.1, 4.4.3, and 4.4.4, respectively). As such, it is natural to adopt a tensor-product piecewise polynomial space. Let $\mathcal{E} = \bigcup_i E_i$ denote the set of elements of a mesh of Ω, let $p \geq 1$ be an integer, and define $\mathcal{Q}_p(E)$ to be the space of tensor-product polynomials of degree p on the element E. For example, $\mathcal{Q}_3$ is the space of bicubic (in 2D) or tricubic (in 3D) polynomials having 16 or 64 degrees of freedom, respectively. We define the corresponding spaces of discontinuous piecewise polynomials and vector fields on the mesh as

$$V_h(\mathcal{E}) = \left\{ v : \Omega \to \mathbb{R} \;\middle|\; v|_E \in \mathcal{Q}_p(E) \text{ for every } E \in \mathcal{E} \right\}, \tag{4.2.3}$$

$$V_h^d(\mathcal{E}) = \left\{ \boldsymbol{\omega} : \Omega \to \mathbb{R}^d \;\middle|\; \boldsymbol{\omega}|_E \in [\mathcal{Q}_p(E)]^d \text{ for every } E \in \mathcal{E} \right\}, \tag{4.2.4}$$

respectively. We denote by $(\cdot, \cdot)$ the natural L^2 inner product on V_h and by $\|\cdot\|$ the corresponding norm, $\|u\|^2 = (u, u)$, with analogous definitions for V_h^d.

In a DG method, both $\boldsymbol{q}$ and its divergence are defined weakly via numerical fluxes defined on each mesh face. The weak form of (4.2.2) consists of finding $(\boldsymbol{q}_h, u_h) \in V_h^d \times V_h$ such that

$$\int_E \boldsymbol{q}_h \cdot \boldsymbol{\omega} = -\int_E u_h \nabla \cdot \boldsymbol{\omega} + \int_{\partial E} \widehat{u}_h \, \boldsymbol{\omega} \cdot \boldsymbol{n}, \tag{4.2.5}$$

$$\int_E \boldsymbol{q}_h \cdot \nabla v - \int_{\partial E} \widehat{\boldsymbol{q}}_h \, v \cdot \boldsymbol{n} = \int_E f \, v, \tag{4.2.6}$$

for all test functions $(\boldsymbol{\omega}, v) \in [\mathcal{Q}_p(E)]^d \times \mathcal{Q}_p(E)$ and for all $E \in \mathcal{E}$. The numerical fluxes $\widehat{\boldsymbol{q}}_h$ and $\widehat{u}_h$ are approximations to $\boldsymbol{q}_h$ and u_h, respectively, on each mesh face and define how the degrees of freedom in each element are coupled together.

To more succinctly describe the coupling between elements, the following standard notation is adopted. Consider two adjacent elements E^+ and E^- which share a face in $\mathcal{E}$. Let $\boldsymbol{n}^\pm$ denote the outward unit normals of $\partial E^\pm$ along the shared face and $(\boldsymbol{\omega}^\pm, v^\pm)$ denote the traces of $(\boldsymbol{\omega}, v) \in V_h^d \times V_h$ from $E^\pm$ on the shared face. The average $\{\!\{\cdot\}\!\}$ and jump $[\![\cdot]\!]$ operators on the shared face are then defined as

$$\begin{aligned} \{\!\{\boldsymbol{\omega}\}\!\} &= \tfrac{1}{2}(\boldsymbol{\omega}^+ + \boldsymbol{\omega}^-), & \{\!\{v\}\!\} &= \tfrac{1}{2}(v^+ + v^-), \\ [\![\boldsymbol{\omega}]\!] &= \boldsymbol{\omega}^+ \cdot \boldsymbol{n}^+ + \boldsymbol{\omega}^- \cdot \boldsymbol{n}^-, & [\![v]\!] &= v^+ \boldsymbol{n}^+ + v^- \boldsymbol{n}^-. \end{aligned}$$

On boundary faces, $(\boldsymbol{\omega}^-, v^-)$ shall refer to the traces of $(\boldsymbol{\omega}, v)$ from the corresponding element touching $\partial\Omega$.

4.2.3 The local discontinuous Galerkin method

The choice of numerical flux in (4.2.5)–(4.2.6) defines a DG method. Here we focus on the LDG method [51], which chooses numerical fluxes $\widehat{\boldsymbol{q}}_h$ and $\widehat{u}_h$ according to the general form

$$\widehat{\boldsymbol{q}}_h = \begin{cases} \{\!\{\boldsymbol{q}_h\}\!\} + \boldsymbol{\beta}[\![\boldsymbol{q}_h]\!] - \tau_0[\![u_h]\!] & \text{on any interior face,} \\ \boldsymbol{q}_h^- - \tau_D(u_h^- - g)\boldsymbol{n} & \text{on any face of } \Gamma_D, \\ h\,\boldsymbol{n} & \text{on any face of } \Gamma_N, \end{cases} \tag{4.2.7}$$

and

$$\widehat{u}_h = \begin{cases} \{\!\{u_h\}\!\} - \boldsymbol{\beta} \cdot [\![u_h]\!] & \text{on any interior face,} \\ g & \text{on any face of } \Gamma_D, \\ u_h^- & \text{on any face of } \Gamma_N, \end{cases} \tag{4.2.8}$$

where $\boldsymbol{\beta}$ is a (possibly face-dependent) user-defined vector; for example, in a one-sided flux scheme, $\boldsymbol{\beta} = \pm\frac{1}{2}\boldsymbol{n}$. Here, the numerical flux $\widehat{\boldsymbol{q}}_h$ includes penalty stabilization terms; $\tau_0 \geq 0$ is a penalty parameter associated with interior faces and $\tau_D > 0$ is associated with Dirichlet boundary faces (if any). Generally, τ_0 must be strictly positive to ensure well-posedness of the discrete problem, but in some cases (e.g., on Cartesian grids with particular choices of $\boldsymbol{\beta}$), τ_0 can be set equal to zero [47]. If Γ_D is nonempty, τ_D must be positive to ensure well-posedness of the final discrete problem. To be consistent with the scaling of penalty parameters in other DG methods, we choose the penalty parameters to scale inversely with the element size h [44] so that $\tau_0 = \widetilde{\tau}_0/h$ and $\tau_D = \widetilde{\tau}_D/h$ where $\widetilde{\tau}_0$ and $\widetilde{\tau}_D$ are constants.[18] Furthermore, although arbitrarily small choices of $\widetilde{\tau}_0$ and $\widetilde{\tau}_D$ suffice to ensure well-posedness, later we show that a carefully considered choice of these values can greatly benefit multigrid performance (see subsection 4.4.2).

[18] To obtain uniform stability in the limit of large p, τ_0 should also scale with p^2 [139]; we do not consider this aspect for the moderate values of p tested in this work ($p = 1$–8).

We first particularize (4.2.5) for the LDG method, which is the weak statement that $\boldsymbol{q} = \nabla u$. We slightly modify the weak form (4.2.5) by defining $\boldsymbol{q}_h \in V_h^d$ in strong-weak form,[19] such that

$$\int_E \boldsymbol{q}_h \cdot \boldsymbol{\omega} = \int_E \nabla u_h \cdot \boldsymbol{\omega} + \int_{\partial E} (\widehat{u}_h - u_h)\, \boldsymbol{\omega} \cdot \boldsymbol{n} \tag{4.2.9}$$

holds for every element $E \in \mathcal{E}$ and every test function $\boldsymbol{\omega} \in [\mathcal{Q}_p(E)]^d$. Upon summing (4.2.9) over every element of the mesh and using the definition of the numerical flux $\widehat{u}_h$ in (4.2.8), we have that, for any $\boldsymbol{\omega} \in V_h^d$,

$$\int_\Omega \boldsymbol{q}_h \cdot \boldsymbol{\omega} = \sum_{E \in \mathcal{E}} \int_E \nabla u_h \cdot \boldsymbol{\omega} - \int_{\Gamma_0} [\![u_h]\!] \cdot (\{\!\{ \boldsymbol{\omega} \}\!\} + \boldsymbol{\beta} [\![\boldsymbol{\omega}]\!]) + \int_{\Gamma_D} (g - u_h^-) \boldsymbol{\omega}^- \cdot \boldsymbol{n}, \tag{4.2.10}$$

where Γ_0 denotes the union of all interior faces of $\mathcal{E}$. Define the following operators:

- Let $\nabla_h : V_h \to V_h^d$ be the *broken gradient operator* and $L : V_h \to V_h^d$ be the *lifting operator*, such that

  $$\int_\Omega (\nabla_h u) \cdot \boldsymbol{\omega} = \sum_{E \in \mathcal{E}} \int_E \nabla u \cdot \boldsymbol{\omega},$$
  $$\int_\Omega (Lu) \cdot \boldsymbol{\omega} = -\int_{\Gamma_0} [\![u]\!] \cdot (\{\!\{ \boldsymbol{\omega} \}\!\} + \boldsymbol{\beta} [\![\boldsymbol{\omega}]\!]) - \int_{\Gamma_D} u^- \boldsymbol{\omega}^- \cdot \boldsymbol{n}$$

  holds for every $\boldsymbol{\omega} \in V_h^d$ and each $u \in V_h$.

- Define $J_D(g) \in V_h^d$ such that

  $$\int_\Omega J_D(g) \cdot \boldsymbol{\omega} = \int_{\Gamma_D} g\, \boldsymbol{\omega}^- \cdot \boldsymbol{n}$$

  holds for every $\boldsymbol{\omega} \in V_h^d$.

Accordingly, (4.2.10) is equivalent to the statement that

$$\boldsymbol{q}_h = (\nabla_h + L) u_h + J_D(g) = G u_h + J_D(g) \tag{4.2.11}$$

where $G : V_h \to V_h^d$ is the *discrete gradient operator*, $G = \nabla_h + L$. The formula (4.2.11) is the LDG discretization of the statement $\boldsymbol{q} = \nabla u$, taking into account Dirichlet boundary data.

[19] The *strong-weak form* states that $\boldsymbol{q}_h$ must satisfy $\int_E \boldsymbol{q}_h \cdot \boldsymbol{\omega} = \int_E \nabla u_h \cdot \boldsymbol{\omega} + \int_{\partial E} (\widehat{u}_h - u_h)\, \boldsymbol{\omega} \cdot \boldsymbol{n}$ whereas the *weak form* states that $\boldsymbol{q}_h$ must satisfy $\int_E \boldsymbol{q}_h \cdot \boldsymbol{\omega} = -\int_E u_h \nabla \cdot \boldsymbol{\omega} + \int_{\partial E} \widehat{u}_h\, \boldsymbol{\omega} \cdot \boldsymbol{n}$. The two forms are equivalent whenever the employed quadrature scheme exactly satisfies the identity of integration by parts, which in practice is generally true for quadrilateral, prismatic, simplicial elements, etc., but is generally not true when approximate numerical quadrature schemes are used, e.g., as on implicitly defined curved elements. In the latter situation, to ensure symmetry of the final discrete Laplacian operator, it is necessary to use the strong-weak form to define $\boldsymbol{q}_h$ and the weak form to define the divergence of $\boldsymbol{q}_h$ (or vice versa) [151].

Next, we particularize (4.2.6) for the LDG method, which is the weak statement that $-\nabla \cdot \boldsymbol{q} = f$. Upon summing (4.2.6) over every mesh element and using the definition of the numerical flux $\widehat{\boldsymbol{q}}_h$ in (4.2.7), we have that, for any $v \in V_h$,

$$\begin{aligned}\sum_{E\in\mathcal{E}} \int_E \boldsymbol{q}_h \cdot \nabla v - \int_{\Gamma_0} (\{\!\{\boldsymbol{q}_h\}\!\} + \boldsymbol{\beta}[\![\boldsymbol{q}_h]\!] - \tau_0 [\![u_h]\!]) \cdot [\![v]\!] - \int_{\Gamma_D} (\boldsymbol{q}_h^- \cdot \boldsymbol{n} - \tau_D u_h^-)\, v^- \\ = \int_\Omega f\, v + \tau_D \int_{\Gamma_D} g\, v^- + \int_{\Gamma_N} h\, v^-.\end{aligned} \tag{4.2.12}$$

Additionally, define the following operators:

- Similar to the operator J_D above, let $J_N(h) \in V_h$ be such that

  $$\int_\Omega J_N(h)\, v = \int_{\Gamma_N} h\, v^-$$

  for all $v \in V_h$.

- Let $E_0, E_D : V_h \to V_h$ be the operators such that, for each $u \in V_h$,

  $$\int_\Omega E_0(u)\, v = \int_{\Gamma_0} [\![u]\!] \cdot [\![v]\!], \qquad \int_\Omega E_D(u)\, v = \int_{\Gamma_D} u^- v^-$$

  hold for every $v \in V_h$. These operators penalize jumps in the discrete solution on interior and Dirichlet boundary faces, respectively.

- Let $a_D(g) \in V_h$ be such that

  $$\int_\Omega a_D(g)\, v = \int_{\Gamma_D} g\, v^-$$

  for all $v \in V_h$.

Then, using the fact that $(\boldsymbol{q}_h, \nabla_h v) + (\boldsymbol{q}_h, Lv) = (\boldsymbol{q}_h, Gv)$, (4.2.12) is equivalent to

$$(\boldsymbol{q}_h, Gv) + \tau_0 (E_0 u_h, v) + \tau_D (E_D u_h, v) = (f + J_N(h) + \tau_D a_D(g), v), \tag{4.2.13}$$

or, putting aside penalty terms, $G^* \boldsymbol{q}_h = f + J_N(h)$, where $-G^*$ is the *discrete divergence operator*, the negative adjoint of the discrete gradient operator G; this is the LDG discretization of the statement that $-\nabla \cdot \boldsymbol{q} = f$, taking into account Neumann boundary data.

4.2.3.1 Primal formulation

To obtain the *primal formulation* of the LDG method, we combine (4.2.13) with (4.2.11) to eliminate $\boldsymbol{q}_h$ and arrive at an equation for u_h. The primal LDG formulation of (4.2.1) reads as follows: find $u_h \in V_h$ such that

$$a(u_h, v) = \ell(v) \tag{4.2.14}$$

for all $v \in V_h$, where the bilinear form $a(\cdot,\cdot)$ is given by

$$a(u_h, v) = (Gu_h, Gv) + \tau_0(E_0 u_h, v) + \tau_D(E_D u_h, v)$$

and the linear functional $\ell(\cdot)$ is given by

$$\ell(v) = (f, v) - (J_D(g), Gv) + (J_N(h), v) + \tau_D(a_D(g), v).$$

One may verify that the bilinear form $a(u, v)$ is symmetric. Discretization of the primal form (4.2.14) with respect to a particular basis of V_h yields a symmetric positive (semi)definite linear system of the form[20].

$$Au_h = \ell, \tag{4.2.15}$$

where A is the matrix form of the negative *discrete Laplacian operator*.

4.2.3.2 Flux formulation

An alternative, but equivalent, characterization of the LDG method is the so-called *flux formulation*, which does not eliminate the auxiliary variable $\boldsymbol{q}_h$ from the system (4.2.11) and (4.2.13) but instead retains it as a primary unknown. The flux formulation of (4.2.1) then reads as follows: find $(\boldsymbol{q}_h, u_h) \in V_h^d \times V_h$ such that

$$\begin{aligned} m(\boldsymbol{q}_h, \boldsymbol{\omega}) - \operatorname{grad}(u_h, \boldsymbol{\omega}) &= j(\boldsymbol{\omega}), \\ -\operatorname{div}(\boldsymbol{q}_h, v) + \tau(u_h, v) &= k(v), \end{aligned} \tag{4.2.16}$$

for all $(\boldsymbol{\omega}, v) \in V_h^d \times V_h$, where

$$\begin{aligned} m(\boldsymbol{q}, \boldsymbol{\omega}) &= (\boldsymbol{q}, \boldsymbol{\omega}), & \tau(u, v) &= \tau_0(E_0 u, v) + \tau_D(E_D u, v), \\ \operatorname{grad}(u, \boldsymbol{\omega}) &= (Gu, \boldsymbol{\omega}), & j(\boldsymbol{\omega}) &= (J_D(g), \boldsymbol{\omega}), \\ \operatorname{div}(\boldsymbol{q}, v) &= -(\boldsymbol{q}, Gv), & k(v) &= (f, v) + (J_N(h), v) + \tau_D(a_D(g), v). \end{aligned}$$

[20] Throughout this chapter we shall frequently use the same symbol to denote (i) elements of spaces such as V_h or operators acting on such elements, and (ii) vectors of coefficients in the chosen basis or matrices acting on such vectors. The distinction should be clear from context. Further comments are provided in subsection 4.2.3.3

Discretization of the flux form (4.2.16) with respect to a particular basis of $V_h^d \times V_h$ yields a symmetric positive (semi)definite linear system of the form

$$\begin{bmatrix} M & -MG \\ -MD & MT \end{bmatrix} \begin{bmatrix} \boldsymbol{q}_h \\ u_h \end{bmatrix} = \begin{bmatrix} j \\ k \end{bmatrix}. \tag{4.2.17}$$

Here, M is the block diagonal mass matrix for V_h or V_h^d (depending on context), G is the matrix form of the discrete gradient operator, $D = -M^{-1}G^\top M$ is the matrix form of the discrete divergence operator, and $T = \tau_0 E_0 + \tau_D E_D$ contains the discrete penalty terms. Since M^{-1} is also block diagonal, we can easily take the Schur complement of M in (4.2.17) to obtain a linear system for the unknown vector u_h,

$$Au_h = \ell, \tag{4.2.18}$$

where $A = M(-DG + T) = G^\top MG + MT$ and $\ell = k - G^\top j$.

The reduced linear system (4.2.18) is equivalent to the discrete primal formulation (4.2.15). However, as we will demonstrate next in section 4.3, the two formulations have different implications for multigrid methods. In particular, applying standard operator coarsening to the discrete primal formulation results in poor multigrid performance; coarsening the discrete flux formulation (4.2.17) in both $\boldsymbol{q}_h$ and u_h *before* taking the Schur complement (4.2.18) results in optimal multigrid performance, and is equivalent to pure geometric multigrid.

4.2.3.3 Remarks on the choice of basis

The analysis and discussion presented in this work is agnostic to the particular choice of basis for the piecewise polynomial space V_h. One may use a nodal basis, a modal basis, or some other choice, provided it is understood that every basis-dependent matrix (e.g., the mass matrix M) is defined consistently, relative to the chosen basis. In a numerical implementation, one should consider aspects of conditioning, accuracy, stability, sparsity, and computational complexity. For example, for low-to-moderate polynomial degree on rectangular elements, a nodal basis using Gauss–Lobatto nodes is a natural choice [91]; for very large p, a modal basis may have better conditioning or improved cost of mass matrix inversion, and thus may be more suitable. In our particular implementation, we have used a tensor-product Gauss–Lobatto nodal basis. We emphasize however that the presented multigrid methods and the essential conclusions drawn are not dependent on this choice.

4.3 Multigrid methods

We assume here that the reader has some familiarity with multigrid methods; see for example Briggs, Henson, and McCormick [38] for a review of their design and

operation. A geometric multigrid method consists of four main ingredients: a mesh hierarchy, an interpolation operator to transfer approximate solutions from a coarse mesh onto a fine mesh, a restriction operator to formulate a coarse mesh correction problem by restricting the residual from the fine mesh, and a smoother/relaxation method. We consider these ingredients separately first, and then combine them into a multigrid V-cycle.

Multigrid methods rely on the complementarity between relaxation and interpolation. In the geometric multigrid context, a relaxation method that is effective at damping high-frequency, oscillatory errors but slow to damp smooth, low-frequency ones benefits from the action of an interpolation operator that can accurately transfer low-frequency information. By solving a correction equation for the error on a coarser grid, fine-grid low-frequency errors become coarse-grid high-frequency errors for which coarse-grid relaxation is effective. An interpolation operator then transfers this low-frequency correction to the fine grid.

In the following sections we focus our description on h-multigrid methods, wherein the mesh is coarsened geometrically at each level. However, much of our analysis carries over to p-multigrid methods, which hold the mesh fixed and instead coarsen the polynomial space by reducing p at each level. We will try to point out the distinctions between the two methods when the analogues are not immediately obvious, though we will use the notation h in our description.

4.3.1 Mesh hierarchy

In this work, we employ quadtrees (in 2D) and octrees (in 3D) to define the finest mesh—whether it is uniform, adaptively refined, or used as the background grid for an implicitly defined mesh (see subsection 4.4.4). The tree structure then naturally defines a hierarchy of nested meshes for use in h-multigrid, which are spatially coarsened by a factor of two in each dimension on each level. For adaptively refined meshes where the cell size is not uniform, we coarsen each element as rapidly as the tree structure permits (see, e.g., Figure 4.6).

In the context of p-multigrid methods, a mesh hierarchy is defined by applying a specific p-coarsening strategy to the fine mesh. For example, one could coarsen p sequentially ($p \to p-1 \to p-2 \to \ldots \to 1$), by a factor of two ($p \to p/2 \to p/4 \to \cdots \to 1$), or by some user-defined sequence of p's. The first method is a common choice when low-order polynomials are used on the finest mesh, whereas the second method is better suited to high-order discretizations.

Mesh hierarchies can also be generated by combining coarsening strategies in both h and p. For example, a popular choice is to layer p-multigrid on top of h-multigrid, so that h-multigrid with a low-order polynomial degree is used as the bottom solver in the p-multigrid hierarchy.

4.3.2 Interpolation

The interpolation operator I_c^f transfers a piecewise polynomial function $u_c \in V_{2h}(\mathcal{E}_c)$ defined on a coarse mesh $\mathcal{E}_c$ to a piecewise polynomial function $u_f \in V_h(\mathcal{E}_f)$ on a fine mesh $\mathcal{E}_f$. (Throughout this work, subscripts or superscripts f, h and c, $2h$ shall denote objects corresponding to the fine mesh and coarse mesh, respectively.) We define the interpolation operator so that it injects the piecewise polynomial function on the coarse mesh into the fine mesh, unmodified. From the h-multigrid perspective, $u_f|_{E_f}$ on the fine element E_f is simply the polynomial $u_c|_{E_c}$ restricted to E_f, where $E_c \supset E_f$ is the corresponding coarse element in the mesh hierarchy. From the p-multigrid perspective, the lower-degree polynomial u_c can be exactly represented as a higher-degree polynomial by taking the higher-order coefficients of u_f to be zero. In either case, when regarded as an operator from $L^2(\Omega) \to L^2(\Omega)$, I_c^f is the identity operator. The operator is linear and has the property that it preserves constant functions, i.e., $u_c \equiv 1$ is mapped to $u_f \equiv 1$. This property ensures that, throughout a V-cycle, the coarse mesh discrete problems preserve the compatibility condition required in semidefinite problems having solely Neumann boundary conditions.

4.3.3 Restriction

We define the restriction operator $R_f^c : V_h(\mathcal{E}_f) \to V_{2h}(\mathcal{E}_c)$ as the adjoint of the interpolation operator, i.e., such that

$$(R_f^c u_f, u_c)_{\mathcal{E}_c} = (u_f, I_c^f u_c)_{\mathcal{E}_f} \tag{4.3.1}$$

holds for every $u_f \in V_h(\mathcal{E}_f)$ and every $u_c \in V_{2h}(\mathcal{E}_c)$. Equivalently, letting I_c^f, R_f^c, u_c, and u_f also denote matrices and vectors relative to the user-defined bases of $V_h(\mathcal{E}_f)$ and $V_{2h}(\mathcal{E}_c)$, (4.3.1) can be restated as

$$(R_f^c u_f)^\top M_c u_c = u_f^\top M_f I_c^f u_c$$

where M_c and M_f are the block-diagonal mass matrices of the coarse and fine meshes, respectively. Therefore,

$$R_f^c = M_c^{-1} (I_c^f)^\top M_f. \tag{4.3.2}$$

Defining the restriction operator in this manner—sometimes referred to as Galerkin projection—results in several notable properties:

- One may interpret $R_f^c u_f$ as "averaging" elemental polynomials of $u_f \in V_h(\mathcal{E}_f)$ on the fine mesh to determine a coarsened piecewise-polynomial representation on the coarse mesh. The averaging is performed in a way that locally preserves

the mass of u_f: indeed, since I_c^f preserves constant functions, we have that $(R_f^c u_f, 1)_{\mathcal{E}_c} = (u_f, 1)_{\mathcal{E}_f}$ for all $u_f \in V_h(\mathcal{E}_f)$.

- The adjoint method can also be viewed as an L^2 projection of $u_f \in V_h(\mathcal{E}_f)$ onto $V_{2h}(\mathcal{E}_c)$. The variational problem $\arg\min_{u_c \in V_{2h}(\mathcal{E}_c)} \|u_c - u_f\|_\Omega^2$ optimizes the functional

$$V_{2h}(\mathcal{E}_c) \ni u_c \mapsto (u_c, u_c)_{\mathcal{E}_c} - 2(u_f, I_c^f u_c)_{\mathcal{E}_f} = u_c^\top M_c u_c - 2u_f^\top M_f I_c^f u_c$$

  whose unique minimum is given by $u_c = M_c^{-1}(I_c^f)^\top M_f u_f$.

- From the preceding property, one can immediately infer that

$$R_f^c I_c^f = \mathbb{I}, \tag{4.3.3}$$

  where $\mathbb{I}$ is the identity operator; i.e., interpolating a piecewise polynomial function from a coarse mesh onto a fine mesh and immediately restricting the result shall return the original function. The relation in (4.3.3) together with (4.3.2) also provides a method to compute the coarse-mesh mass matrix from the fine-mesh mass matrix:

$$M_c = (I_c^f)^\top M_f I_c^f. \tag{4.3.4}$$

Given a linear operator $A : V_h(\mathcal{E}_f) \to V_h(\mathcal{E}_f)$, one can define a coarsened operator $\mathcal{C}(A) : V_{2h}(\mathcal{E}_c) \to V_{2h}(\mathcal{E}_c)$ in a similar way, by proceeding variationally: we define $\mathcal{C}(A)$ such that

$$(\mathcal{C}(A)u_c, v_c)_{\mathcal{E}_c} = (AI_c^f u_c, I_c^f v_c)_{\mathcal{E}_f}$$

holds for all $u_c, v_c \in V_{2h}(\mathcal{E}_c)$. Viewing A and $\mathcal{C}(A)$ as matrix operators, mapping vectors in the user-defined bases of $V_h(\mathcal{E}_f)$ and $V_{2h}(\mathcal{E}_c)$,

$$\mathcal{C}(A) = M_c^{-1}(I_c^f)^\top M_f A I_c^f = R_f^c A I_c^f.$$

The last form is perhaps more commonly seen or referred to as *"RAT"* in the multigrid literature [178], where R is restriction, A is the fine-mesh operator, and T (or P) is the interpolation (or prolongation) operator; the essence of the present work is to show that directly applying *RAT* to the negative discrete Laplacian resulting from the primal formulation of an LDG method results in an inefficient multigrid algorithm and that, instead, applying *RAT* to the flux formulation, $\boldsymbol{q} = \nabla u$, $-\nabla \cdot \boldsymbol{q} = f$, leads to more efficient multigrid solvers.

4.3.4 Operator coarsening and pure geometric multigrid

In this section we compare a standard, purely geometric multigrid method to two multigrid schemes based on operator coarsening: (i) applying *RAT* to the negative discrete Laplacian matrix of the primal formulation ("primal coarsening") and (ii) applying *RAT* to the 2×2 block matrix of the flux formulation ("flux coarsening"). By a pure geometric method, we mean one in which each level of the hierarchy is explicitly meshed and the LDG formulation is canonically applied to each level, with the above restriction and interpolation operators transferring residual and correction vectors (in a V-cycle) between levels. Our motivation here concerns an h-multigrid method; however, much of the following discussion has direct analogy with p-multigrid methods. In addition, in the context of DG methods requiring penalty parameters, a design choice can be made as to how the value of the penalty parameter is chosen at each level of the hierarchy. In this work we consider the natural choice in which every level of the hierarchy inherits the same value as the finest mesh. With this in mind, we discuss interaction between a pair of levels: suppose $\mathcal{E}_f$ is the mesh of a fine level and $\mathcal{E}_c$ is the mesh of the next-coarsest level.

4.3.4.1 Primal coarsening

Recall the primal form of the negative discrete Laplacian operator of an LDG method: as a matrix mapping the coefficient vectors in the basis of $V_h(\mathcal{E}_f)$ into the basis of $V_h(\mathcal{E}_f)$, i.e., premultiplying (4.2.18) by M^{-1},

$$A = -DG + \tau_0 E_0 + \tau_D E_D,$$

where $G = \nabla_h + L$ is the discrete gradient operator and $D = -M^{-1}G^\top M$ is the discrete divergence operator. To discuss the application of *RAT* to A and how it relates to a geometric multigrid implementation, we consider the individual terms making up A.

- First, we note that the broken gradient operator satisfies the *RAT* property, i.e., $\mathcal{C}(\nabla_h) = \nabla_{2h}$. Computing the piecewise gradient on a coarse mesh and interpolating the result to the fine mesh is the same as computing the piecewise gradient of the interpolant, i.e., $I_c^f \nabla_{2h} u_c = \nabla_h I_c^f u_c$ for all $u_c \in V_{2h}(\mathcal{E}_c)$; consequently, $\mathcal{C}(\nabla_h) = R_f^c \nabla_h I_c^f = R_f^c I_c^f \nabla_{2h} = \nabla_{2h}$ by (4.3.3).

- The lifting operator also satisfies the *RAT* property, i.e., $\mathcal{C}(L_f) = L_c$. This is perhaps not immediately obvious, since source terms on a coarse mesh face will lift into the corresponding large coarse element, whereas the corresponding source terms on the fine mesh faces lift only into the smaller elements touching that face; however, the restriction of the result on the set of smaller elements

agrees with the result of L_c. To see this, apply the variational formulation of $\mathcal{C}(\cdot)$ to observe that

$$\begin{aligned}
&(\mathcal{C}(L_f)u_c, \boldsymbol{v}_c)_{\mathcal{E}_c} \\
&\quad = (L_f I_c^f u_c, I_c^f \boldsymbol{v}_c)_{\mathcal{E}_f} \\
&\quad = -\int_{\Gamma_{0,f}} [\![I_c^f u_c]\!] \cdot \left(\{\!\{ I_c^f \boldsymbol{v}_c \}\!\} + \boldsymbol{\beta} [\![I_c^f \boldsymbol{v}_c]\!] \right) - \int_{\Gamma_{D,f}} (I_c^f u_c)^- (I_c^f \boldsymbol{v}_c)^- \cdot \boldsymbol{n} \\
&\quad = -\int_{\Gamma_{0,c}} [\![u_c]\!] \cdot (\{\!\{ \boldsymbol{v}_c \}\!\} + \boldsymbol{\beta} [\![\boldsymbol{v}_c]\!]) - \int_{\Gamma_{D,c}} u_c^- \boldsymbol{v}_c^- \cdot \boldsymbol{n} \\
&\quad = (L_c u_c, \boldsymbol{v}_c)_{\mathcal{E}_c}
\end{aligned}$$

holds for all $u_c \in V_{2h}(\mathcal{E}_c)$ and $\boldsymbol{v}_c \in V_{2h}^d(\mathcal{E}_c)$. Here, $\Gamma_{0,f}$ and $\Gamma_{0,c}$ denote the union of interior faces of the fine and coarse meshes, respectively, and similarly for $\Gamma_{D,f}$ and $\Gamma_{D,c}$. The third equality holds because the interpolation operator introduces no nonzero jumps on the set of new fine mesh faces, i.e., on $\Gamma_{0,f} \setminus \Gamma_{0,c}$ and $\Gamma_{D,f} \setminus \Gamma_{D,c}$. (The preceding assumes that fine mesh faces inherit the same $\boldsymbol{\beta}$ value as coarse mesh faces; in particular, this is true for the one-sided LDG scheme in which $\boldsymbol{\beta} = \pm\frac{1}{2}\boldsymbol{n}$.)

- It immediately follows from the preceding two properties that $\mathcal{C}(G_f) = G_c$. Moreover,

$$\begin{aligned}
\mathcal{C}(D_f) = R_f^c D_f I_c^f &= \left(M_c^{-1}(I_c^f)^\top M_f\right)\left(-M_f^{-1} G_f^\top M_f\right) I_c^f \\
&= -M_c^{-1}\left((I_c^f)^\top G_f^\top M_f I_c^f M_c^{-1}\right) M_c = -M_c^{-1}\left(\mathcal{C}(G_f)\right)^\top M_c \\
&= -M_c^{-1} G_c^\top M_c = D_c.
\end{aligned}$$

- It is straightforward to show that the penalty operators also satisfy the *RAT* property, i.e., $\mathcal{C}(E_{0,f}) = E_{0,c}$ and $\mathcal{C}(E_{D,f}) = E_{D,c}$. As in the case of the lifting operator, this property derives from the fact the interpolation operator does not introduce jumps on fine mesh faces that do not overlap with coarse mesh faces.

Despite these consistencies, the negative discrete Laplacian does not satisfy the *RAT* property—the application of *RAT* to the fine-mesh negative discrete Laplacian A_f does not yield the coarse-mesh operator A_c obtained from pure geometric multigrid. Using the properties derived above,

$$\begin{aligned}
A_c &= -D_c G_c + \tau_0 E_{0,c} + \tau_D E_{D,c} \\
&= -\mathcal{C}(D_f)\mathcal{C}(G_f) + \tau_0 \mathcal{C}(E_{0,f}) + \tau_D \mathcal{C}(E_{D,f})
\end{aligned}$$

which differs from the direct coarsening of A_f,

$$\mathcal{C}(A_f) = -\mathcal{C}(D_f G_f) + \tau_0 \mathcal{C}(E_{0,f}) + \tau_D \mathcal{C}(E_{D,f}) \neq A_c,$$

since in general $\mathcal{C}(D_f G_f) \neq \mathcal{C}(D_f)\mathcal{C}(G_f)$. Informally, $\mathcal{C}(D_f G_f)u_c$ interpolates a function $u_c \in V_{2h}(\mathcal{E}_c)$ onto the fine mesh $\mathcal{E}_f$, computes the gradient as a function in $V_h^d(\mathcal{E}_f)$, computes the divergence as a function in $V_h(\mathcal{E}_f)$, and projects the result back to the coarse mesh $\mathcal{E}_c$. On the other hand, $\mathcal{C}(D_f)\mathcal{C}(G_f)u_c$ projects the computed fine-mesh gradient onto the coarse mesh and then immediately interpolates the result in order to compute the discrete divergence on the fine mesh, before projecting the final result back to the coarse mesh. That is,

$$\mathcal{C}(D_f)\mathcal{C}(G_f) = \mathcal{C}(D_f I_c^f R_f^c G_f) \neq \mathcal{C}(D_f G_f),$$

since $I_c^f R_f^c \neq \mathbb{I}$.

4.3.4.2 Flux coarsening

The coarse operator A_c obtained from pure geometric multigrid may be viewed as applying *RAT* to the equations $\boldsymbol{q} = \nabla u$ and $-\nabla \cdot \boldsymbol{q} = f$ separately. The flux formulation of LDG, (4.2.17), naturally displays this coarsening strategy. To show this, note that we can write the flux formulation with input and output in the user-defined basis by premultiplying (4.2.17) by the inverse mass matrix to obtain

$$\begin{bmatrix} I & -G \\ -D & T \end{bmatrix} \begin{bmatrix} \boldsymbol{q}_h \\ u_h \end{bmatrix} = \begin{bmatrix} M^{-1} j \\ M^{-1} k \end{bmatrix}. \tag{4.3.5}$$

Applying *RAT* in a block fashion to the flux formulation (4.3.5) then yields the discrete operator

$$\begin{bmatrix} R_f^c & 0 \\ 0 & R_f^c \end{bmatrix} \begin{bmatrix} I & -G_f \\ -D_f & T_f \end{bmatrix} \begin{bmatrix} I_c^f & 0 \\ 0 & I_c^f \end{bmatrix} = \begin{bmatrix} I & -G_c \\ -D_c & T_c \end{bmatrix}. \tag{4.3.6}$$

Taking the Schur complement of the right-hand side of (4.3.6), we obtain

$$\begin{aligned} A_c &= -D_c G_c + T_c \\ &= -\mathcal{C}(D_f)\mathcal{C}(G_f) + \tau_0 \mathcal{C}(E_{0,f}) + \tau_D \mathcal{C}(E_{D,f}), \end{aligned} \tag{4.3.7}$$

which is exactly the coarse operator from pure geometric multigrid. Thus, applying operator coarsening to the flux formulation of LDG, which is equivalent to separately coarsening the equations $\boldsymbol{q} = \nabla u$ and $-\nabla \cdot \boldsymbol{q} = f$, is the same as pure geometric multigrid.

Figure 4.1 depicts the three types of coarsening that can be performed, given a hierarchy of meshes. In the left column, pure geometric multigrid defines the coarse operators directly from the coarse meshes; in the center column, primal coarsening applies *RAT* to the discrete Laplacian operator; and in the right column, flux coarsening applies *RAT* separately to the discrete divergence and gradient operators. In the above, we have shown the equivalence of the left and right columns.

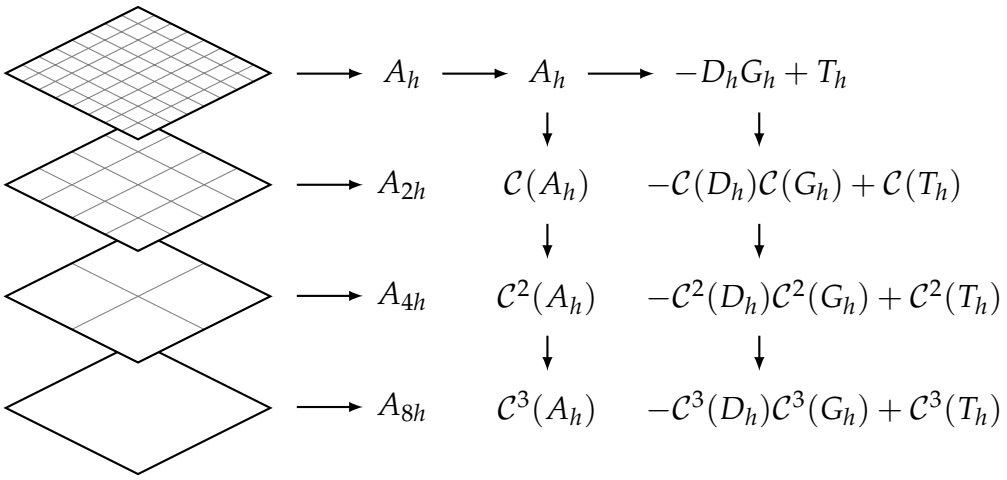


Figure 4.1: *Three coarsening methods can be used to generate a hierarchy of operators for multigrid: (left column) pure geometric multigrid, where the coarse operators are defined directly from the corresponding coarse meshes, (center column) primal coarsening, where the coarse operators are defined by applying RAT to the fine-mesh discrete Laplacian, and (right column) flux coarsening, where the coarse operators are defined by applying RAT to the fine-mesh discrete divergence, discrete gradient, and discrete penalty operators, and recombining the results.*

An implementation of constructing the operator hierarchy using flux coarsening is outlined in Algorithm 4.3.1.

4.3.4.3 Benefits of operator coarsening

It can be useful to define coarse operators directly from fine operators (e.g., by using *RAT*) rather than via discretizations computed directly from coarse meshes. Since coarse mass matrices can be computed automatically according to (4.3.4), quadrature schemes do not need to be computed for coarse elements—instead, fine-mesh quadrature rules are coarsened automatically via (4.3.4). Similarly, coarse lifting matrices are not explicitly needed since their contribution to the discrete gradient is automatically computed via $G_c = \mathcal{C}(G_f)$, and so quadrature rules for coarse faces also do not need to be defined. For implicitly defined meshes, such as the ones shown in subsection 4.4.4, computing coarse quadrature rules can be computationally intricate or taxing; the fact that operator coarsening obviates the need for this is a substantial benefit. Additionally, operator coarsening can be efficiently implemented using basic linear algebra operations, e.g., block-sparse matrix multiplication, for which highly optimized and parallelized libraries exist; in contrast, computing discretizations directly from coarse meshes relies heavily on the efficiency of one's own code. It is worth noting that the complexity of constructing the operator hierarchy in Algorithm 4.3.1 is the same as the complexity of the multigrid V-cycle in Algorithm 4.3.2 (i.e., $\mathcal{O}(N)$ for N elements); as an approximate indication, in practice the former takes the same computing time as about three to four applications of a V-cycle.

4.3.4.4 Relation to other DG methods

Although we have focused on the LDG method in our discussion, we expect that other DG methods may require similar care in coarsening fine-grid operators such that they are consistent with a pure geometric multigrid method. For instance, methods for which the numerical flux $\widehat{\boldsymbol{q}}_h$ depends on the discrete gradient of u_h—such as the BR1 [25] or Brezzi [37] methods—may need similar treatment, as the contribution from the lifting operator L must be coarsened separately.

Other methods, such as the symmetric interior penalty (SIP) method [10, 57], do not require the discrete divergence and discrete gradient operators to be coarsened separately. To demonstrate this for SIP, start with its corresponding bilinear form for a pure Neumann problem: find $u \in V_h$ such that $a(u,v) = l(v)$ for all $v \in V_h$, where

$$a(u,v) = (\nabla_h u, \nabla_h v) - \int_{\Gamma_0} (\{\!\{\nabla_h u\}\!\} \cdot [\![v]\!] + [\![u]\!] \cdot \{\!\{\nabla_h v\}\!\}) + \tau \int_{\Gamma_0} [\![u]\!] \cdot [\![v]\!]$$

and

$$l(v) = (f,v) + \int_{\Gamma_N} h\, v^-,$$

with τ scaling inversely to the element size. Now consider a pure geometric multigrid method. Let $u_c, v_c \in V_{2h}(\mathcal{E}_c)$. Then

$$\begin{aligned}
a_c(u_c,v_c) &= (\nabla_{2h} u_c, \nabla_{2h} v_c) - \int_{\Gamma_{0,c}} (\{\!\{\nabla_{2h} u_c\}\!\} \cdot [\![v_c]\!] + [\![u_c]\!] \cdot \{\!\{\nabla_{2h} v_c\}\!\}) \\
&\quad + \tau \int_{\Gamma_{0,c}} [\![u_c]\!] \cdot [\![v_c]\!] \\
&= (\nabla_h I_c^f u_c, \nabla_h I_c^f v_c) - \int_{\Gamma_{0,f}} (\{\!\{\nabla_h I_c^f u_c\}\!\} \cdot [\![I_c^f v_c]\!] + [\![I_c^f u_c]\!] \cdot \{\!\{\nabla_h I_c^f v_c\}\!\}) \\
&\quad + \tau \int_{\Gamma_{0,f}} [\![I_c^f u_c]\!] \cdot [\![I_c^f v_c]\!] \\
&= a_f(I_c^f u_c, I_c^f v_c),
\end{aligned}$$

where equality holds between the first and second lines because I_c^f does not introduce nonzero jumps on new mesh faces. Therefore, as matrices (mapping vectors in the user-defined basis to vectors in the same basis),

$$u_c^\top M_c A_c v_c = (I_c^f u_c)^\top M_f A_f (I_c^f v_c).$$

Assuming the quadratic form is nondegenerate (which is true since A_c and A_f are symmetric positive definite, ignoring the trivial kernel), this implies

$$A_c = M_c^{-1} (I_c^f)^\top M_f A_f I_c^f = R_f^c A_f I_c^f.$$

This is *RAT* applied to A_f, and so applying pure geometric multigrid to SIP is the same as recursively applying standard (primal) operator coarsening to A_f.

4.3.5 Multigrid preconditioned conjugate gradient

Recall that a geometric multigrid method utilizes a combination of relaxation/smoothing together with interpolated approximate solutions of coarsened problems. In the present case, the coarsened problem solves for the correction in a residual equation for the same elliptic problem except on a coarser mesh. In particular, it is important to note that, instead of solving $-\Delta_h u = f$, where Δ_h is the discrete Laplacian in the chosen basis, one instead solves $-M\Delta_h u = Mf$, as the latter system is symmetric positive (semi)definite. Thus, to appropriately define the coarse mesh problem, one may: (i) calculate the residual of the fine mesh linear system $A_f x_f = b_f$, (ii) multiply the residual by M_f^{-1} to correctly determine the residual as a piecewise polynomial function, (iii) restrict the residual to the coarse mesh, and then (iv) multiply this residual by M_c of the coarse mesh. Thus, the coarse mesh problem consists of (approximately) solving for x_c such that

$$A_c x_c = M_c(R_f^c[M_f^{-1}(b_f - A_f x_f)]),$$

which, according to the derived restriction operator (4.3.2), conveniently simplifies to

$$A_c x_c = (I_c^f)^\top (b_f - A_f x_f),$$

and so it is unnecessary to multiply by mass matrices in the implementation of the multigrid method; instead, one can simply apply the transpose of the interpolation matrix. With this consideration in mind, the design of a multigrid V-cycle is relatively straightforward and is outlined in Algorithm 4.3.2.

The V-cycle is designed to preserve the symmetric positive (semi)definite property of the discrete problem, making it suitable for preconditioning the conjugate gradient method. To that end, the relaxation sweeps are performed in a symmetric fashion; for order-dependent relaxation methods such as Gauss–Seidel, the first set of relaxation sweeps uses a given ordering of the unknowns and the second set uses the reverse of that ordering.[21] A multigrid preconditioned conjugate gradient method [160] (MGPCG) combines the advantages of both solvers: the multigrid preconditioner is effective in the interior of the domain where the elliptic behavior of the matrix dominates, while the conjugate gradient method effectively treats the remaining eigenmodes, which in turn are largely associated with the (weak) imposition of the boundary conditions (and in the case of multi-phase elliptic interface problems, jump

[21] If a relaxation scheme is used that is itself symmetric then there is no need to reverse the ordering of unknowns between pre- and post-smoothing steps, as the V-cycle will automatically preserve symmetry.

Algorithm 4.3.1 Construction of coarse operators, Build$(\mathcal{E}_f, M_f, G_f, T_f)$

Input: Fine-mesh operators M_f, G_f, T_f
Output: List of coarse operators

$\boldsymbol{A} := \{\}$
$A_f := G_f^\top M_f G_f + T_f$
if $\mathcal{E}_f$ is not the coarsest mesh **then**
    $M_c := (I_c^f)^\top M_f I_c^f$
    $G_c := M_c^{-1}(I_c^f)^\top M_f G_f I_c^f$
    $T_c := (I_c^f)^\top T_f I_c^f$
    $\boldsymbol{A} :=$ Build$(\mathcal{E}_c, M_c, G_c, T_c)$
return $\{A_f, \boldsymbol{A}\}$

Algorithm 4.3.2 Multigrid V-cycle $V(\mathcal{E}_f, x_f, b_f)$ on mesh $\mathcal{E}_f$ with ν pre- and post-smoothing steps

if $\mathcal{E}_f$ is the bottom level **then**
    Solve $A_f x_f = b_f$ directly
else
    Relax ν times
    $r_c := (I_c^f)^\top (b_f - A_f x_f)$
    $x_c := V(\mathcal{E}_c, 0, r_c)$
    $x_f := x_f + I_c^f x_c$
    Relax ν times
return x_f

conditions on internal interfaces) [151, 159]. We use a single multigrid V-cycle as a preconditioner in the conjugate gradient method.

4.4 Numerical results

In this section, we present numerical experiments to assess the efficacy of flux coarsening for LDG discretizations of elliptic PDEs. As the smoother/relaxation method, we use a block Gauss–Seidel smoother with $\nu = 3$ pre- and post-smoothing steps[22] in the V-cycle. We initially set the interior penalty parameter to $\tau_0 = 0.01/h$; additional analysis of the influence of penalty parameters is given in subsection 4.4.2. We measure multigrid performance via the average convergence factor

$$\rho = \exp\left(\frac{1}{N}\log\frac{\|e_N\|_2}{\|e_0\|_2}\right), \tag{4.4.1}$$

where N is the number of iterations required to reduce the relative error by a factor of 10^{-10} and e_i is the error at iteration i of either standalone multigrid (i.e., i many V-cycles) or MGPCG (i.e., the i^{th} iteration of CG preconditioned by a single V-cycle). In effect, ρ measures the average slope of e_i on a log-linear graph. Convergence is measured using a right-hand side of $f = 0$ with a random nonzero initial guess for u. Convergence results are presented in the following graphs with ρ as a function of element size h, polynomial degree p, etc. The same data is presented in tabular form in Appendix D.

[22] As is typical in multigrid methods, increasing the number of pre- and post-smoothing steps can increase the speed of convergence, i.e., decrease ρ as measured by (4.4.1); however, doing so comes at the cost of a more expensive V-cycle and therefore may be less efficient. We observed that $\nu = 3$ gave the best computational efficiency in our numerical experiments in terms of reducing the error by a fixed factor.

4.4.1 Uniform Cartesian grids

We start by solving (4.2.1) with homogeneous Neumann boundary conditions on the domain $\Omega = [0,1]^d$ using a uniform Cartesian grid of size $n \times n$ (for $d = 2$) or $n \times n \times n$ (for $d = 3$) with cell size $h = 1/n$. We build an h-multigrid hierarchy based on uniform grid refinement by applying both primal and flux coarsening to the discretized LDG system, and solve using both standalone V-cycles and MGPCG.

Figures 4.2 and 4.3 show the average convergence factor versus n for polynomial orders $1 \leq p \leq 5$ in 2D and 3D, respectively. In both cases, the multigrid scheme built using flux coarsening exhibits nearly h-independent convergence factors of $\rho \approx 0.1$, whereas the scheme based on primal coarsening exhibits poor performance that degrades as $h \to 0$.

Similar results hold for p-multigrid on uniform Cartesian grids. We generate a p-multigrid hierarchy by successively halving the polynomial order (i.e., $p \to p/2 \to p/4 \to \cdots \to 1$) and applying both primal and flux coarsening to the discretized LDG system. Figure 4.4 shows convergence factor versus p for grid sizes $4 \leq n \leq 512$ in 2D and 3D. Again, convergence appears to be independent of p (at least up to $p = 8$) when flux coarsening is used with p-multigrid, whereas performance degrades with increasing p for primal coarsening.

4.4.2 On the effect of penalty parameters on multigrid performance

Figure 4.5 (left) shows a study of the impact the interior penalty parameter $\tau_0 = \widetilde{\tau}_0/h$ has on multigrid convergence for a Poisson problem on a uniform $n \times n$ mesh with periodic boundary conditions and $p = 2$. Smaller values of $\widetilde{\tau}_0$ yield better convergence factors; for $\widetilde{\tau}_0 > 10$, multigrid performance begins to degrade as the mesh is refined. For the remainder of our tests, we set $\widetilde{\tau}_0 = 0.01$ so that $\tau_0 = 0.01/h$.

In our tests, the imposition or combination of Dirichlet, Neumann, or periodic boundary conditions does not affect the conclusions made in this work. However, for problems with Dirichlet boundary conditions the choice of Dirichlet penalty parameter τ_D can impact multigrid efficiency. Informally, Dirichlet boundary conditions are enforced in a DG method only weakly and the smoothing/relaxation method of a V-cycle can only effectively enforce the boundary condition if the associated penalty parameter is sufficiently strong. Figure 4.5 (right) shows a study of the impact that $\tau_D = \widetilde{\tau}_D/h$ has on multigrid performance for a homogeneous Dirichlet problem on a uniform $n \times n$ mesh. For $\widetilde{\tau}_D < 1$, the average convergence factor ρ degrades as n increases; a good choice in this case appears to be $10 < \widetilde{\tau}_D < 100$.

Ultimately, the proper choice of both τ_0 and τ_D is application-dependent and concerns not only multigrid performance but also discretization accuracy and effects of penalty stabilization on the conditioning of the linear systems.

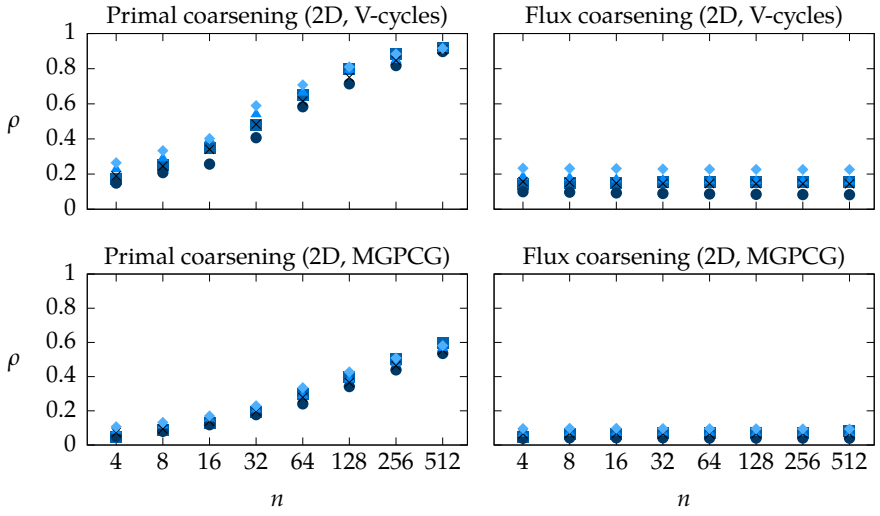


Figure 4.2: *h-multigrid convergence factors for primal coarsening ($A_c = \mathcal{C}(A_f)$) and flux coarsening ($A_c = -\mathcal{C}(D_f)\mathcal{C}(G_f) + \mathcal{C}(T_f)$) applied to the LDG discretization of Poisson's equation on a uniform $n \times n$ Cartesian grid as $h \to 0$. The top row results are computed using V-cycles whereas the bottom row results are computed using MGPCG. The plot markers indicate different polynomial orders:* ■ , ● , ▲ , × , *and* ◆ *denote $p = 1$, 2, 3, 4, and 5, respectively.*

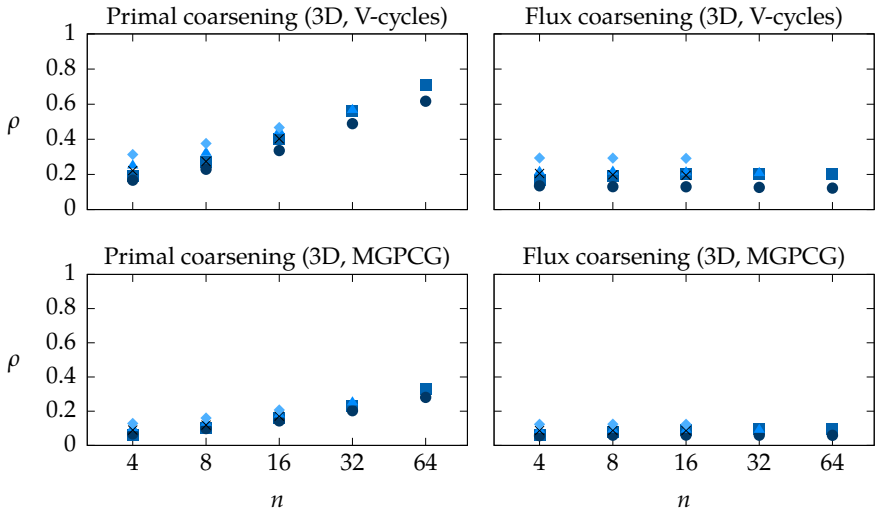


Figure 4.3: *h-multigrid convergence factors for primal coarsening ($A_c = \mathcal{C}(A_f)$) and flux coarsening ($A_c = -\mathcal{C}(D_f)\mathcal{C}(G_f) + \mathcal{C}(T_f)$) applied to the LDG discretization of Poisson's equation on a uniform $n \times n \times n$ Cartesian grid as $h \to 0$. The top row results are computed using V-cycles whereas bottom row results are computed using MGPCG. The plot markers indicate different polynomial orders:* ■ , ● , ▲ , × , *and* ◆ *denote $p = 1$, 2, 3, 4, and 5, respectively. (Omitted data points correspond to simulations whose memory requirements approximately exceed* 120 GB.*)*

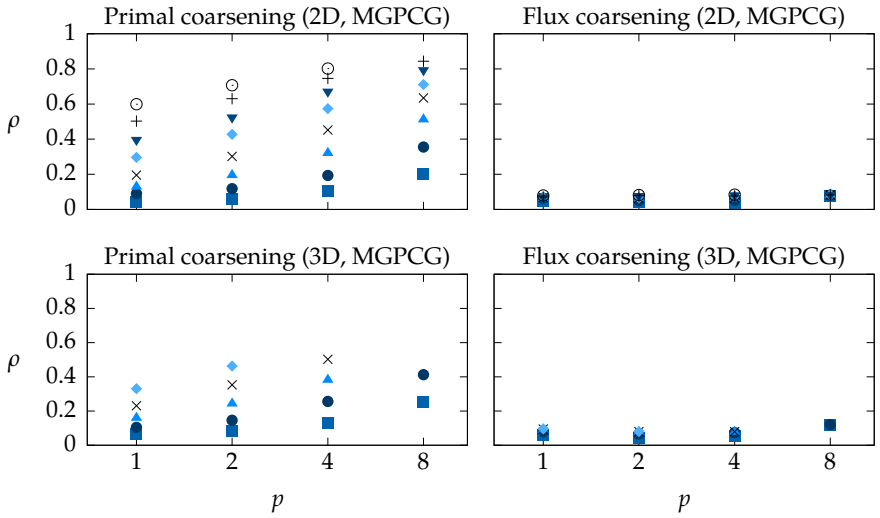


Figure 4.4: *p-multigrid convergence factors for primal coarsening ($A_c = \mathcal{C}(A_f)$) and flux coarsening ($A_c = -\mathcal{C}(D_f)\mathcal{C}(G_f) + \mathcal{C}(T_f)$) applied to the LDG discretization of Poisson's equation on uniform grids. The p-multigrid hierarchy is generated by successively halving the polynomial order (i.e., $p \to p/2 \to p/4 \to \cdots \to 1$). The plot markers indicate different grid sizes:* ■ , ● , ▲ , × , ◆ , ▼ , + , *and* ○ *denote* $n = 4$, 8, 16, 32, 64, 128, 256, *and* 512, *respectively. (Omitted data points correspond to simulations whose memory requirements approximately exceed* 120 GB.*)*

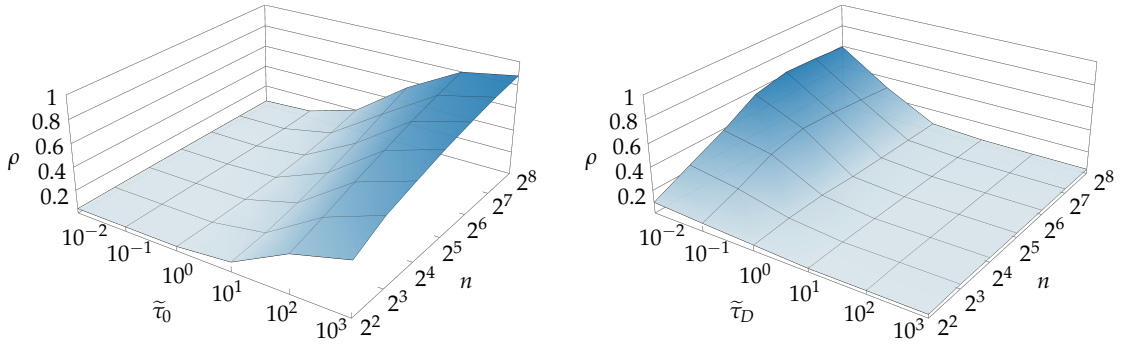


Figure 4.5: *Effects of interior penalty parameter $\tau_0 = \widetilde{\tau}_0/h$ (left) and Dirichlet penalty parameter $\tau_D = \widetilde{\tau}_D/h$ (right) on multigrid convergence factor ρ for an $n \times n$ Cartesian mesh with $p = 2$, where $h = 1/n$; see subsection* 4.4.2.

4.4.3 Adaptive mesh refinement

Next, we solve the Neumann problem (4.2.1) on an adaptively refined Cartesian mesh, where refinement is performed according to some prescribed spatially-varying function. We implement adaptivity using a quadtree (in 2D) or octree (in 3D), which naturally defines a geometric hierarchy of meshes. An example hierarchy is shown in Figure 4.6. Note that some elements at various levels of the hierarchy have four neighbors on a single side, so that large elements and small elements may share part of a face.

Figure 4.7 shows the average convergence factor versus $1/h_{\min}$ for polynomial orders $1 \leq p \leq 5$, where $h_{\min}$ is the size of the smallest element in the mesh. In both

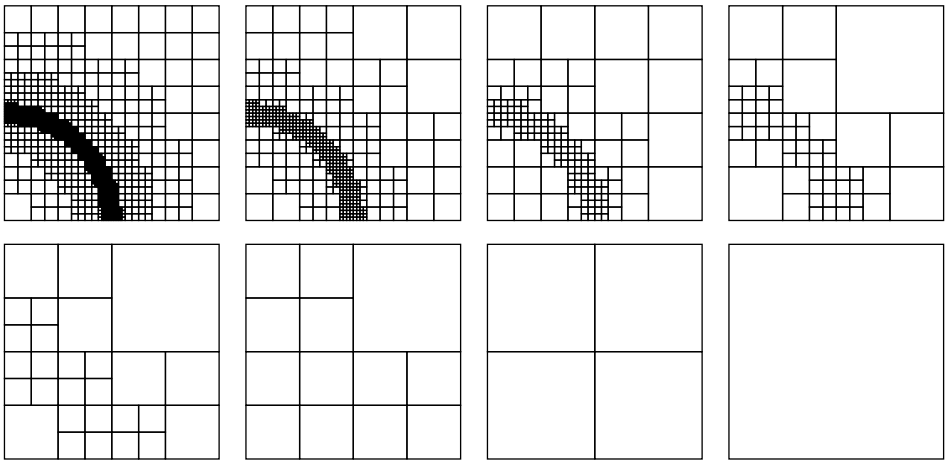


Figure 4.6: *An example of a geometric multigrid hierarchy inherited from an adaptively refined quadtree with rapid coarsening. The finest level depicted has smallest cell size equal to* $h_{min} = 1/128$.

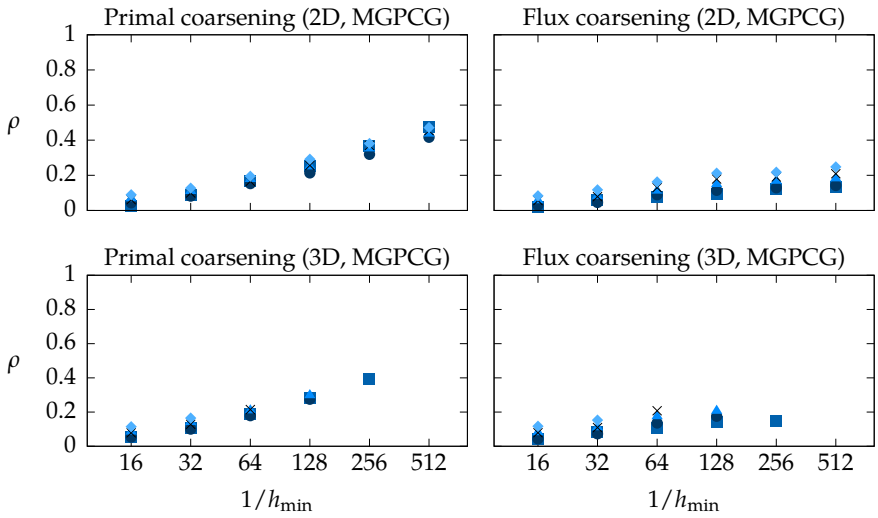


Figure 4.7: *h-multigrid convergence factors for primal* ($A_c = \mathcal{C}(A_f)$) *and flux* ($A_c = -\mathcal{C}(D_f)\mathcal{C}(G_f) + \mathcal{C}(T_f)$) *coarsening applied to the LDG discretization of Poisson's equation on an adaptively refined grid in 2D and 3D. The plot markers indicate different polynomial orders:* ■, ●, ▲, ×, *and* ◆ *denote* $p = 1, 2, 3, 4,$ *and* 5, *respectively. (Omitted data points correspond to simulations whose memory requirements approximately exceed* 120 GB.*)*

2D and 3D, the multigrid method based on primal coarsening exhibits performance that degrades as $h_{\min} \to 0$, whereas the method based on flux coarsening yields good performance that is nearly independent of $h_{\min}$ for all p considered.

4.4.4 Implicitly defined meshes and elliptic interface problems

Our last two examples are designed to exemplify the benefits of operator coarsening by considering cases in which a pure geometric multigrid method would be intricate or difficult to implement, such as on nontrivial domains with complex geometry or for elliptic interface problems in which the interface has extreme geometry. The

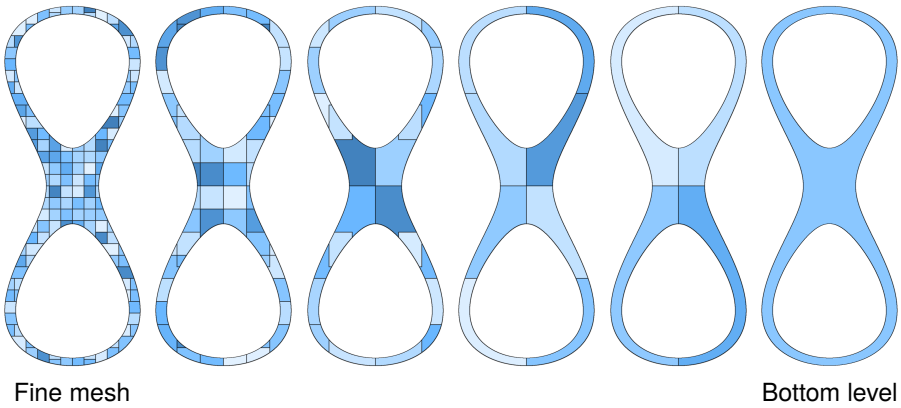


Figure 4.8: *A single-phase test case applied to a curved domain using implicitly defined meshes. Depicted is the implied h-multigrid hierarchy (elements are randomly colored). The coarse meshes are not explicitly constructed in our operator coarsening approach; instead, the discrete gradient and penalty operators and mass matrices are constructed top-down at each level (see Algorithm* 4.3.1*). In particular, note that the bottom level of the hierarchy consists of a mesh with a single element containing two holes.*

first example consists of a curved domain containing holes and thin pieces, and the second example is a multi-phase elliptic interface problem with small circles, filaments, and cusps in the interface geometry. In both cases, we make use of a recently developed framework for computing high-order accurate multi-phase multi-physics using implicitly defined meshes [151, 152]. The framework shares some aspects with cut-cell techniques wherein a level set function defining the domain geometry or internal interfaces is used to cut through the cells of a background quadtree or octree; tiny cut cells are then merged with neighboring cells to create a mesh in which the shapes of interfacial elements are defined implicitly by the level set function. Quadrature rules for curved elements and nontrivial mesh faces are then computed using high-order accurate schemes for computing integrals on implicitly defined domains restricted to hyperrectangles [150]; these quadrature schemes are then used in the DG weak formulation, e.g., for computing mass matrices and the lifting operator L on the finest-level mesh.

In both examples, we consider an elliptic PDE problem with Dirichlet boundary conditions. The Dirichlet penalty parameter is chosen to scale inversely with h, the typical element size on the finest mesh, such that $\tau_D = 100/h$; the value 100 was determined empirically as being approximately the smallest possible while giving good multigrid performance. To measure the convergence rate, we apply the MGPCG method to a homogeneous problem with random nonzero initial condition and measure the average convergence rate using (4.4.1).

The first example of a single-phase Poisson problem on a curved domain is illustrated in Figure 4.8 and consists of a figure-eight domain with two holes surrounded by thin segments. It is important to note that the illustrated mesh hierarchy is implicitly formed by our operator coarsening scheme in Algorithm 4.3.1 and it is only the finest-level mesh which is built. On coarser levels, the elements are agglomerated according to the coarsening of the background quadtree; in particular, we note that the bottom level consists of a single element containing two holes. Using the operator

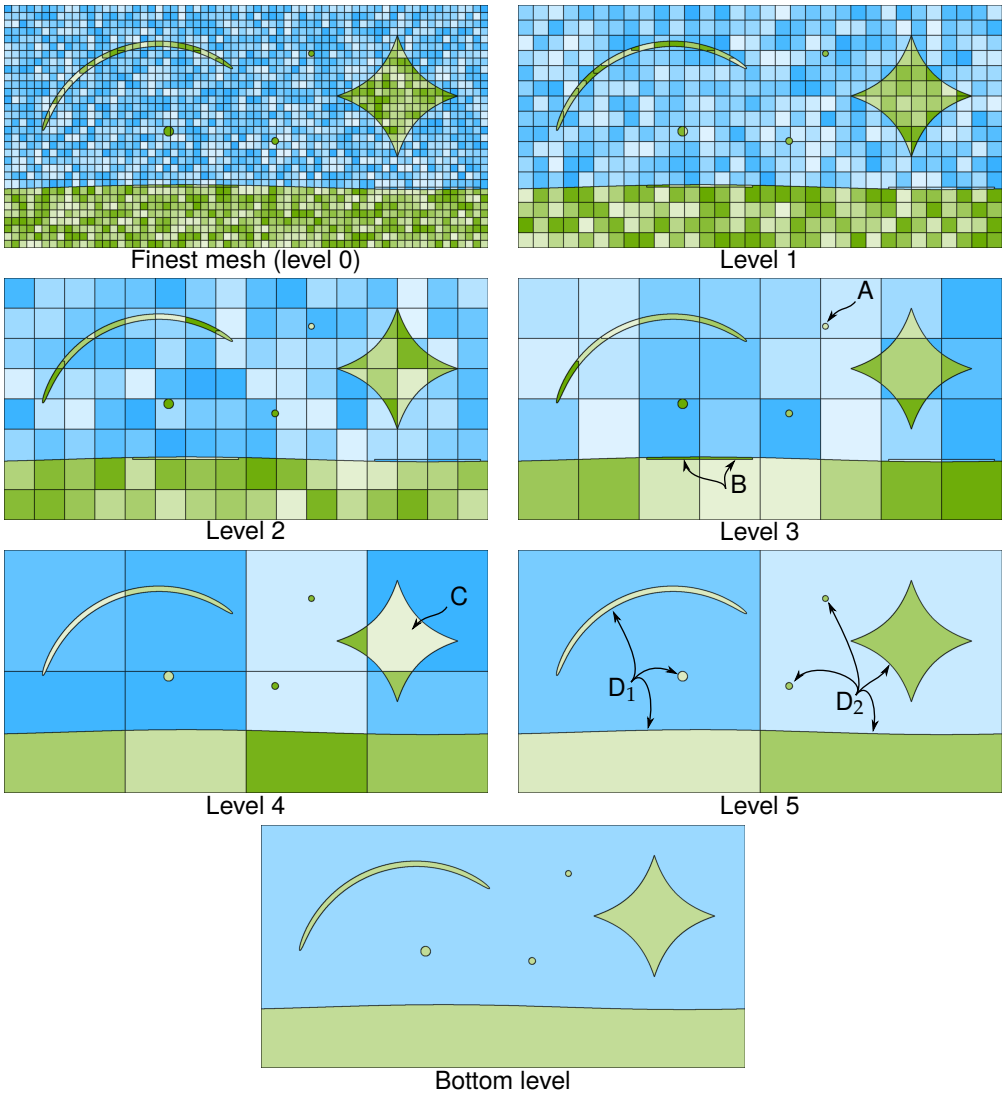


Figure 4.9: *A test case involving a multi-phase elliptic interface problem. Depicted is the implied h-multigrid hierarchy, wherein the elements are randomly colored a shade of green or blue for phase one or two, respectively. Similar to Figure* 4.8*, only the finest mesh is explicitly constructed; on coarser levels, the discrete gradient and penalty operators and mass matrices are constructed top-down at each level (see Algorithm* 4.3.1*). Note that the bottom level of the hierarchy consists of a mesh of just two elements—the blue element has one component with five holes, whereas the green element consists of six connected components. See the discussion following* (4.4.2) *for a description of the noted features* A*,* B*,* C*,* D_1*, and* D_2*.*

coarsening strategy, there is no need to compute quadrature rules for the coarse levels of the mesh hierarchy—the quadrature rules from the fine mesh are effectively coarsened automatically. The results for solving the Dirichlet problem (4.2.1) using flux coarsening[23] and h-multigrid on the curved domain of Figure 4.8 are shown in Figure 4.10 (left); we observe good multigrid convergence factors of $\rho \approx 0.05$–0.2, nearly independent of grid size, for $1 \leq p \leq 5$.

[23]Results using primal coarsening are similar to previous examples that use primal coarsening—poor multigrid performance is observed that degrades with mesh size—and have been omitted for brevity.

The second example considers a two-phase elliptic interface problem in a rectangular domain illustrated in Figure 4.9. The corresponding PDE consists of solving

$$\begin{aligned} -\nabla \cdot (\mu_i \nabla u) = f \quad \text{in } \Omega_i, \qquad & [\![u]\!] = g_\Gamma \quad \text{on } \Gamma, \\ u = g_\partial \quad \text{on } \partial\Omega, \qquad & [\![\mu \nabla u \cdot \boldsymbol{n}]\!] = h_\Gamma \quad \text{on } \Gamma, \end{aligned} \tag{4.4.2}$$

where Γ is the interface between phases Ω_1 (green region, with ellipticity coefficient[24] $\mu_1 = 1$) and Ω_2 (blue region, with $\mu_2 = 4$), and $[\![\cdot]\!]$ denotes the interfacial jump in the indicated quantity. In this test case, the geometry of the interface has been designed to be challenging—the crescent shape is long and thin; there are three isolated, small circles; and the star shape has sharp cusp-like corners. Once more we note that the illustrated hierarchy in Figure 4.9 is implicitly formed by the operator coarsening strategy, and only the finest-level mesh is actually built. However, we note that the agglomeration strategy for this multi-phase problem is slightly different from the previous test examples—here, elements are only agglomerated with elements belonging to the same phase. Thus, the interface remains sharp throughout all levels of the hierarchy, and this can dramatically improve the performance of multigrid methods for elliptic interface problems, especially when using high-order accurate techniques [151]. Owing to this agglomeration strategy, coarse mesh levels can have intricate element shapes. Some example features are noted in Figure 4.9—A indicates a tiny green-phase element surrounded by a large blue-phase element; B indicates two sliver elements; C indicates an element whose cusp-like corners would make it rather difficult to apply a black-box quadrature scheme if one were to explicitly build the coarse-level mesh; and $\mathsf{D}_{1,2}$ show two green-phase elements, each with multiple connected components (three for D_1 and four for D_2), which is perhaps rather unusual for a finite element method. Another aspect which motivated the present work on operator coarsening is that it would be nearly impossible to directly apply the cell merging algorithms underlying implicitly defined meshes [151] to these coarse levels. Although elements with extreme shapes like these (especially tiny elements next to large elements) can traditionally be of concern for numerical discretization of PDEs, according to our tests they pose no problem when present in the coarse levels of a multigrid solver. Results for solving the homogeneous version of (4.4.2) (f, g_∂, g_Γ, and h_Γ all zero) with a random nonzero initial guess using flux operator coarsening are shown in Figure 4.10 (right). In this case of an implicitly defined mesh for which different cell merging decisions take place depending on the refinement of the background grid, leading to different mesh topologies as n is increased, we naturally expect some amount of noise in ρ. For the majority of grid sizes, we see that the convergence factor ρ is relatively constant, taking values $\rho \approx 0.15$–0.3 reflective of the challenging interface geometry; meanwhile, for the largest mesh corresponding

[24] The chosen multi-phase elliptic interface problem has a somewhat mild coefficient jump of a factor of four across the interface. For much larger ratios, e.g., 10^3 to 10^6 and beyond, the performance degrades. In these cases, modifications to the LDG discretization can improve accuracy, conditioning, and multigrid performance [153].

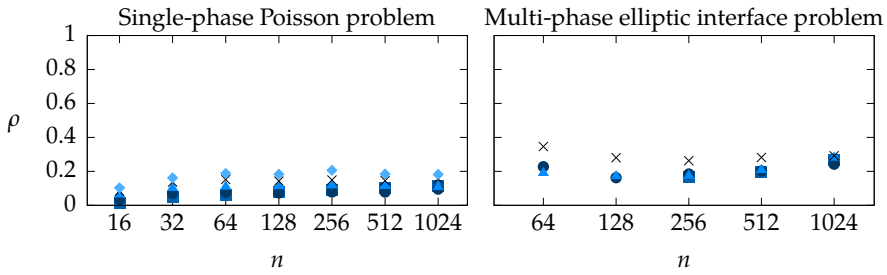


Figure 4.10: *h-multigrid convergence factors for flux coarsening and MGPCG applied to the LDG discretization of (left) the single-phase Poisson problem on the curved domain shown in Figure* 4.8 *and (right) the multi-phase elliptic interface test problem in* (4.4.2) *on the domain shown in Figure* 4.9*. The grid size n is the number of cells of the background Cartesian grid required to cover the longest extent of the domain. The plot markers indicate different polynomial orders:* ■ , ● , ▲ , × , *and* ◆ *denote* $p = 1, 2, 3, 4,$ *and* 5*, respectively.*

to $n = 1024$, the slight increase in the convergence rate ρ for all p is attributed to the increased ill-conditioning of the system.

Chapter 5

Conclusion and future directions

This thesis presented three fast methods for the high-order accurate numerical solution of elliptic PDEs. Chapter 2 developed optimal complexity Poisson solvers that achieve spectral accuracy by exploiting the separated spectra of differential operators discretized by ultraspherical polynomials. There, we employed ADI as a direct solver to solve Sylvester matrix equations for Poisson problems on the rectangle, cylinder, sphere, and cube.

To move beyond simple geometries, we developed the ultraspherical spectral element method for solving elliptic PDEs with high-order polynomials on unstructured meshes in Chapter 3. Using a hierarchical analogue of Gaussian elimination for domain decomposition, we created a fast direct solver for our SEM that computes the solution in $\mathcal{O}(p^4/h^3)$ operations and allows for fast repeated solves, enabling the acceleration of implicit time-steppers. We packaged the method into the `ultraSEM` software, which is designed for fast spectral element computation and hp-adaptivity with polynomials of very high degree.

In Chapter 4, we presented an hp-multigrid method for LDG discretizations of elliptic problems that is based on coarsening the discrete gradient and divergence operators from the flux formulation. We showed that coarsening fine-grid operators in this way results in a method that is equivalent to pure geometric multigrid, but avoids the need to compute quantities associated with coarse meshes, such as lifting operators and quadrature rules. Convergence factors were shown to be nearly independent of both mesh size h and polynomial order p for the demonstrated test problems on uniform Cartesian grids, adaptively refined meshes, and implicitly defined meshes on complex geometries.

* * *

We now briefly describe some potential extensions of the work in thesis.

Spectral element methods

The ultraspherical spectral element method allows for sparse, very high-order discretizations of elliptic PDEs on two-dimensional meshes to be solved efficiently.

Though heavily optimized, our `ultraSEM` software—which implements the element method in MATLAB—is far from ready for large-scale use. In order to realize the potential of hp-adaptivity with large p in high-performance settings, significant advancements to `ultraSEM` are necessary:

- **Iterative methods and robust preconditioners.** Iterative methods are typically avoided in the global spectral methods community due to their sensitivity to conditioning, despite the fact that many spectral discretizations possess fast, FFT-based matrix-vector products. Recently, Krylov methods based on operator-function products have performed well for solving Poisson problems on the square with careful preconditioning [77]. For problems on meshed geometries or for more complex elliptic PDEs, it may be possible to build a robust preconditioner based on `ultraSEM`. Using a continuous analogue of the singular value decomposition for partial differential operators, near-optimal Kronecker product preconditioners may be constructed at the continuous level, which can be inverted by `ultraSEM` and applied in an iterative method. Combine `ultraSEM` with Newton iteration may also allow `ultraSEM` to solve nonlinear problems.

- **Stability.** It is unknown if the merge step in the HPS scheme is stable. That is, does the merged solution operator S_Γ always exist? And if so, how does the conditioning of the merged solution operator S_Γ depend on the conditioning of its children? While preliminary experiments indicate that the conditioning of S_Γ does not depend on the merge order of its children, it is unclear as to the role that merge order plays in the HPS scheme.

- **Optimal complexity in h and p.** In d dimensions, the number of degrees of freedom in a (uniform) spectral element method scales as $N \approx (p/h)^d$. Thus, a spectral element method with optimal computational complexity would scale as $\mathcal{O}(p^2/h^2)$ in two dimensions or $\mathcal{O}(p^3/h^3)$ in three dimensions. Currently, the ultraspherical spectral element method has an overall complexity of $\mathcal{O}(p^4/h^3)$ in two dimensions, and can be extended to three-dimensional geometries with a complexity of $\mathcal{O}(p^6/h^6)$. It is an open question as to whether an optimal complexity SEM exists in two or three dimensions. It is possible that further structure in the almost block-banded matrices created by the ultraspherical spectral method may be exploited using low-rank compression techniques, to further reduce the complexity by a factor of p.

We are hoping to apply `ultraSEM` to problems such as advection-dominated fluid flow and high-frequency scattering, where low-order methods can artificially pollute the solution.

Discontinuous Galerkin methods

Designing geometric multigrid methods for LDG discretizations of elliptic problems requires careful treatment of the discrete Laplacian operator to achieve good multigrid performance. A number of interesting questions arise from generalizing operator-coarsening approaches:

- **Beyond LDG.** Though most of the analysis in Chapter 4 focused on the LDG method, we believe that the essential observation applies to other forms of DG discretization of elliptic problems, particularly those in which lifting operators enter the numerical flux for $\boldsymbol{q}$. A more thorough analysis for other DG methods and more general choices of numerical fluxes would be required to determine whether the multigrid method described here extends to other methods, such as CDG or HDG. Similarly, though we have employed equal-order elements in this work, i.e., polynomials of the same degree for both u and $\boldsymbol{q}$, operator-coarsening for mixed-order elements [36] would be an interesting topic for future investigation.

- **Algebraic multigrid.** Algebraic multigrid (AMG) methods build a hierarchy of coarse operators directly from a given matrix, by identifying fine and coarse unknowns and constructing interpolation and restriction operators that preserve algebraically smooth error. For finite element problems on unstructured and adaptively-refined meshes, and for problems with anisotropic variable coefficients, the algebraic multigrid method can be an efficient solver when geometric coarsening is cumbersome or impossible. The idea of coarsening the divergence and gradient operators separately may be useful for AMG methods, which currently treat the discrete Laplacian operator in its entirety as a black box. Indeed, black-box AMG algorithms applied to LDG discretizations appear to struggle [122]; perhaps applying AMG separately to the divergence and gradient operators in the flux formulation may yield better results. The AMG process may identify different coarse-grid unknowns for each operator, and so a unified set of coarse-grid unknowns would need to be constructed in order to compose the divergence and gradient operators in a consistent manner.

- **Unstructured meshes.** Chapter 4 considered structured meshes (Cartesian, quadtree, and octree meshes) as well as semi-unstructured, nonconforming, implicitly defined meshes that result from cell merging procedures (see Figures 4.8 and 4.9); applying the multigrid ideas presented here to problems involving more general unstructured meshes is an area for future investigation. In this setting, it may be worthwhile to consider different types of relaxation methods owing to their critical role in the overall efficacy of a multigrid method. For example, additive Schwarz smoothers have been shown effective on non-nested polygonal meshes resulting from agglomeration procedures [8]; these smoothers could be studied in the flux coarsening context as well.

We are also investigating the development of high-order DG methods for Eulerian solid mechanics. Recent work on the reference map technique [95, 96, 106] has demonstrated the success of low-order finite difference methods for the simulation of finite-strain elasticity in an Eulerian reference frame. Such a formulation allows for fluid–solid coupling to be handled naturally, as both fluid and solid can be tracked on the same fixed mesh [61, 94, 148, 161, 170]. However, the dissipation inherent in finite difference schemes can make accurate simulation challenging, e.g., in rapidly vibrating objects with low physical dissipation such as mechanical resonators [84] and tuning forks [72], or in objects with large spatial variation in elastic moduli such as the Earth's interior. A DG formulation of the reference map technique would allow for the high-order simulation of large-deformation solids with minimal energy loss, and could naturally be coupled to a high-order accurate method for fluid simulation to yield a fully Eulerian, high-order method for fluid–solid interaction.

Appendix A

MATLAB code to compute ADI shifts

Below we provide the MATLAB code that we use to compute the ADI shifts in (2.2.4). Readers may notice that in (2.2.4) the arguments of the complete elliptic integral and Jacobi elliptic functions involve $\sqrt{1-1/\alpha^2}$, while the arguments in the code involve $1-1/\alpha^2$, i.e., square roots are missing in the code. This is an esoteric MATLAB convention of the `ellipke` and `ellipj` commands, which we believe is for numerical accuracy. If one attempts to rewrite our code in another programming language, then one needs to be careful about the conventions in the analogues of the `ellipke` and `ellipj` commands.

```
function [p, q] = ADIshifts(a, b, c, d, tol)
% ADISHIFTS  ADI shifts for AX-XB=F when the eigenvalues of A (B) are in [a,b] and
% the eigenvalues of B (A) are in [c,d]. WLOG, we require that a<b<c<d and 0<tol<1.

gam = (c-a)*(d-b)/(c-b)/(d-a);                  % Cross-ratio of a,b,c,d
% Calculate Mobius transform T:{-alp,-1,1,alp}->{a,b,c,d} for some alp:
alp = -1 + 2*gam + 2*sqrt(gam^2-gam);           % Mobius exists with this t
A = det([-a*alp a 1; -b b 1 ; c c 1]);          % Determinant formulae for Mobius
B = det([-a*alp -alp a; -b -1 b ; c 1 c]);
C = det([-alp a 1; -1 b 1 ; 1 c 1]);
D = det([-a*alp -alp 1; -b -1 1; c 1 1]);
T = @(z) (A*z+B)./(C*z+D);                      % Mobius transfom
J = ceil( log(16*gam)*log(4/tol)/pi^2 );        % No. of ADI iterations
if ( alp < 1e7 )
    K = ellipke( 1-1/alp^2 );                   % ADI shifts for [-1,-1/t]&[1/t,1]
    [~, ~, dn] = ellipj((1/2:J-1/2)*K/J,1-1/alp^2);
else                                            % Prevent underflow when alp large
    K = (2*log(2)+log(alp)) + (-1+2*log(2)+log(alp))/alp^2/4;
    m1 = 1/alp^2;
    u = (1/2:J-1/2)*K/J;
    dn = sech(u) + .25*m1*(sinh(u).*cosh(u)+u).*tanh(u).*sech(u);
end
p = T( -alp*dn ); q = T( alp*dn );              % ADI shifts for [a,b]&[c,d]

end
```

Appendix B

Bounding eigenvalues

In section 2.3 a spectral discretization of Poisson's equation on the square is derived as $\tilde{A}X - X\tilde{B} = F$, where $\tilde{A}$ is a real symmetric pentadiagonal matrix and $\tilde{B} = -\tilde{A}^T$. Here, we prove that P2 holds for the Sylvester matrix equation by showing that $\sigma(\tilde{A}) \in [-1/2, -1/(2n^4)]$. The bound on the spectrum of $\tilde{A}$ is stated in the following lemma, which we use to determine the number of ADI iterations for our fast Poisson solver on the square.

Lemma B.0.1. *Let $\tilde{A} \in \mathbb{C}^{n\times n}$ be the matrix given in* (2.3.9)*. Then*

$$\sigma(\tilde{A}) \subset \left[-\frac{1}{2}, -\frac{1}{2n^4}\right], \tag{B.0.1}$$

where $\sigma(\tilde{A})$ is the spectrum of $\tilde{A}$.

Proof. The matrix $\tilde{A} = D_s^{-1}AD_s = D_s^{-1}D^{-1}MD_s = D^{-1/2}MD^{-1/2}$ for $D_s = D^{-1/2}$, where D is a diagonal matrix. The matrix M here represents multiplication by $1 - x^2$ in the $C^{(3/2)}$ basis and thus M is positive definite as its eigenvalues are the values of $1 - x_j^2$, where x_j are the $C^{(3/2)}$ Gauss quadrature nodes with $-1 < x_j < 1$. The matrix D is negative definite as its diagonal entries are all negative, which implies that $D^{-1/2} = i\,\mathrm{Re}(D^{-1/2})$. Therefore we can write $\tilde{A} = -C^TC$, where $C = M^{1/2}D^{-1/2}$ and so $\tilde{A}$ is negative definite.

The (absolute) minimal eigenvalue of $\tilde{A}$ is given by $\lambda_{\min}(\tilde{A}) = \lambda_{\min}(-C^TC) = \min_{\|v\|_2=1} \|Cv\|_2^2$. Let w be proportional to the normalized vector minimizing $\|M^{1/2}w\|_2^2$, which is equal to $(1 - x_0^2)\|w\|_2^2$, where x_0 is the leftmost $C^{(3/2)}$ Gauss node. From the bounds given in [56], one may verify that

$$1 - x_0^2 \geq \frac{b + (n-2)\sqrt{\delta}}{a} \geq \frac{1}{n^2}, \qquad \begin{aligned} a &= (1+2n)(10 + n/2 + n^2) \\ b &= n^3 + n^2 + n/2 + 2 \\ \delta &= n^4 + 6n^3 + 13n^2 + 36n + 16 \end{aligned}$$

for all $n > 0$. Then if we set $v = D^{1/2}w$, we obtain

$$\|Cv\|_2^2 = \|M^{1/2}w\|_2^2 = (1 - x_0^2)\|w\|_2^2 \geq \frac{1 - x_0^2}{(n-1)(n+2)+2}\|v\|_2^2 \geq \frac{1}{2n^4}.$$

for all $n > 0$.

The (absolute) maximal eigenvalue of $\tilde{A}$ is given by

$$\lambda_{\max}(\tilde{A}) = \|\tilde{A}\|_2 = \|D^{-1/2}MD^{-1/2}\|_2 \leq \|D^{-1/2}\|_2\|M\|_2\|D^{-1/2}\|_2 \leq 1/2,$$

since $\|D^{-1/2}\|_2 = 1/\sqrt{2}$ and $\|M\|_2 \leq 1$. □

Appendix C

Constructing an interpolant of Dirichlet data

Consider Dirichlet data on the square domain $[-1,1]^2$. Let g_{left}, g_{right}, g_{bottom}, $g_{\text{top}} \in \mathbb{C}^n$ represent the univariate Chebyshev coefficients of Dirichlet data on the left, right, bottom, and top of the square domain, respectively, and assume that compatibility conditions are met so that the function values match at the four corners of the square. Then g_{left}, g_{right}, g_{bottom}, and g_{top} encode $4n-4$ degrees of freedom, and a bivariate interpolant may be constructed whose Chebyshev coefficients $X_{\text{bc}} \in \mathbb{C}^{n\times n}$ are given by specifying $4n-4$ entries in the first two columns and rows, i.e.,

$$
\begin{aligned}
X_{\text{bc}}(1,3:n) &= \frac{g_{\text{bottom}}(3:n) + g_{\text{top}}(3:n)}{2}, \\
X_{\text{bc}}(2,3:n) &= \frac{g_{\text{bottom}}(3:n) - g_{\text{top}}(3:n)}{2}, \\
X_{\text{bc}}(3:n,1) &= \frac{g_{\text{right}}(3:n) + g_{\text{left}}(3:n)}{2}, \\
X_{\text{bc}}(3:n,2) &= \frac{g_{\text{right}}(3:n) - g_{\text{left}}(3:n)}{2}, \\
X_{\text{bc}}(1:2,1) &= \frac{g_{\text{right}}(1:2) + g_{\text{left}}(1:2)}{2} - \sum_{\substack{k=3 \\ k \text{ odd}}}^{n} X_{\text{bc}}(1:2,k), \\
X_{\text{bc}}(1:2,2) &= \frac{g_{\text{right}}(1:2) - g_{\text{left}}(1:2)}{2} - \sum_{\substack{k=4 \\ k \text{ even}}}^{n} X_{\text{bc}}(1:2,k),
\end{aligned}
$$

with $X_{\text{bc}}(3:n,3:n) = 0$. The function represented by the bivariate Chebyshev coefficients X_{bc} matches the Dirichlet data on the four sides of the square domain, and its smoothness is limited only by the smoothness of the Dirichlet data. Similar interpolating functions can be constructed to match Dirichlet data on the cylinder, sphere, and cube.

Appendix D

Tables of multigrid convergence factors

Here we present the raw multigrid convergence factors used to generate Figures 4.2–4.4, 4.7, and 4.10 of Chapter 4. The data are arranged below in tables, with grid size, polynomial order, coarsening strategy, and solver indicated. Omitted data points correspond to simulations whose memory requirements approximately exceed 120 GB.

Table D.1: *h-multigrid convergence factors for primal and flux coarsening applied to the LDG discretization of Poisson's equation on a uniform $n \times n$ Cartesian grid as $h \to 0$ using V-cycles (see Figure 4.2).*

		n							
	p	4	8	16	32	64	128	256	512
Primal coarsening	1	0.17	0.25	0.35	0.48	0.65	0.80	0.88	0.92
	2	0.15	0.21	0.26	0.41	0.58	0.71	0.82	0.90
	3	0.23	0.29	0.40	0.54	0.66	0.79	0.85	0.91
	4	0.19	0.25	0.34	0.48	0.62	0.75	0.85	0.91
	5	0.26	0.33	0.40	0.59	0.71	0.81	0.88	0.92
Flux coarsening	1	0.14	0.15	0.15	0.16	0.15	0.15	0.15	0.15
	2	0.10	0.10	0.09	0.09	0.09	0.08	0.08	0.08
	3	0.18	0.18	0.17	0.16	0.16	0.16	0.16	0.16
	4	0.16	0.15	0.15	0.15	0.15	0.14	0.14	0.14
	5	0.23	0.23	0.23	0.23	0.23	0.23	0.23	0.23

Table D.2: *h-multigrid convergence factors for primal and flux coarsening applied to the LDG discretization of Poisson's equation on a uniform $n \times n$ Cartesian grid as $h \to 0$ using MGPCG (see Figure 4.2).*

		n							
	p	4	8	16	32	64	128	256	512
Primal coarsening	1	0.04	0.09	0.13	0.20	0.30	0.40	0.50	0.60
	2	0.05	0.08	0.12	0.18	0.24	0.34	0.44	0.54
	3	0.08	0.11	0.15	0.21	0.29	0.38	0.48	0.57
	4	0.07	0.09	0.14	0.20	0.28	0.37	0.46	0.56
	5	0.11	0.13	0.17	0.23	0.34	0.43	0.51	0.58
Flux coarsening	1	0.05	0.06	0.06	0.07	0.07	0.07	0.07	0.08
	2	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04
	3	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07
	4	0.07	0.07	0.07	0.07	0.06	0.07	0.07	0.07
	5	0.10	0.10	0.10	0.10	0.10	0.09	0.09	0.09

Table D.3: *h-multigrid convergence factors for primal and flux coarsening applied to the LDG discretization of Poisson's equation on a uniform $n \times n \times n$ Cartesian grid as $h \to 0$ using V-cycles (see Figure 4.3).*

		n				
	p	4	8	16	32	64
Primal coarsening	1	0.19	0.27	0.40	0.56	0.71
	2	0.17	0.23	0.34	0.49	0.62
	3	0.25	0.32	0.45	0.57	–
	4	0.22	0.27	0.40	–	–
	5	0.31	0.38	0.47	–	–
Flux coarsening	1	0.17	0.19	0.20	0.20	0.21
	2	0.14	0.13	0.13	0.13	0.12
	3	0.21	0.22	0.21	0.21	–
	4	0.21	0.20	0.19	–	–
	5	0.29	0.29	0.29	–	–

Table D.4: *h-multigrid convergence factors for primal and flux coarsening applied to the LDG discretization of Poisson's equation on a uniform $n \times n \times n$ Cartesian grid as $h \to 0$ using MGPCG (see Figure 4.3).*

		n				
	p	4	8	16	32	64
Primal coarsening	1	0.06	0.10	0.16	0.23	0.33
	2	0.07	0.10	0.14	0.20	0.28
	3	0.10	0.13	0.18	0.25	–
	4	0.09	0.12	0.17	–	–
	5	0.13	0.16	0.21	–	–
Flux coarsening	1	0.06	0.08	0.09	0.09	0.10
	2	0.06	0.06	0.06	0.06	0.06
	3	0.09	0.09	0.09	0.09	–
	4	0.09	0.09	0.09	–	–
	5	0.12	0.12	0.12	–	–

Table D.5: *p-multigrid convergence factors for primal and flux coarsening applied to the LDG discretization of Poisson's equation on 2D uniform grids using MGPCG (see Figure 4.4).*

		p			
	n	1	2	4	8
Primal coarsening	4	0.04	0.06	0.10	0.20
	8	0.09	0.12	0.19	0.36
	16	0.13	0.19	0.32	0.51
	32	0.20	0.30	0.45	0.63
	64	0.30	0.43	0.57	0.71
	128	0.40	0.52	0.67	0.79
	256	0.50	0.63	0.75	0.84
	512	0.60	0.71	0.80	–
Flux coarsening	4	0.05	0.04	0.04	0.08
	8	0.06	0.05	0.05	0.08
	16	0.06	0.06	0.06	0.08
	32	0.07	0.06	0.07	0.08
	64	0.07	0.07	0.08	0.08
	128	0.07	0.08	0.08	0.08
	256	0.07	0.08	0.08	0.08
	512	0.08	0.08	0.08	–

Table D.6: *p-multigrid convergence factors for primal and flux coarsening applied to the LDG discretization of Poisson's equation on 3D uniform grids using MGPCG (see Figure 4.4).*

		p			
	n	1	2	4	8
Primal coarsening	4	0.06	0.08	0.13	0.25
	8	0.10	0.15	0.26	0.41
	16	0.16	0.24	0.38	–
	32	0.23	0.35	0.50	–
	64	0.33	0.46	–	–
Flux coarsening	4	0.06	0.04	0.06	0.12
	8	0.08	0.07	0.07	0.12
	16	0.09	0.08	0.08	–
	32	0.09	0.08	0.08	–
	64	0.10	0.08	–	–

Table D.7: *h-multigrid convergence factors for primal and flux coarsening applied to the LDG discretization of Poisson's equation on an adaptively refined grid in 2D using MGPCG (see Figure 4.7). The effective grid size h_{eff} is the size of the smallest element on the finest grid.*

		$1/h_{\text{eff}}$					
	p	16	32	64	128	256	512
Primal coarsening	1	0.03	0.09	0.17	0.25	0.37	0.47
	2	0.04	0.08	0.15	0.21	0.32	0.42
	3	0.07	0.11	0.18	0.25	0.35	0.44
	4	0.06	0.10	0.17	0.26	0.37	0.45
	5	0.09	0.13	0.19	0.29	0.38	0.47
Flux coarsening	1	0.02	0.06	0.08	0.09	0.12	0.14
	2	0.03	0.05	0.09	0.11	0.13	0.14
	3	0.06	0.08	0.11	0.15	0.17	0.18
	4	0.06	0.08	0.13	0.18	0.19	0.21
	5	0.08	0.12	0.16	0.21	0.22	0.25

Table D.8: *h-multigrid convergence factors for primal and flux coarsening applied to the LDG discretization of Poisson's equation on an adaptively refined grid in 3D using MGPCG (see Figure* 4.7*). The effective grid size h_{eff} is the size of the smallest element on the finest grid.*

		$1/h_{\text{eff}}$				
	p	16	32	64	128	256
Primal coarsening	1	0.05	0.11	0.19	0.28	0.39
	2	0.05	0.10	0.18	0.28	–
	3	0.08	0.14	0.21	0.30	–
	4	0.08	0.13	0.21	–	–
	5	0.11	0.16	–	–	–
Flux coarsening	1	0.04	0.08	0.10	0.14	0.14
	2	0.04	0.07	0.14	0.18	–
	3	0.08	0.11	0.17	0.21	–
	4	0.08	0.11	0.21	–	–
	5	0.12	0.15	–	–	–

Table D.9: *h-multigrid convergence factors for flux coarsening and MGPCG applied to the LDG discretization of the single-phase Poisson problem on the curved domain shown in Figure* 4.8 *(see Figure* 4.10*). The grid size n is the number of cells of the background Cartesian grid required to cover the longest extent of the domain.*

	n						
p	16	32	64	128	256	512	1024
1	0.01	0.05	0.06	0.08	0.09	0.10	0.11
2	0.04	0.07	0.08	0.07	0.08	0.08	0.10
3	0.06	0.10	0.11	0.11	0.12	0.11	0.11
4	0.08	0.14	0.15	0.14	0.15	0.15	0.16
5	0.10	0.16	0.19	0.18	0.21	0.18	0.18

Table D.10: *h-multigrid convergence factors for flux coarsening and MGPCG applied to the LDG discretization of the multi-phase elliptic interface test problem in* (4.4.2) *on the domain shown in Figure* 4.9 *(see Figure* 4.10*). The grid size n is the number of cells of the background Cartesian grid required to cover the longest extent of the domain.*

	n				
p	64	128	256	512	1024
1	–	–	0.17	0.20	0.27
2	0.23	0.16	0.18	0.20	0.24
3	0.19	0.17	0.17	0.21	0.28
4	0.35	0.28	0.26	0.28	0.29

References

[1] M. Ainsworth and J. T. Oden, *A posteriori error estimation in finite element analysis*, Comput. Methods Appl. Mech. Eng., 142 (1997), pp. 1–88, https://doi.org/10.1016/S0045-7825(96)01107-3.

[2] M. Ainsworth and B. Senior, *An adaptive refinement strategy for hp-finite element computations*, Appl. Numer. Math., 26 (1998), pp. 165–178, https://doi.org/10.1016/S0168-9274(97)00083-4.

[3] R. Anderson, J. Andrej, A. Barker, J. Bramwell, J.-S. Camier, J. Cerveny, V. Dobrev, Y. Dudouit, A. Fisher, T. Kolev, W. Pazner, M. Stowell, V. Tomov, I. Akkerman, J. Dahm, D. Medina, and S. Zampini, *MFEM: A modular finite element methods library*, Comput. Math. Appl., (2020), https://doi.org/10.1016/j.camwa.2020.06.009.

[4] P. Antonietti, B. Ayuso de Dios, S. Brenner, and L.-y. Sung, *Schwarz methods for a preconditioned WOPSIP method for elliptic problems*, Comput. Methods Appl. Math., 12 (2012), pp. 241–272, https://doi.org/doi:10.2478/cmam-2012-0021.

[5] P. Antonietti, M. Sarti, and M. Verani, *Multigrid algorithms for hp-discontinuous Galerkin discretizations of elliptic problems*, SIAM J. Numer. Anal., 53 (2015), pp. 598–618, https://doi.org/10.1137/130947015.

[6] P. F. Antonietti, P. Houston, X. Hu, M. Sarti, and M. Verani, *Multigrid algorithms for hp-version interior penalty discontinuous Galerkin methods on polygonal and polyhedral meshes*, Calcolo, 54 (2017), pp. 1169–1198, https://doi.org/10.1007/s10092-017-0223-6.

[7] P. F. Antonietti, P. Houston, and G. Pennesi, *Fast numerical integration on polytopic meshes with applications to discontinuous Galerkin finite element methods*, J. Sci. Comput., 77 (2018), pp. 1339–1370, https://doi.org/10.1007/s10915-018-0802-y.

[8] P. F. Antonietti and G. Pennesi, *V-cycle multigrid algorithms for discontinuous Galerkin methods on non-nested polytopic meshes*, J. Sci. Comput., 78 (2019), pp. 625–652, https://doi.org/10.1007/s10915-018-0783-x.

[9] Argonne National Laboratory, *Nek5000*, 2020, https://nek5000.mcs.anl.gov (accessed 03/24/2020). Version 19.0.

[10] D. N. Arnold, *An interior penalty finite element method with discontinuous elements*, SIAM J. Numer. Anal., 19 (1982), pp. 742–760, https://doi.org/10.1137/0719052.

[11] D. N. Arnold, F. Brezzi, B. Cockburn, and L. D. Marini, *Unified analysis of discontinuous Galerkin methods for elliptic problems*, SIAM J. Numer. Anal., 39 (2002), pp. 1749–1779, https://doi.org/10.1137/S0036142901384162.

[12] U. M. Ascher, S. J. Ruuth, and B. T. R. Wetton, *Implicit-explicit methods for time-dependent partial differential equations*, SIAM J. Numer. Anal., 32 (1995), pp. 797–823, https://doi.org/10.1137/0732037.

[13] K. Atkinson and W. Han, *Spherical Harmonics and Approximations on the Unit Sphere: An Introduction*, vol. 2044 of Lecture Notes in Mathematics, Springer, Berlin, 2012, https://doi.org/10.1007/978-3-642-25983-8.

[14] A. Averbuch, M. Israeli, and L. Vozovoi, *A fast Poisson solver of arbitrary order accuracy in rectangular regions*, SIAM J. Sci. Comput., 19 (1998), pp. 933–952, https://doi.org/10.1137/S1064827595288589.

[15] T. Babb, A. Gillman, S. Hao, and P.-G. Martinsson, *An accelerated Poisson solver based on multidomain spectral discretization*, BIT Numer. Math., 58 (2018), pp. 851–879, https://doi.org/10.1007/s10543-018-0714-0.

[16] T. Babb, P.-G. Martinsson, and D. Appelö, *HPS accelerated spectral solvers for time dependent problems: Part I, Algorithms*, in Spectral and High Order Methods for Partial Differential Equations ICOSAHOM 2018, S. J. Sherwin, D. Moxey, J. Peiró, P. E. Vincent, and C. Schwab, eds., Cham, 2020, Springer International Publishing, pp. 131–141, https://doi.org/10.1007/978-3-030-39647-3_9.

[17] T. Babb, P.-G. Martinsson, and D. Appelö, *HPS accelerated spectral solvers for time dependent problems: Part II, Numerical experiments*, in Spectral and High Order Methods for Partial Differential Equations ICOSAHOM 2018, S. J. Sherwin, D. Moxey, J. Peiró, P. E. Vincent, and C. Schwab, eds., Cham, 2020, Springer International Publishing, pp. 155–166, https://doi.org/10.1007/978-3-030-39647-3_11.

[18] I. Babuška and B. Guo, *The h-p version of the finite element method*, Comput. Mech., 1 (1986), pp. 21–41, https://doi.org/10.1007/BF00298636.

[19] I. Babuška and M. Suri, *The h-p version of the finite element method with quasiuniform meshes*, ESAIM: Math. Model. Numer. Anal., 21 (1987), pp. 199–238, https://doi.org/10.1051/m2an/1987210201991.

[20] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, A. Dener, V. Eijkhout, W. D. Gropp, D. Karpeyev, D. Kaushik, M. G. Knepley, D. A. May, L. C. McInnes, R. T. Mills, T. Munson, K. Rupp, P. Sanan, B. F. Smith, S. Zampini, H. Zhang, and H. Zhang, *PETSc Web page*, 2019, https://www.mcs.anl.gov/petsc.

[21] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, A. Dener, V. Eijkhout, W. D. Gropp, D. Karpeyev, D. Kaushik, M. G. Knepley, D. A. May, L. C. McInnes, R. T. Mills, T. Munson, K. Rupp, P. Sanan, B. F. Smith, S. Zampini, H. Zhang, and H. Zhang, *PETSc users manual*, Tech. Report ANL-95/11 - Revision 3.13, Argonne National Laboratory, 2020, https://www.mcs.anl.gov/petsc.

[22] W. Bangerth, R. Hartmann, and G. Kanschat, *deal.II—a general-purpose object-oriented finite element library*, ACM Trans. Math. Softw., 33 (2007), pp. 24–es, https://doi.org/10.1145/1268776.1268779.

[23] R. H. Bartels and G. W. Stewart, *Solution of the matrix equation $AX + XB = C$*, Commun. ACM, 15 (1972), pp. 820–826, https://doi.org/10.1145/361573.361582.

[24] F. Bassi, A. Ghidoni, S. Rebay, and P. Tesini, *High-order accurate p-multigrid discontinuous Galerkin solution of the Euler equations*, Int. J. Numer. Methods Fluids, 60 (2008), pp. 847–865, https://doi.org/10.1002/fld.1917.

[25] F. Bassi and S. Rebay, *A high-order accurate discontinuous finite element method for the numerical solution of the compressible Navier–Stokes equations*, J. Comput. Phys., 131 (1997), pp. 267–279, https://doi.org/10.1006/jcph.1996.5572.

[26] F. Bassi, S. Rebay, G. Mariotti, S. Pedinotti, and M. Savini, *A high-order accurate discontinuous finite element method for inviscid and viscous turbomachinery flows*, in Proceedings of the 2nd European Conference on Turbomachinery Fluid Dynamics and Thermodynamics, Technologisch Instituut, Antwerpen, Belgium, 1997, pp. 99–109.

[27] P. Bastian, M. Blatt, and R. Scheichl, *Algebraic multigrid for discontinuous Galerkin discretizations of heterogeneous elliptic problems*, Numer. Lin. Alg. Appl., 19 (2012), pp. 367–388, https://doi.org/10.1002/nla.1816.

[28] B. Beckermann and A. Townsend, *On the singular values of matrices with displacement structure*, SIAM J. Matrix Anal. Appl., 38 (2017), pp. 1227–1248, https://doi.org/10.1137/16M1096426.

[29] P. Benner, R.-C. Li, and N. Truhar, *On the ADI method for Sylvester equations*, J. Comput. Appl. Math., 233 (2009), pp. 1035–1045, https://doi.org/10.1016/j.cam.2009.08.108.

[30] S. Beuchler and J. Schöberl, *New shape functions for triangular p-FEM using integrated Jacobi polynomials*, Numer. Math., 103 (2006), pp. 339–366, https://doi.org/10.1007/s00211-006-0681-2.

[31] R. T. Biedron, J. R. Carlson, J. M. Derlaga, P. A. Gnoffo, D. P. Hammond, W. T. Jones, B. Kleb, E. M. Lee-Rausch, E. J. Nielsen, M. A. Park, C. L. Rumsey, J. L. Thomas, K. B. Thompson, and W. A. Wood, *FUN3D manual: 13.6*, Tech. Report NASA/TM-2019-220416, NASA, 2019.

[32] J. P. Boyd, *Chebyshev and Fourier Spectral Methods*, Dover, Mineola, 2001.

[33] A. Brandt, *Multi-level adaptive solutions to boundary-value problems*, Math. Comput., 31 (1977), pp. 333–390, https://doi.org/10.2307/2006422.

[34] E. Braverman, M. Israeli, A. Averbuch, and L. Vozovoi, *A fast 3D Poisson solver of arbitrary order accuracy*, J. Comput. Phys., 144 (1998), pp. 109–136, https://doi.org/10.1006/jcph.1998.6001.

[35] S. C. Brenner and J. Zhao, *Convergence of multigrid algorithms for interior penalty methods*, Appl. Numer. Anal. Comput. Math., 2 (2005), pp. 3–18, https://doi.org/10.1002/anac.200410019.

[36] F. Brezzi, T. J. R. Hughes, L. D. Marini, and A. Masud, *Mixed discontinuous Galerkin methods for darcy flow*, J. Sci. Comput., 22 (2005), pp. 119–145, https://doi.org/10.1007/s10915-004-4150-8.

[37] F. Brezzi, G. Manzini, D. Marini, P. Pietra, and A. Russo, *Discontinuous Galerkin approximations for elliptic problems*, Numerical Methods for Partial Differential Equations, 16 (2000), pp. 365–378, https://doi.org/10.1002/1098-2426(200007)16:4<365::AID-NUM2>3.0.CO;2-Y.

[38] W. L. Briggs, V. E. Henson, and S. F. McCormick, *A Multigrid Tutorial, Second Edition*, SIAM, 2000, https://doi.org/10.1137/1.9780898719505.

[39] V. Britanak, P. C. Yip, and K. R. Rao, *Discrete Cosine and Sine Transforms: General Properties, Fast Algorithms and Integer Approximations*, Academic Press, Oxford, 2010.

[40] K. J. Burns, G. M. Vasil, J. S. Oishi, D. Lecoanet, and B. P. Brown, *Dedalus: A flexible framework for numerical simulations with spectral methods*, Phys. Rev. Research, 2 (2020), p. 023068, https://doi.org/10.1103/PhysRevResearch.2.023068.

[41] B. L. Buzbee, G. H. Golub, and C. W. Nielson, *On direct methods for solving Poisson's equations*, SIAM J. Numer. Anal., 7 (1970), pp. 627–656, https://doi.org/10.1137/0707049.

[42] C. Cantwell, D. Moxey, A. Comerford, A. Bolis, G. Rocco, G. Mengaldo, D. De Grazia, S. Yakovlev, J.-E. Lombard, D. Ekelschot, B. Jordi, H. Xu, Y. Mohamied, C. Eskilsson, B. Nelson, P. Vos, C. Biotto, R. Kirby, and S. Sherwin, *Nektar++: An open-source spectral/hp element framework*, Comput. Phys. Commun., 192 (2015), pp. 205–219, https://doi.org/10.1016/j.cpc.2015.02.008.

[43] C. Canuto, A. Quarteroni, M. Y. Hussaini, and T. A. Zang, *Spectral Methods: Evolution to Complex Geometries and Applications to Fluid Dynamics*, Scientific Computation, Springer, Berlin, 2007, https://doi.org/10.1007/978-3-540-30728-0.

[44] P. Castillo, B. Cockburn, I. Perugia, and D. Schötzau, *An a priori error analysis of the local discontinuous Galerkin method for elliptic problems*, SIAM J. Numer. Anal., 38 (2000), pp. 1676–1706, https://doi.org/10.1137/S0036142900371003.

[45] A. J. Chorin, *A numerical method for solving incompressible viscous flow problems*, J. Comput. Phys., 2 (1967), pp. 12–26, https://doi.org/10.1016/0021-9991(67)90037-X.

[46] A. J. Chorin, *Numerical solution of the Navier-Stokes equations*, Math. Comput., 22 (1968), pp. 745–762, https://doi.org/10.1090/S0025-5718-1968-0242392-2.

[47] B. Cockburn and B. Dong, *An analysis of the minimal dissipation local discontinuous Galerkin method for convection–diffusion problems*, J. Sci. Comput., 32 (2007), pp. 233–262, https://doi.org/10.1007/s10915-007-9130-3.

[48] B. Cockburn, J. Gopalakrishnan, and R. Lazarov, *Unified hybridization of discontinuous Galerkin, mixed, and continuous Galerkin methods for second order elliptic problems*, SIAM J. Numer. Anal., 47 (2009), pp. 1319–1365, https://doi.org/10.1137/070706616.

[49] B. Cockburn, G. Kanschat, I. Perugia, and D. Schötzau, *Superconvergence of the local discontinuous Galerkin method for elliptic problems on Cartesian grids*, SIAM J. Numer. Anal., 39 (2001), pp. 264–285, https://doi.org/10.1137/S0036142900371544.

[50] B. Cockburn, G. E. Karniadakis, and C.-W. Shu, *Discontinuous Galerkin Methods: Theory, Computation and Applications*, Springer, Berlin, 2000, https://doi.org/10.1007/978-3-642-59721-3.

[51] B. Cockburn and C.-W. Shu, *The local discontinuous Galerkin method for time-dependent convection-diffusion systems*, SIAM J. Numer. Anal., 35 (1998), pp. 2440–2463, https://doi.org/10.1137/S0036142997316712.

[52] J. W. Cooley and J. W. Tukey, *An algorithm for the machine calculation of complex Fourier series*, Math. Comput., 19 (1965), pp. 297–301, https://doi.org/10.1090/S0025-5718-1965-0178586-1.

[53] R. Courant, K. Friedrichs, and H. Lewy, *Über die partiellen Differenzengleichungen der mathematischen Physik*, Math. Ann., 100 (1928), pp. 32–74, https://doi.org/10.1007/BF01448839.

[54] B. N. Datta, *Numerical Linear Algebra and Applications*, SIAM, Philadelpha, PA, 2nd ed., 2010.

[55] L. Demkowicz, *Computing with hp-Adaptive Finite Elements: Volume 1: One and Two Dimensional Elliptic and Maxwell Problems*, CRC Press, 2006.

[56] D. K. Dimitrov and G. P. Nikolov, *Sharp bounds for the extreme zeros of classical orthogonal polynomials*, J. Approx. Theory, 162 (2010), pp. 1793–1804, https://doi.org/10.1016/j.jat.2009.11.006.

[57] J. Douglas and T. Dupont, *Interior penalty procedures for elliptic and parabolic Galerkin methods*, in Computing Methods in Applied Sciences, R. Glowinski and J. L. Lions, eds., Berlin, 1976, Springer, pp. 207–216, https://doi.org/10.1007/BFb0120591.

[58] T. A. Driscoll, N. Hale, and L. N. Trefethen, eds., *Chebfun Guide*, Pafnuty Publications, Oxford, 2014.

[59] M. G. Duffy, *Quadrature over a pyramid or cube of integrands with a singularity at a vertex*, SIAM J. Numer. Anal., 19 (1982), pp. 1260–1262, https://doi.org/10.1137/0719090.

[60] C. F. Dunkl and Y. Xu, *Orthogonal Polynomials of Several Variables*, Encyclopedia of Mathematics and its Applications, Cambridge University Press, Cambridge, 2nd ed., 2014, https://doi.org/10.1017/CBO9781107786134.

[61] T. Dunne, *An Eulerian approach to fluid–structure interaction and goal-oriented mesh adaptation*, Int. J. Numer. Methods Fluids, 51 (2006), pp. 1017–1039, https://doi.org/10.1002/fld.1205.

[62] L. C. Evans, *Partial Differential Equations*, vol. 19 of Graduate Studies in Mathematics, American Mathematical Society, Providence, 2nd ed., 2010.

[63] M. S. Fabien, M. G. Knepley, R. T. Mills, and B. M. Rivière, *Manycore parallel computing for a hybridizable discontinuous Galerkin nested multigrid method*, SIAM J. Sci. Comput., 41 (2019), pp. C73–C96, https://doi.org/10.1137/17M1128903.

[64] R. D. Falgout and U. M. Yang, *hypre: A library of high performance preconditioners*, in Proceedings of the International Conference on Computational Science-Part III, ICCS '02, Berlin, 2002, Springer, pp. 632–641.

[65] K. J. Fidkowski, T. A. Oliver, J. Lu, and D. L. Darmofal, *p-multigrid solution of high-order discontinuous Galerkin discretizations of the compressible Navier–Stokes equations*, J. Comput. Phys., 207 (2005), pp. 92–113, https://doi.org/10.1016/j.jcp.2005.01.005.

[66] B. Fornberg, *A pseudospectral approach for polar and spherical geometries*, SIAM J. Sci. Comput., 16 (1995), pp. 1071–1081, https://doi.org/10.1137/0916061.

[67] D. Fortunato, N. Hale, and A. Townsend. GitHub repository, 2020. https://github.com/danfortunato/ultraSEM.

[68] D. Fortunato, N. Hale, and A. Townsend, *The ultraspherical spectral element method*, Jun 2020, https://arxiv.org/abs/2006.08756.

[69] D. Fortunato, C. H. Rycroft, and R. Saye, *Efficient operator-coarsening multigrid schemes for local discontinuous Galerkin methods*, SIAM J. Sci. Comput., 41 (2019), pp. A3913–A3937, https://doi.org/10.1137/18M1206357.

[70] D. Fortunato and A. Townsend. GitHub repository, 2017. https://github.com/danfortunato/fast-poisson-solvers.

[71] D. Fortunato and A. Townsend, *Fast Poisson solvers for spectral methods*, IMA J. Numer. Anal., 40 (2020), pp. 1994–2018, https://doi.org/10.1093/imanum/drz034.

[72] B. Froehle and P.-O. Persson, *High-order accurate fluid–structure simulation of a tuning fork*, Comput. Fluids, 98 (2014), pp. 230–238, https://doi.org/10.1016/j.compfluid.2013.11.009.

[73] M. J. Gander and K. Santugini, *Cross-points in domain decomposition methods with a finite element discretization*, Electron. Trans. Numer. Anal., 45 (2016), pp. 219–240.

[74] P. Geldermans and A. Gillman, *An adaptive high order direct solution technique for elliptic boundary value problems*, SIAM J. Sci. Comput., 41 (2019), pp. A292–A315, https://doi.org/10.1137/17M1156320.

[75] C. Geuzaine and J.-F. Remacle, *Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities*, Int. J. Numer. Methods Eng., 79 (2009), pp. 1309–1331, https://doi.org/10.1002/nme.2579.

[76] A. Gholami, D. Malhotra, H. Sundar, and G. Biros, *FFT, FMM, or multigrid? A comparative study of state-of-the-art Poisson solvers for uniform and nonuniform grids in the unit cube*, SIAM J. Sci. Comput., 38 (2016), pp. C280–C306, https://doi.org/10.1137/15M1010798.

[77] M. A. Gilles and A. Townsend, *Continuous analogues of Krylov subspace methods for differential operators*, SIAM J. Numer. Anal., 57 (2019), pp. 899–924, https://doi.org/10.1137/18M1177810.

[78] A. Gillman, A. H. Barnett, and P.-G. Martinsson, *A spectrally accurate direct solution technique for frequency-domain scattering problems with variable media*, BIT Numer. Math., 55 (2015), pp. 141–170, https://doi.org/10.1007/s10543-014-0499-8.

[79] A. Gillman and P.-G. Martinsson, *A direct solver with $O(N)$ complexity for variable coefficient elliptic PDEs discretized via a high-order composite spectral collocation method*, SIAM J. Sci. Comput., 36 (2014), pp. A2023–A2046, https://doi.org/10.1137/130918988.

[80] A. Gillman and P.-G. Martinsson, *An $O(N)$ algorithm for constructing the solution operator to 2D elliptic boundary value problems in the absence of body loads*, Adv. Comput. Math., 40 (2014), pp. 773–796, https://doi.org/10.1007/s10444-013-9326-z.

[81] A. Gopal and L. N. Trefethen, *Solving Laplace problems with corner singularities via rational functions*, SIAM J. Numer. Anal., 57 (2019), pp. 2074–2094, https://doi.org/10.1137/19M125947X.

[82] J. Gopalakrishnan and G. Kanschat, *A multilevel discontinuous Galerkin method*, Numer. Math., 95 (2003), pp. 527–550, https://doi.org/10.1007/s002110200392.

[83] D. Gottlieb and S. A. Orszag, *Numerical Analysis of Spectral Methods*, SIAM, Philadelphia, 1977, https://doi.org/10.1137/1.9781611970425.

[84] S. Govindjee and P.-O. Persson, *A time-domain discontinuous Galerkin method for mechanical resonator quality factor computations*, J. Comput. Phys., 231 (2012), pp. 6380–6392, https://doi.org/10.1016/j.jcp.2012.05.034.

[85] L. Greengard and J. Lee, *A direct adaptive Poisson solver of arbitrary order accuracy*, J. Comput. Phys., 125 (1996), pp. 415–424, https://doi.org/10.1006/jcph.1996.0103.

[86] L. Greengard and V. Rokhlin, *A fast algorithm for particle simulations*, J. Comput. Phys., 73 (1987), pp. 325–348, https://doi.org/10.1016/0021-9991(87)90140-9.

[87] D. B. Haidvogel and T. Zang, *The accurate solution of Poisson's equation by expansion in Chebyshev polynomials*, J. Comput. Phys., 30 (1979), pp. 167–180, https://doi.org/10.1016/0021-9991(79)90097-4.

[88] S. Hao and P.-G. Martinsson, *A direct solver for elliptic PDEs in three dimensions based on hierarchical merging of Poincaré–Steklov operators*, J. Comput. Appl. Math., 308 (2016), pp. 419–434, https://doi.org/10.1016/j.cam.2016.05.013.

[89] B. T. Helenbrook, D. Mavriplis, and H. L. Atkins, *Analysis of p-multigrid for continuous and discontinuous finite element discretizations*, in 16th AIAA Computational Fluid Dynamics Conference, 2003, https://doi.org/10.2514/6.2003-3989.

[90] P. Henrici, *Fast Fourier methods in computational complex analysis*, SIAM Review, 21 (1979), pp. 481–527, https://doi.org/10.1137/1021093.

[91] J. S. Hesthaven and T. Warburton, *Nodal Discontinuous Galerkin Methods: Algorithms, Analysis, and Applications*, vol. 54 of Texts in Applied Mathematics, Springer, New York, 2008, https://doi.org/10.1007/978-0-387-72067-8.

[92] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, SIAM, Philadelphia, 2nd ed., 2002.

[93] T. J. Hughes, *The finite element method: linear static and dynamic finite element analysis*, Courier Corporation, 2012.

[94] S. S. Jain, K. Kamrin, and A. Mani, *A conservative and non-dissipative Eulerian formulation for the simulation of soft solids in fluids*, J. Comput. Phys., 399 (2019), https://doi.org/10.1016/j.jcp.2019.108922.

[95] K. Kamrin and J.-C. Nave, *An Eulerian approach to the simulation of deformable solids: Application to finite-strain elasticity*, Jan. 2009, https://arxiv.org/abs/0901.3799.

[96] K. Kamrin, C. H. Rycroft, and J.-C. Nave, *Reference map technique for finite-strain elasticity and fluid–solid interaction*, J. Mech. Phys. Solids, 60 (2012), pp. 1952–1969, https://doi.org/10.1016/j.jmps.2012.06.003.

[97] G. Kanschat, *Preconditioning methods for local discontinuous Galerkin discretizations*, SIAM J. Sci. Comput., 25 (2003), pp. 815–831, https://doi.org/10.1137/S1064827502410657.

[98] G. Kanschat, *Multilevel methods for discontinuous Galerkin FEM on locally refined meshes*, Computers & Structures, 82 (2004), pp. 2437–2445, https://doi.org/10.1016/j.compstruc.2004.04.015.

[99] G. Karypis and V. Kumar, *A fast and high quality multilevel scheme for partitioning irregular graphs*, SIAM J. Sci. Comput., 20 (1998), pp. 359–392, https://doi.org/10.1137/S1064827595287997.

[100] T. Koornwinder, *Two-variable analogues of the classical orthogonal polynomials*, in Theory and Application of Special Functions, R. A. Askey, ed., Academic Press, 1975, pp. 435–495, https://doi.org/10.1016/B978-0-12-064850-4.50015-X.

[101] L. I. G. Kovasznay, *Laminar flow behind a two-dimensional grid*, Math. Proc. Camb. Philos. Soc., 44 (1948), pp. 58–62, https://doi.org/10.1017/S0305004100023999.

[102] C. Lanczos, *Trigonometric interpolation of empirical and analytical functions*, J. Math. Phys., 17 (1938), pp. 123–199, https://doi.org/10.1002/sapm1938171123.

[103] V. I. Lebedev, *On a Zolotarev problem in the method of alternating directions*, USSR Comput. Math. Math. Phys., 17 (1977), pp. 58–76, https://doi.org/10.1016/0041-5553(77)90036-2.

[104] R. J. LeVeque, *Finite Difference Methods for Ordinary and Partial Differential Equations*, SIAM, Philadelpha, PA, 2007, https://doi.org/10.1137/1.9780898717839.

[105] R. J. LeVeque, *Top ten reasons to not share your code (and why you should anyway)*, SIAM News, 46 (2013), https://sinews.siam.org/Details-Page/top-ten-reasons-to-not-share-your-code-and-why-you-should-anyway.

[106] D. I. W. Levin, J. Litven, G. L. Jones, S. Sueda, and D. K. Pai, *Eulerian solid simulation with contact*, ACM Trans. Graph., 30 (2011), https://doi.org/10.1145/2010324.1964931.

[107] H. Liu, N. Mitchell, M. Aanjaneya, and E. Sifakis, *A scalable Schur-complement fluids solver for heterogeneous compute platforms*, ACM Trans. Graph., 35 (2016), https://doi.org/10.1145/2980179.2982430.

[108] A. Lu and E. L. Wachspress, *Solution of Lyapunov equations by alternating direction implicit iteration*, Comput. Math. Appl., 21 (1991), pp. 43–58, https://doi.org/10.1016/0898-1221(91)90124-M.

[109] H. Luo, J. D. Baum, and R. Löhner, *A p-multigrid discontinuous Galerkin method for the Euler equations on unstructured grids*, J. Comput. Phys., 211 (2006), pp. 767–783, https://doi.org/10.1016/j.jcp.2005.06.019.

[110] T. Mach and J. Saak, *How competitive is the ADI for tensor structured equations?*, PAMM, 12 (2012), pp. 635–636, https://doi.org/10.1002/pamm.201210306.

[111] Y. Maday and R. Muñoz, *Spectral element multigrid. II. Theoretical justification*, J. Sci. Comput., 3 (1988), pp. 323–353, https://doi.org/10.1007/BF01065177.

[112] P. Martinsson, *A direct solver for variable coefficient elliptic PDEs discretized via a composite spectral collocation method*, J. Comput. Phys., 242 (2013), pp. 460–479, https://doi.org/10.1016/j.jcp.2013.02.019.

[113] P.-G. Martinsson, *A fast direct solver for a class of elliptic partial differential equations*, J. Sci. Comput., 38 (2009), pp. 316–330, https://doi.org/10.1007/s10915-008-9240-6.

[114] P.-G. Martinsson, *Fast Direct Solvers for Elliptic PDEs*, SIAM, Philadelphia, 2019, https://doi.org/10.1137/1.9781611976045.

[115] P. G. Martinsson, V. Rokhlin, and M. Tygert, *A fast algorithm for the inversion of general Toeplitz matrices*, Comput. Math. Appl., 50 (2005), pp. 741–751, https://doi.org/10.1016/j.camwa.2005.03.011.

[116] A. McKenney, L. Greengard, and A. Mayo, *A fast Poisson solver for complex geometries*, J. Comput. Phys., 118 (1995), pp. 348–355, https://doi.org/10.1006/jcph.1995.1104.

[117] P. E. Merilees, *The pseudospectral approximation applied to the shallow water equations on a sphere*, Atmosphere, 11 (1973), pp. 13–20, https://doi.org/10.1080/00046973.1973.9648342.

[118] W. F. Mitchell, *A collection of 2D elliptic problems for testing adaptive grid refinement algorithms*, Appl. Math. Comput., 220 (2013), pp. 350–364, https://doi.org/10.1016/j.amc.2013.05.068.

[119] W. F. Mitchell and M. A. McClain, *A survey of hp-adaptive strategies for elliptic partial differential equations*, in Recent Advances in Computational and Applied Mathematics, T. E. Simos, ed., Dordrecht, 2011, Springer, pp. 227–258, https://doi.org/10.1007/978-90-481-9981-5_10.

[120] G. E. Moore, *Cramming more components onto integrated circuits*, Electronics, 38 (1965), pp. 114–117.

[121] D. Moxey, C. D. Cantwell, Y. Bao, A. Cassinelli, G. Castiglioni, S. Chun, E. Juda, E. Kazemi, K. Lackhove, J. Marcon, G. Mengaldo, D. Serson, M. Turner, H. Xu, J. Peiró, R. M. Kirby, and S. J. Sherwin, *Nektar++: Enhancing the capability and application of high-fidelity spectral/hp element methods*, Comput. Phys. Commun., 249 (2020), https://doi.org/10.1016/j.cpc.2019.107110.

[122] L. N. Olson and J. B. Schroder, *Smoothed aggregation multigrid solvers for high-order discontinuous Galerkin methods for elliptic problems*, J. Comput. Phys., 230 (2011), pp. 6959–6976, https://doi.org/10.1016/j.jcp.2011.05.009.

[123] F. W. J. Olver, D. W. Lozier, R. F. Boisvert, and C. W. Clark, *NIST Handbook of Mathematical Functions*, Cambridge University Press, New York, 2010.

[124] S. Olver. GitHub repository, https://github.com/JuliaApproximation/ApproxFun.jl.

[125] S. Olver and A. Townsend, *A fast and well-conditioned spectral method*, SIAM Rev., 55 (2013), pp. 462–489, https://doi.org/10.1137/120865458.

[126] S. Olver and A. Townsend, *A practical framework for infinite-dimensional linear algebra*, in Proceedings of the 1st First Workshop for High Performance Technical Computing in Dynamic Languages, Piscataway, NJ, USA, 2014, IEEE Press, pp. 57–69, https://doi.org/10.1109/HPTCDL.2014.10.

[127] S. Olver, A. Townsend, and G. Vasil, *A sparse spectral method on triangles*, SIAM J. Sci. Comput., 41 (2019), pp. A3728–A3756, https://doi.org/10.1137/19M1245888.

[128] S. A. Orszag, *Accurate solution of the Orr–Sommerfeld stability equation*, J. Fluid Mech., 50 (1971), pp. 689–703, https://doi.org/10.1017/S0022112071002842.

[129] S. A. Orszag, *Spectral methods for problems in complex geometries*, J. Comput. Phys., 37 (1980), pp. 70–92, https://doi.org/10.1016/0021-9991(80)90005-4.

[130] E. L. Ortiz, *The tau method*, SIAM J. Numer. Anal., 6 (1969), pp. 480–492, https://doi.org/10.1137/0706044.

[131] E. L. Ortiz and H. Samara, *An operational approach to the Tau method for the numerical solution of non-linear differential equations*, Computing, 27 (1981), pp. 15–25, https://doi.org/10.1007/BF02243435.

[132] I. Oseledets, *Tensor-train decomposition*, SIAM J. Sci. Comput., 33 (2011), pp. 2295–2317, https://doi.org/10.1137/090752286.

[133] I. Oseledets and S. Dolgov, *Solution of linear systems and matrix inversion in the TT-format*, SIAM J. Sci. Comput., 34 (2012), pp. A2718–A2739, https://doi.org/10.1137/110833142.

[134] A. T. Patera, *A spectral element method for fluid dynamics: Laminar flow in a channel expansion*, J. Comput. Phys., 54 (1984), pp. 468–488, https://doi.org/10.1016/0021-9991(84)90128-1.

[135] W. Pazner, *Efficient low-order refined preconditioners for high-order matrix-free continuous and discontinuous Galerkin methods*, Aug 2019, https://arxiv.org/abs/1908.07071.

[136] D. W. Peaceman and J. H. H. Rachford, *The numerical solution of parabolic and elliptic differential equations*, J. Soc. Ind. Appl. Math., 3 (1955), pp. 28–41, https://doi.org/10.1137/0103003.

[137] J. Peraire and P.-O. Persson, *The compact discontinuous Galerkin (CDG) method for elliptic problems*, SIAM J. Sci. Comput., 30 (2008), pp. 1806–1824, https://doi.org/10.1137/070685518.

[138] P.-O. Persson, *A sparse and high-order accurate line-based discontinuous Galerkin method for unstructured meshes*, J. Comput. Phys., 233 (2013), pp. 414–429, https://doi.org/10.1016/j.jcp.2012.09.008.

[139] I. Perugia and D. Schötzau, *An hp-analysis of the local discontinuous Galerkin method for diffusion problems*, J. Sci. Comput., 17 (2002), pp. 561–571, https://doi.org/10.1023/A:1015118613130.

[140] R. B. Platte, L. N. Trefethen, and A. B. Kuijlaars, *Impossibility of fast stable approximation of analytic functions from equispaced samples*, SIAM Review, 53 (2011), pp. 308–318, https://doi.org/10.1137/090774707.

[141] F. Prill, M. Lukáčová-Medviďová, and R. Hartmann, *Smoothed aggregation multigrid for the discontinuous Galerkin method*, SIAM J. Sci. Comput., 31 (2009), pp. 3503–3528, https://doi.org/10.1137/080728457.

[142] F. Rathgeber, D. A. Ham, L. Mitchell, M. Lange, F. Luporini, A. T. T. Mcrae, G.-T. Bercea, G. R. Markall, and P. H. J. Kelly, *Firedrake: Automating the finite element method by composing abstractions*, ACM Trans. Math. Soft., 43 (2016), https://doi.org/10.1145/2998441.

[143] W. H. Reed and T. R. Hill, *Triangular mesh methods for the neutron transport equation*, Tech. Report LA-UR-73-479, Los Alamos Scientific Laboratory, 1973.

[144] E. M. Rønquist and A. T. Patera, *Spectral element multigrid. I. Formulation and numerical results*, J. Sci. Comput., 2 (1987), pp. 389–406, https://doi.org/10.1007/BF01061297.

[145] U. Rüde, K. Willcox, L. C. McInnes, and H. De Sterck, *Research and education in computational science and engineering*, SIAM Rev., 60 (2018), pp. 707–754, https://doi.org/10.1137/16M1096840.

[146] J. Rudi, A. C. I. Malossi, T. Isaac, G. Stadler, M. Gurnis, P. W. J. Staar, Y. Ineichen, C. Bekas, A. Curioni, and O. Ghattas, *An extreme-scale implicit solver for complex PDEs: Highly heterogeneous flow in earth's mantle*, in Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '15, New York, 2015, ACM, https://doi.org/10.1145/2807591.2807675.

[147] S. J. Ruuth, *Implicit-explicit methods for reaction-diffusion problems in pattern formation*, J. Math. Biol., 34 (1995), pp. 148–176, https://doi.org/10.1007/BF00178771.

[148] C. H. Rycroft, C.-H. Wu, Y. Yu, and K. Kamrin, *Reference map technique for incompressible fluid–structure interaction*, J. Fluid Mech., 898 (2020), A9, https://doi.org/10.1017/jfm.2020.353.

[149] J. Sabino, *Solution of large-scale Lyapunov equations via the block modified Smith method*, PhD thesis, Rice University, 2007, https://hdl.handle.net/1911/20641.

[150] R. I. Saye, *High-order quadrature methods for implicitly defined surfaces and volumes in hyperrectangles*, SIAM J. Sci. Comput., 37 (2015), pp. A993–A1019, https://doi.org/10.1137/140966290.

[151] R. I. Saye, *Implicit mesh discontinuous Galerkin methods and interfacial gauge methods for high-order accurate interface dynamics, with applications to surface tension dynamics, rigid body fluid–structure interaction, and free surface flow: Part I*, J. Comput. Phys., 344 (2017), pp. 647–682, https://doi.org/10.1016/j.jcp.2017.04.076.

[152] R. I. Saye, *Implicit mesh discontinuous Galerkin methods and interfacial gauge methods for high-order accurate interface dynamics, with applications to surface tension dynamics, rigid body fluid–structure interaction, and free surface flow: Part II*, J. Comput. Phys., 344 (2017), pp. 683–723, https://doi.org/10.1016/j.jcp.2017.05.003.

[153] R. I. Saye, *Efficient multigrid solution of elliptic interface problems using viscosity-upwinded local discontinuous Galerkin methods*, Comm. Appl. Math. Comput. Sci., 14 (2019), pp. 247–283, https://doi.org/10.2140/camcos.2019.14.247.

[154] S. Sherwin and G. Karniadakis, *Spectral/hp element methods for computational fluid dynamics*, Oxford University Press, Oxford, 2005, https://doi.org/10.1093/acprof:oso/9780198528692.001.0001.

[155] S. J. Sherwin and G. E. Karniadakis, *A new triangular and tetrahedral basis for high-order (hp) finite element methods*, Int. J. Numer. Methods Eng., 38 (1995), pp. 3775–3802, https://doi.org/10.1002/nme.1620382204.

[156] C. Siefert, R. Tuminaro, A. Gerstenberger, G. Scovazzi, and S. S. Collis, *Algebraic multigrid techniques for discontinuous Galerkin methods with varying polynomial order*, Comput. Geosci., 18 (2014), pp. 597–612, https://doi.org/10.1007/s10596-014-9419-x.

[157] J. P. Slotnick, A. Khodadoust, J. J. Alonso, D. L. Darmofal, W. D. Gropp, E. A. Lurie, and D. J. Mavriplis, *CFD vision 2030 study: A path to revolutionary computational aerosciences*, Tech. Report NASA/CR-2014-218178, NASA Langley Research Center, 2014.

[158] J. Stiller, *Robust multigrid for high-order discontinuous Galerkin methods: A fast Poisson solver suitable for high-aspect ratio Cartesian grids*, J. Comput. Phys., 327 (2016), pp. 317–336, https://doi.org/10.1016/j.jcp.2016.09.041.

[159] M. Sussman, A. S. Almgren, J. B. Bell, P. Colella, L. H. Howell, and M. L. Welcome, *An adaptive level set approach for incompressible two-phase flows*, J. Comput. Phys., 148 (1999), pp. 81–124, https://doi.org/10.1006/jcph.1998.6106.

[160] O. Tatebe, *The multigrid preconditioned conjugate gradient method*, in The Sixth Copper Mountain Conference on Multigrid Methods, NASA Langley Research Center, 1993, pp. 621–634.

[161] Y. Teng, D. I. W. Levin, and T. Kim, *Eulerian solid-fluid coupling*, ACM Trans. Graph., 35 (2016), https://doi.org/10.1145/2980179.2980229.

[162] D. J. Torres and E. A. Coutsias, *Pseudospectral solution of the two-dimensional Navier–Stokes equations in a disk*, SIAM J. Sci. Comput., 21 (1999), pp. 378–403, https://doi.org/10.1137/S1064827597330157.

[163] A. Townsend and S. Olver, *The automatic solution of partial differential equations using a global spectral method*, J. Comput. Phys., 299 (2015), pp. 106–123, https://doi.org/10.1016/j.jcp.2015.06.031.

[164] A. Townsend and L. N. Trefethen, *An extension of Chebfun to two dimensions*, SIAM J. Sci. Comput., 35 (2013), pp. C495–C518, https://doi.org/10.1137/130908002.

[165] A. Townsend, M. Webb, and S. Olver, *Fast polynomial transforms based on Toeplitz and Hankel matrices*, Math. Comput., 87 (2018), pp. 1913–1934, https://doi.org/10.1090/mcom/3277.

[166] A. Townsend and H. Wilber, *On the singular values of matrices with high displacement rank*, Lin. Alg. Appl., 548 (2018), pp. 19–41, https://doi.org/10.1016/j.laa.2018.02.025.

[167] A. Townsend, H. Wilber, and G. B. Wright, *Computing with functions in spherical and polar geometries I. The sphere*, SIAM J. Sci. Comput., 38 (2016), pp. C403–C425, https://doi.org/10.1137/15M1045855.

[168] L. N. Trefethen, *Spectral Methods in MATLAB*, SIAM, Philadelphia, 2000, https://doi.org/10.1137/1.9780898719598.

[169] L. N. Trefethen, *Approximation Theory and Approximation Practice*, SIAM, Philadelphia, 2012.

[170] B. Valkov, C. H. Rycroft, and K. Kamrin, *Eulerian method for multiphase interactions of soft solid bodies in fluids*, J. Appl. Mech., 82 (2015), https://doi.org/10.1115/1.4029765.

[171] P. E. Vos, S. J. Sherwin, and R. M. Kirby, *From h to p efficiently: Implementing finite and spectral/hp element methods to achieve optimal performance for low- and high-order discretisations*, J. Comput. Phys., 229 (2010), pp. 5161–5181, https://doi.org/10.1016/j.jcp.2010.03.031.

[172] E. Wachspress, *Three-variable alternating-direction-implicit iteration*, Comput. Math. with Appl., 27 (1994), pp. 1–7, https://doi.org/10.1016/0898-1221(94)90040-X.

[173] E. Wachspress, *The ADI Model Problem*, Springer, New York, 2013, https://doi.org/10.1007/978-1-4614-5122-8.

[174] J. A. C. Weideman and L. N. Trefethen, *The eigenvalues of second-order spectral differentiation matrices*, SIAM J. Numer. Anal., 25 (1988), pp. 1279–1298, https://doi.org/10.1137/0725072.

[175] O. B. Widlund, *On the rate of convergence of an alternating direction implicit method in a noncommutative case*, Math. Comput., 20 (1966), pp. 500–515, https://doi.org/10.2307/2003540.

[176] H. Wilber, *Numerical computing with functions on the sphere and disk*, master's thesis, Boise State University, 2016, https://scholarworks.boisestate.edu/td/1158.

[177] H. Wilber, A. Townsend, and G. B. Wright, *Computing with functions in spherical and polar geometries II. The disk*, SIAM J. Sci. Comput., 39 (2017), pp. C238–C262, https://doi.org/10.1137/16M1070207.

[178] J. Xu, *Iterative methods by space decomposition and subspace correction*, SIAM Review, 34 (1992), pp. 581–613, https://doi.org/10.1137/1034116.

[179] J. Yan and C.-W. Shu, *Local discontinuous Galerkin methods for partial differential equations with higher order derivatives*, J. Sci. Comput., 17 (2002), pp. 27–47, https://doi.org/10.1023/A:1015132126817.

[180] A. Yeiser and A. Townsend, *A spectral element method for meshes with skinny elements*, SIAM Undergraduate Research Online, (2018), pp. 421–437, https://doi.org/10.1137/18S017053.

[181] F. Zernike, *Beugungstheorie des schneidenver-fahrens und seiner verbesserten form, der phasenkontrastmethode*, Physica, 1 (1934), pp. 689–704, https://doi.org/10.1016/S0031-8914(34)80259-5.

[182] Y. Zhang and A. Gillman, *A fast direct solver for boundary value problems on locally perturbed geometries*, J. Comput. Phys., 356 (2018), pp. 356–371, https://doi.org/10.1016/j.jcp.2017.12.013.

[183] E. I. Zolotarev, *Application of elliptic functions to questions of functions deviating least and most from zero*, Zap. Imp. Akad. Nauk. St. Petersburg, 30 (1877), pp. 1–59.