OVERVIEW OF SOLUTION METHODS FOR ELLIPTIC PARTIAL DIFFERENTIAL EQUATIONS ON CARTESIAN AND HIERARCHICAL GRIDS

DAMYN M. CHIPMAN

Abstract.
Elliptic partial differential equations (PDEs) arise in many areas of computational sciences such as computational fluid dynamics, biophysics, engineering, geophysics and more. They are difficult to solve due to their global nature and sometimes ill-conditioned operators. We review common discretization methods for elliptic PDEs such as the finite difference, finite volume, finite element, and spectral methods and the linear systems they form. We also provide an overview of classic to modern solution methods for the linear systems formed by these discretization methods. These methods include splitting and Krylov methods, direct methods, and hierarchical methods. Finally, we show applications that would benefit from fast and efficient solvers for elliptic PDEs, including projection methods for the incompressible Navier-Stokes equations and the shallow water wave equations with dispersive corrections.

1. Introduction.

Many physical systems can be described using differential equations. Examples include Newton's Laws of Motion, almost all conservation laws (mass, momentum, energy, etc.), many engineering applications, and many more. To simulate or model such systems, we often use numerical techniques to discretize and solve the corresponding problem on computers. As computational resources are finite, researchers must employ efficient algorithms to solve these problems.

Elliptic partial differential equations are a class of PDEs that explain global and steady state phenomena. Elliptic PDEs arise in fluid dynamics, heat transfer, electromagnetism, geophysics, biology, and other application areas. Many elliptic problems can be solved efficiently, but problems on complex geometries or complicated meshes are more challenging to solve. Solution methods for elliptic PDEs is a well studied topic, with many papers, books, and courses detailing their methods. However, fast and efficient ways to solve elliptic PDEs is an ongoing field of research. Improvements on classical methods, as well as new ideas, have been recently introduced.

1.1. Outline.

We will first address Laplace's and Poisson's equation in Section 2 as well as numerical approaches to solving these problems. In Section 3, we will look at various solution methods for solving the linear system formed by the discretization methods discussed in Section 2. Next in Section 4, we look at how elliptic solvers can be used on unstructured and adaptive meshes. Finally in Section 5, we give a brief overview of the pressure Poisson equation needed to maintain divergence free velocity fields and a shallow water equation model with dispersive corrections.

2. The Elliptic Partial Differential Equation.

Partial differential equations are classified by their highest order derivative terms. A second order, linear differential equation can be written as

$$A(x,y)\frac{\partial^2 u(x,y)}{\partial x^2} + B(x,y)\frac{\partial^2 u(x,y)}{\partial x \partial y} + C(x,y)\frac{\partial^2 u(x,y)}{\partial y^2} + ...$$
$$... + D(x,y)\frac{\partial u(x,y)}{\partial x} + E(x,y)\frac{\partial u(x,y)}{\partial y} + F(x,y)u(x,y) + G(x,y) = 0.$$

Second-order linear PDEs are classified according to the value of the determinant of this expression:

$$\begin{aligned} B^2 - 4AC &< 0, \quad \text{Elliptic} \\ B^2 - 4AC &= 0, \quad \text{Parabolic} \\ B^2 - 4AC &> 0, \quad \text{Hyperbolic} \end{aligned}$$

In this overview, we will consider common elliptic PDEs such as Laplace's Equation

$$\nabla^2 u(x,y) = 0, \tag{2.1}$$

Poisson's Equation

$$\nabla^2 u(x,y) = f(x,y), \tag{2.2}$$

and the variable coefficient Poisson's Equation

$$\nabla \cdot \Big(\beta(x,y)\nabla u(x,y)\Big) = f(x,y), \tag{2.3}$$

where $\nabla = (\frac{\partial}{\partial x}, \frac{\partial}{\partial y})$, $\nabla^2 = \nabla \cdot \nabla = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$, $x, y \in \Omega$, and each with the appropriate boundary conditions on the domain boundary $\partial\Omega = \Gamma$. Such boundary conditions (BCs) can either be Dirichlet (Type-I), Neumann (Type-II), or Robin/Mixed (Type-III) BCs. Dirichlet problems impose the value of u on the boundaries, Neumann problems impose the flux or normal gradient $\partial_n u$ on the boundaries, while Robin problems impose a linear combination of Dirichlet or Neumann type BCs.

Although analytical solutions exist for some (simple) variations of the problems above, we are interested in looking at numerical methods to solving these equations. To use such numerical methods, we first look at various ways to discretize the domain of elliptic PDEs. Common ways to numerically solve elliptic PDEs start with a discrete representation of the domain. We will look at node based, cell based, element based, and spectral based approaches. These approaches are called the finite difference method, finite volume method, finite element method, and the spectral method, respectively. We will refer to Figure 2.1 for a diagram of each.

2.1. Finite Difference. If Ω is on a logically rectangular domain (i.e. 2D Cartesian plane), perhaps the simplest discretization of the domain Ω is by collocating a mesh of points throughout the domain. Given upper and lower bounds in both directions, $[x_l, x_u], [y_l, y_u]$, the x- and y-location of each point can be defined as

$$x_i = x_l + i\Delta x, \quad i = 0, 1, ..., N_x - 1 \tag{2.4}$$

$$y_i = y_l + j\Delta y, \quad j = 0, 1, ..., N_y - 1 \tag{2.5}$$

where N_x, N_y is the number of points in the x- and y-direction and grid spacing is defined as

$$\Delta x = \frac{x_u - x_l}{N_x - 1} \quad \Delta y = \frac{y_u - y_l}{N_y - 1}. \tag{2.6}$$

These points are shown in the first plot of Figure 2.1. This leads to the finite difference method, where the function to approximate is solved for at each mesh point. In the finite difference approach, the expressions for the derivatives in a PDE are replaced

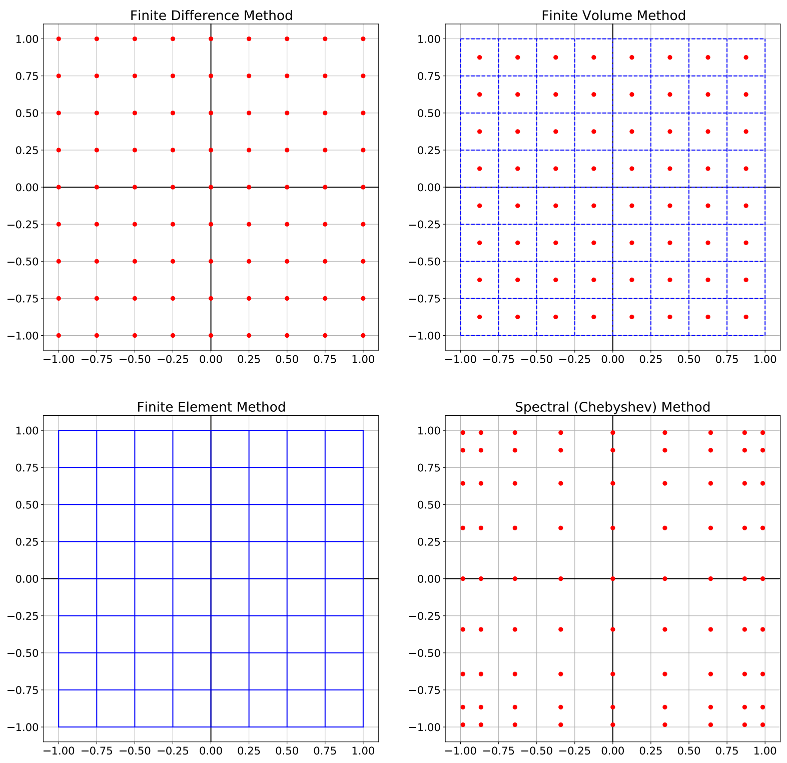

FIG. 2.1. *Discretization Methods. Top Left: Finite difference grid based on a mesh of points or nodes. Top Right: Finite volume mesh made of cells (dashed blue boxes) with cell averages in the center (red points). Bottom Left: Finite element mesh where each blue box is an element with a shape function defined at all points within the element. Bottom Right: Spectral mesh made up of a tensor of Chebyshev nodes.*

with Taylor Series approximations. For example, a second order accurate, central-difference approximation to the second derivative can be given as

$$\frac{\partial^2 u}{\partial x^2} = \frac{u(x-\Delta x, y) - 2u(x,y) + u(x+\Delta x, y)}{\Delta x^2} + \mathcal{O}(\Delta x^2) \tag{2.7}$$

and if we let $u(x_i, y_j) = u_{i,j}$, then we can write this as

$$\frac{\partial^2 u}{\partial x^2} = \frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{\Delta x^2} + \mathcal{O}(\Delta x^2). \tag{2.8}$$

Using these Taylor Series expansions, we can replace the continuous Poisson's equation 2.2 with the discrete system of equations:

$$\frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{\Delta x^2} + \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{\Delta y^2} = f_{i,j}. \tag{2.9}$$

For each grid point $i, j = 0, ..., N-1$, this expression forms a linear system with $(N_x - 1) \times (N_y - 1)$ unknowns.

2.2. Finite Volume. In a finite volume scheme, the domain is broken up into cells, which can be structured or unstructured polygons (2D) or polyhedrons (3D).

The function to approximate is solved for in terms of a cell average. Finite volume schemes are often used to solve equations modeling conservation laws, where the cell quantity is conserved in time through balancing fluxes (what comes in and out of the cell boundaries) and the cell source (what the cell is generating or destroying). In a finite volume scheme, the cell average is updated according to the integral formulation of the conservation law as

$$\frac{\partial}{\partial t}\int_{\Omega_i} \mathbf{q}\, d\Omega_i + \int_{\Gamma_i} \mathbf{F}(\mathbf{q}) \cdot \hat{n}_i d\Gamma_i + \int_{\Omega_i} \mathbf{s}\, d\Omega_i = 0, \tag{2.10}$$

where Ω_i and Γ_i are the cell domains and boundaries, respectively, $\mathbf{q}$ is a vector of quantities in each cell, $\mathbf{F}$ is a flux function, and $\mathbf{s}$ is the cell's source term. In the second plot of Figure 2.1, the blue dashed lines are the cell boundaries, and the red points are the cell centers. For up to second order schemes, the cell average is collocated at the center of the cell; this changes for higher order schemes.

2.3. Finite Element. In a finite element scheme, the domain is broken into elements. These elements can also be structured or unstructured polygons (2D) or polyhedrons (3D) as in the finite volume method. The PDE to solve is converted into a weak form by multiplying the PDE by a test function $v(x, y)$ and integrating over the domain. For example, converting Poisson's equation into the weak form looks like this

$$\int_\Omega \frac{\partial^2 u}{\partial x^2} v d\Omega + \int_\Omega \frac{\partial^2 u}{\partial y^2} v d\Omega = \int_\Omega f v d\Omega \tag{2.11}$$

$$\Rightarrow \int_\Omega \frac{\partial u}{\partial x}\frac{\partial v}{\partial x} d\Omega + \int_\Omega \frac{\partial u}{\partial y}\frac{\partial v}{\partial y} d\Omega - \left[\frac{\partial u}{\partial x} v\right]\Big|_{x_l}^{x^u} - \left[\frac{\partial u}{\partial y} v\right]\Big|_{y_l}^{y^u} = \int_\Omega f v d\Omega, \tag{2.12}$$

where integration by parts is used to transfer a derivative to the test function. The idea behind the finite element method is to use shape or basis functions inside each element (defined over the entire element)

$$u(x, y) \approx \bar{u} = \sum_{k=1}^{N_k} c_k \phi(x, y) \tag{2.13}$$

and to use the same basis ϕ as the test function v (this is called the Galerkin principle). Upon substitution of the above into the weak form, and integrating the basis function with either quadrature or with analytical expressions, a linear system is formed for the coefficients c_k. Often, the coefficients are the actual function values $c_k = u_{i,j}$.

There are several variations of the finite element method. The method explained above forms a global linear system involving all elements. To avoid this, mapping a physical element to some reference element, makes the contributions from a single element only influence itself and its neighbors. This makes the coefficient matrix in the linear system much more sparse and structured, making it easier to solve. Additionally, by not imposing continuity across element boundaries, one arrives at the discontinuous Galerkin method, where the discontinuities are handled through element fluxes. By choosing certain shape or basis functions, one can derive different variations of the finite element method, including the B-spline finite element method [11] and the spectral element method [22].

2.4. Spectral Methods. In spectral methods, the approach is to approximate the solution u as a linear combination of a finite set of orthogonal basis functions

$$u(x,y) = \sum_{i=1}^{N_x} \sum_{j=1}^{N_y} c_{i,j} \phi_i(x) \phi_j(y) \tag{2.14}$$

where the coefficients are chosen to minimize a norm, such as the L^2 norm of the residual $r(x,y) = \nabla^2 u(x,y) - f(x,y)$. On a discrete domain, this is similar to requiring $\nabla^2 u(x_i, y_j) = f(x_i, y_j)$ at all interior grid points. As this acts like an interpolation scheme, increasing the number of discretization points on with a fixed interval actually leads to highly oscillatory results. Thus, in spectral methods, it is common to use grid points that are clustered near the ends of the interval, such as Chebyshev points, defined on an interval $[a,b]$ as

$$x_i = a + \frac{1}{2}(b-a)\Big(1 + \cos\big(\pi(1 - \frac{i}{N+1})\big)\Big), i = 0, ..., N+1. \tag{2.15}$$

These points are shown on the last plot of Figure 2.1. With a good basis of grid points, spectral methods can achieve very fast convergence. The linear system formed by this method will be dense as it functions like a high-order interpolation scheme. However, as convergence is much faster than high-order finite difference methods, one can use far fewer points on a grid, so the size of the system is kept small. If one uses Fourier series as the basis functions ϕ, one can accelerate the solution of the linear system using fast Fourier Transform algorithms. Though, due to the periodic nature of the Fourier series, it is more difficult to implement non-periodic boundary conditions ([14], [26]).

2.5. Other Methods and Summary. These methods are not all the possible ways to solve an elliptic PDE. Other methods include using quadrature methods on the integral formulation of the PDE, as well as collocation methods such as radial basis function approximations. However, the methods we reviewed show some of the most classic approaches to solving elliptic PDEs.

3. Solution Methods for Elliptic Partial Differential Equations.

The discretization methods described above will generally lead to a linear system of equations. The next step is to take advantage of the structure of these linear systems to solve the system efficiently. We will focus on finite difference and finite volume approaches. To generally talk about these solution methods, we assume that we form the linear system

$$\mathbf{A}\mathbf{u} = \mathbf{b} \tag{3.1}$$

where $\mathbf{A} \in \mathbb{R}^{N \times N}$ is a coefficient matrix from one of the discretization methods, $\mathbf{u} = u_i = u(x_i)$ for $i \in \mathbf{I}_x$ and index set $\mathbf{I}_x$ for the discrete domain, and $\mathbf{b}$ is a vector with the right hand side information f and encoded boundary information. The goal here is to solve for $\mathbf{u}$ where we take advantage of the structure of $\mathbf{A}$. We organize the various methods into the following three categories: 1) iterative methods, 2) direct methods, and 3) hierarchical methods.

3.1. Iterative Methods. Iterative methods start with an initial guess of a solution to 3.1 and correct the iterate until convergence to a specified tolerance. The simplest iterative methods are called splitting methods where the linear system is modified according to

$$\mathbf{A} = \mathbf{M} - \mathbf{N} \Rightarrow \mathbf{M}\mathbf{u} = \mathbf{N}\mathbf{u} + \mathbf{b}. \tag{3.2}$$

This suggests the following recursion relationship for the next iteration:

$$\mathbf{M}\mathbf{u}^{k+1} = \mathbf{N}\mathbf{u}^{k} + \mathbf{b} \tag{3.3}$$

$$\mathbf{u}^{k+1} = \mathbf{M}^{-1}\mathbf{N}\mathbf{u}^{k} + \mathbf{M}^{-1}\mathbf{b}. \tag{3.4}$$

The idea is to choose $\mathbf{M}$ that captures as much of $\mathbf{A}$ as possible, but is still easy and quick to invert. As $\mathbf{A}$ is either banded or sparse, classical splitting methods for elliptic PDEs split $\mathbf{A}$ into $\mathbf{A} = \mathbf{D} - \mathbf{L} - \mathbf{U}$, where $\mathbf{D}$ is the diagonal components of $\mathbf{A}$, and $\mathbf{L}$ and $\mathbf{U}$ are the lower and upper pieces, respectively. Classical choices for $\mathbf{M}$ and $\mathbf{N}$ are summarized in Table 3.1.

Jacobi	$\mathbf{M} = \mathbf{D}$	$\mathbf{N} = \mathbf{L} + \mathbf{U}$
Gauss-Sidel	$\mathbf{M} = \mathbf{D} - \mathbf{L}$	$\mathbf{N} = \mathbf{U}$
Successive Over Relaxation	$\mathbf{M} = \frac{1}{\omega}(\mathbf{D} - \omega\mathbf{L})$	$\mathbf{N} = \frac{1}{\omega}\big((1-\omega)\mathbf{D} + \omega\mathbf{U}\big)$

Table 3.1
Iterative Methods: Splitting Methods

Another class of iterative methods are called Krylov subspace methods. The Krylov space is defined as

$$\mathcal{K}_k = \text{span}\{\mathbf{r}_0, \mathbf{A}\mathbf{r}_0, \mathbf{A}^2\mathbf{r}_0, ..., \mathbf{A}^{k-1}\mathbf{r}_0\} \tag{3.5}$$

based on the initial residual $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{u}^{(0)}$. The goal is to take the next iteration from this particular space. Two common Krylov methods are the Conjugate Gradient method [9] and the Generalized Minimal Residual (GMRES) method [25]. In the conjugate gradient method, the approximation is adjusted by a conjugate direction, or a vector that is conjugate with respect to $\mathbf{A}$. This vector is called the search direction $\mathbf{p}$ and is scaled by α, which is computed by solving a quadratic minimization problem. The GMRES method builds up an orthogonal matrix $\mathbf{Q}$ through a process called the Arnoldi iteration. The Arnoldi iteration forms $\mathbf{A} = \mathbf{Q}\mathbf{H}\mathbf{Q}^*$ for orthogonal matrix $\mathbf{Q}$ and Hessenberg matrix $\mathbf{H}$. The next iteration is found via $\mathbf{Q}\mathbf{y}$ where $\mathbf{y}$ is found from a least squares problem involving $\mathbf{H}$. These methods are summarized in Table 3.2.

Conjugate Gradient	$\mathbf{u}^{(k+1)} = \mathbf{u}^{(k)} + \alpha^{(k)}\mathbf{p}^{(k)}$
GMRES	$\mathbf{u}^{(k)} = \mathbf{Q}^{(k)}\mathbf{y}$

Table 3.2
Iterative Methods: Krylov Subspace Methods

Most iterative methods are considered "matrix-free" methods. A "matrix-free" method is a method that does not explicitly form the matrix $\mathbf{A}$, but is rather applied to a vector. For example, if implementing a Conjugate Gradient method for a finite difference discretization, one has to compute the product $\mathbf{A}\mathbf{u}^{(k)}$. Instead of doing the full matrix-vector calculation, one can write a function that takes $\mathbf{u}$ and returns the 2nd-order, central difference operator as computed in 2.9.

As finite difference discretization schemes for elliptic PDEs lead to sparse matrices, the application of $\mathbf{A}$ to a vector can be done in $\mathcal{O}(N)$ operations, where N is the number of unknowns in the vector. Thus, the performance for most iterative methods is approximately $\mathcal{O}(N \times N_{iter})$, where N_{iter} is the number of iterations required for a specified tolerance. However, as N gets larger, often so does N_{iter}, leading to poor scaling, as noted in [19]. In addition, iterative methods may not always converge for a given initial guess or structure of $\mathbf{A}$, which make them unfavorable for "black-box" implementations for linear solvers.

3.2. Direct Methods. Motivation for direct solvers stems from wanting to improve upon the disadvantages of iterative methods. Martinsson notes in [21] some advantages to using direct methods over iterative ones:

- Direct methods can be applied to multiple right-hand side vectors $\mathbf{b}$ or multiple boundary conditions once a factorization or solution operator is built, whereas iterative methods must be solved anew for each right-hand side or for different boundary conditions.
- Direct methods can take advantage of "close" matrices (i.e. if we have an inverse or factorization of $\mathbf{A}$ and perturb it by ϵ, we could adjust the inverse to account for it instead of recompute the inverse).
- Most direct methods can take advantage of fast and efficient algorithms for matrix factorization such as the singular value decomposition, LU decomposition, QR decomposition, etc.

We will look at how direct methods are useful as we consider some common direct methods from [14] and [27].

Many direct methods are based on matrix factorizations. Perhaps the most well-known is the LU decomposition. LU decomposition factors the coefficient matrix into a lower and upper triangular matrix: $\mathbf{A} = \mathbf{LU}$. The idea is to use Gaussian elimination to eliminate entries below the main diagonal, and then use back-substitution to solve for each entry in $\mathbf{u}$. In general, LU decomposition requires $\mathcal{O}(N^3)$ floating point operations and thus is impractical for large matrices. There are banded solvers for Gaussian elimination that can take advantage of the sparsity of a matrix. The Cholesky decomposition is a variant of Gaussian elimination for symmetric matrices. Other matrix factorizations include the QR-decomposition, $\mathbf{A} = \mathbf{QR}$, and the singular value decomposition, $\mathbf{A} = \mathbf{U\Sigma V}^*$.

For a finite difference discretization of elliptic PDEs, $\mathbf{A}$ is banded and sparse, and more efficient algorithms for LU decomposition exist. For 1D problems, $\mathbf{A}$ is diagonally dominant and sparse with the bandwidth (the number of entries off the main diagonal in a matrix) dependent on the order of the stencil. For the stencil shown in the second order discretization in 2.9, $\mathbf{A}$ is tridiagonal. This allows us to use algorithms such as Thomas's algorithm for solving a tridiagonal system in $\mathcal{O}(N)$ steps ([10]). In higher dimensions, block versions of Thomas's algorithm exist ([24]).

Compared to iterative methods, direct methods will terminate in a finite number of steps. Direct methods are more fit for "black-box" implementations. However, because most direct methods need to explicitly form $\mathbf{A}$, they are more difficult to implement with limited computing resources. Indeed, going to higher dimensions and higher orders often dramatically increases memory and compute requirements.

3.3. Hierarchical Methods. The methods grouped under hierarchical methods attempt to accelerate some of the ideas from iterative and direct methods by breaking the problem into a hierarchy of subproblems. By recursively breaking the original problem into smaller subproblems, significant improvements can be made in

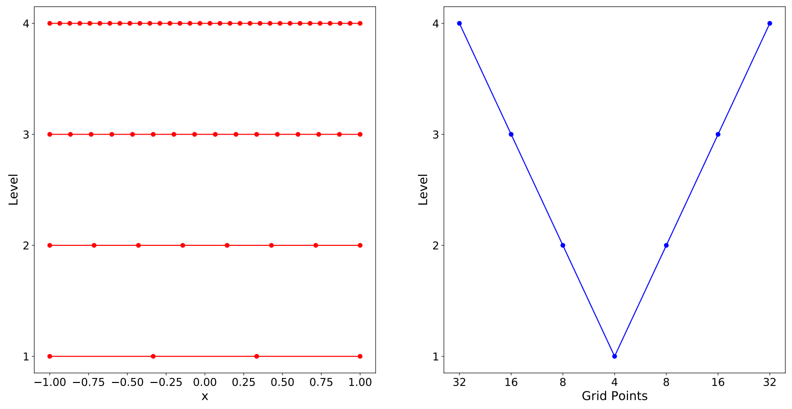

FIG. 3.1. *The 1D Multigrid Method. Starting with the finest grid, perform a few iterations of an iterative method. This is called relaxing the solution. Then project onto a coarser grid, and relax again. Do this down the levels in the grid to a desired precision. Once the solution is converged to on the coarsest level, interpolate back up the levels to obtain the solution on the finest level.*

complexity and performance. We'll talk about three hierarchical methods here: the multigrid method, nested dissection, and the Hierarchical Poincaré-Steklov (HPS) method.

3.3.1. The Multigrid Method. The multigrid method was introduced by Brandt in [2]. It has been widely used in various applications and for all types of solution methods. Briggs in [3] gives an overview and tutorial of the multigrid method.

In the multigrid method, the idea is to use multiple levels of grids and solution methods on each level to solve a larger problem. To look at the multigrid method, we define the error in the linear system to be $\mathbf{e} = \mathbf{u}^{(k)} - \mathbf{u}_{exact}$ (the difference between the exact solution and the kth iteration). After a few iterations of a smoother (typically Jacobi's method), because of the local averaging, any high-frequency error is quickly dampened away. What takes longer to eliminate is the low-frequency error associated with the global problem. By coarsening the grid, the lower frequency error is dampened quicker. Multigrid combines the ability of iterative methods to locally reduce error and a coarsening grid technique to accelerate convergence.

In the multigrid method, one starts with the solution on a fine grid, for example, with grid spacing h, and performs a few iterations of a smoother. After a few iterations, the h-level grid is projected onto a coarser $2h$-level grid. On this level, one performs a few more iterations of a smoother. Because the grid is coarser, this step is faster. Again, after a few iterations, the solution is projected onto an even coarser $4h$-level grid and a few more iterations are performed. This is done a specified number of times, and then the solution is interpolated back up the levels to the original grid. This is shown in Figure 3.1.

In what is called the "full multigrid" method, the process starts on the coarsest level instead of the finest. The solution is relaxed on this level, and then interpolated down onto a finer mesh. The interpolated solution is used as an initial guess for solving the problem on the finer mesh. The relaxed solution on a coarser grid is often an ideal initial guess for the problem on the finer mesh, resulting in quick convergence on that level.

The multigrid method allows one to accelerate an iterative solver. Multigrid methods are very effective as pre-conditioners for iterative methods. This is the general pattern of hierarchical methods: the ability to use classical iterative and direct methods on smaller grids where they perform well, and then "scale" them up to larger problem sizes.

3.3.2. Nested Dissection. The nested dissection method formulated by George in [7] is a direct method that builds upon Gaussian elimination for problems on a grid. It is also the basis for forming what is called the multifrontal method. By taking advantage of the ordering of points on a grid, one can permute $\mathbf{A}$ to first eliminate points that split the mesh into two unconnected meshes. This permutation takes the form of $\mathbf{P}^*\mathbf{A}\mathbf{P}$, where the goal is to form $\mathbf{P}$ to reorganize $\mathbf{A}$ in a way that eliminates points in an efficient manner.

To detail nested dissection better, consider an $N \times N$ mesh such as a finite difference mesh discussed in Section 2. Assume that the initial ordering of points corresponds to an index set that iterates over the points in the mesh row-by-row. Now, the idea of nested dissection is to reorganize the points such that we first eliminate points down the middle of the mesh (i.e. the points at $x = 0$ in the first plot of Figure 2.1). Once these points are solved for, it spilts the mesh into two equally sized pieces that are disconnected. Each of the disconnected meshes now are smaller, and thus easier to solve. This idea of splitting the mesh into disconnected pieces by first eliminating points along an interface can be recursively applied to each split. This means that after dividing the mesh into two, one can divide those two meshes into four meshes, and so on. Recursive splitting is a common characteristic in hierarchical methods.

Nested dissection was first introduced by George in [7], and further generalized by Lipton et al. in [15]. Martinsson has a tutorial on nested dissection in [19]. In fact, nested dissection served as motivation for another hierarchical method proposed by Martinsson and Gillman called the Hierarchical Poincaré-Steklov method.

3.3.3. The Hierarchical Poincaré-Steklov Method. The work done by Gillman and Martinsson in [21], [17], and [8] (with a practical tutorial found in [18]) culminate in what they call the Hierarchical Poincaré-Steklov (HPS) method. It is a direct solver for elliptic PDEs that is based on a binary tree of rectangular patches where the solution operator to $\mathbf{A}$ is built by recursively merging child patches. Like direct methods, the HPS method involves a factorization step and a solve step. The chief advantage of the HPS method over other direct methods is that it does not require the explicit formulation and storage of $\mathbf{A}$.

The HPS method starts with an original problem domain, and recursively divides the domain in half. This creates a binary tree of patches as shown in Figure 3.3. Once the domain has been decomposed into this tree of patches, two operators are defined on the lowest level, called the leaf level. These operators are the solution operator $\mathbf{S}$ and Dirichlet-to-Neumann (DtN) operator $\mathbf{T}$. The solution operator maps boundary data to solution data on the interior of the patch (i.e. solves the local boundary value problem), and the DtN operator maps Dirichlet data on the boundary to Neumann data on the boundary. These operators can be formed using any elliptic PDE solver, including fast solvers like spectral methods. After forming these operators, the next step is to recursively merge each sibling patch up the tree. the merge step is demonstrated in Figure 3.2. This results in a global solution operator that can be stored and used multiple times (at different time steps or with varying boundary conditions, etc.), and is similar to a direct method matrix factorization. The final step is applying the solution operator to each level down the tree to obtain the solution everywhere in the domain. This step is just a matrix-vector multiplication and is very fast.

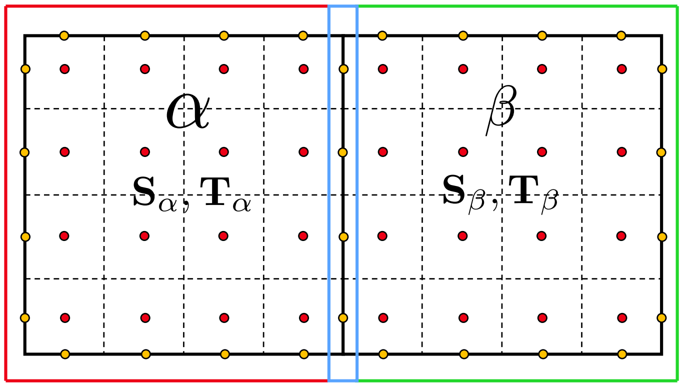
α
$\mathbf{S}_\alpha, \mathbf{T}_\alpha$
β
$\mathbf{S}_\beta, \mathbf{T}_\beta$

FIG. 3.2. *HPS Merge Operation. The merged patch Ω_τ is the union of children Ω_α and Ω_β, i.e. $\Omega_\tau = \Omega_\alpha \cup \Omega_\beta$. Red, green, and blue nodes correspond to index sets $\boldsymbol{I}_1$, $\boldsymbol{I}_2$, and $\boldsymbol{I}_3$, respectively. The merge operation eliminates the nodes on the interface of the children patches.*

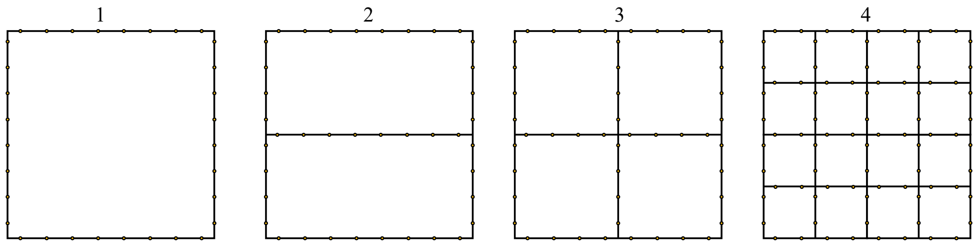

FIG. 3.3. *HPS Solve Stage. Once $\boldsymbol{S}_0$ is formed, apply it to the top level Dirichlet data to get boundary (and solution) data on the interface of the children. Apply the patch solution operator down the tree until each leaf has it's local boundary information. Then apply the solution operator to get the solution data in the interior of each leaf.*

Similar to other direct methods, the HPS method forms an in-memory solution operator that can be applied to several right-hand side vectors. This property makes it ideal for problems where several elliptic solves are necessary. While most iterative methods have better asymptotic performance than other direct methods, the HPS method can be accelerated using hierarchically block seperable (HBS) matrix algebra to achieve near linear asymptotic performance ([8]).

4. Adaptive and Unstructured Mesh Methods for Elliptic Partial Differential Equations. In many applications of elliptic solvers where the dynamics of the problem are localized (i.e. shock waves and fronts) or the geometry of the domain is complicated, more complex meshing is used. These types of meshes include dynamically adaptive meshes, where the mesh is refined/coarsened through time, and unstructured meshes, where the cells or elements are typically polygons or polyhedrons and not logically ordered like Cartesian grids. While solving elliptic PDEs on these types of meshes is more difficult, the methods we have talked about can be extended to work on complex meshes.

4.1. Unstructured Meshes. When working with unstructured meshes, the domain is broken into polygons (2D) or polyhedrons (3D). Finite difference methods are typically difficult to implement on unstructured grids, but other methods like finite volume and finite element methods are often built around these complex geometries. Just as discretization methods on unstructured grids have been developed, so have more robust solvers. For example, in [16], Luo et al. present an unstructured multigrid method for the discontinuous Galerkin method. They use a sequence of solution approximations of different polynomial orders. The ultraspherical spectral element method (ultraSEM) presented by Fortunato et al. is another highly practical extension of the HPS method. In [5], they apply their ultraspherical spectral element method from [20] to the HPS method. Fortunato et al. use spectral discretization on mapped quadrilaterals and triangles and use the same merge process from the HPS method. They show that the HPS method also works well on mapped, unstructured meshes.

4.2. Adaptive Meshes. Hierarchical meshes, like the meshes used in the HPS method, are also well suited for adaptivity. On adaptive, hierarchical meshes like quadtrees (2D) and octrees (3D), a patch can be recursively split to a desired level of refinement. Areas of the domain that have increased dynamics or need higher refinement (say at boundaries or corners) can be resolved with finer meshes, while other areas can be of lower resolution. This adaptivity increases accuracy without significantly increasing runtime.

In [23], Popinet et al. use a quadtree approach to decompose the domain into a hierarchical grid. In locations where more refinement is necessary, they refine the mesh more. For example, in [23], they are solving the Serre-Green-Naghdi model (we will show this as an application in Section 5) which models shallow water equations. As they are looking at a tsunami traveling across the surface of the earth, refinement near the location of the wave fronts is necessary, but without having to have very dense meshing elsewhere. They show that they can implement the multigrid method on a quadtree based mesh.

Additionally, the HPS method has been modified to be used on an adaptive mesh. In [6], Geldermans et al., they show how to use the HPS method on a quadtree of patches. They demonstrate the merge operation between patches on different levels, as well as discuss proper interpolation schemes for their choice of mesh (a Chebyshev tensor product). In addition, the global solution operator formed by this adaptive HPS method can still be applied to multiple right-hand side problems. The study of fast, direct solvers for elliptic problems on adaptively refined quadtrees and octrees is an active area of research.

5. Applications of Elliptic Partial Differential Equations. We will look at two examples where the use of a fast elliptic PDE solver would greatly benefit the solution process. These are cases where an elliptic solve is required multiple times, such as every time step or for multiple problems.

5.1. Navier-Stokes: The Projection Method. The incompressible Navier-Stokes equations is a nonlinear PDE that arises from conservation of momentum and mass in an incompressible fluid. They are expressed as

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = -\frac{1}{\rho}\nabla P + \nu \nabla^2 \mathbf{u}. \tag{5.1}$$

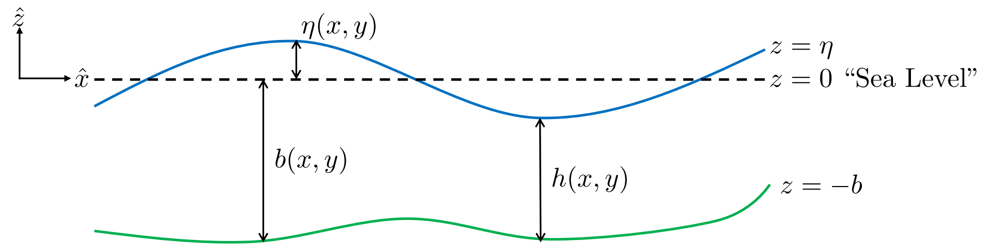

FIG. 5.1. *Shallow Water Equation Reference*

In [4], Chorin first computes an intermediate velocity $\mathbf{u}^*$ by time stepping and ignoring the pressure term

$$\frac{\mathbf{u}^* - \mathbf{u}^n}{\Delta t} = -\mathbf{u}^n \cdot \nabla \mathbf{u}^n + \nu \nabla^2 \mathbf{u}^n \tag{5.2}$$

and then "projecting" the intermediate velocity to the next time step via

$$\mathbf{u}^{n+1} = \mathbf{u}^* - \frac{\Delta t}{\rho} \nabla P^{n+1} \tag{5.3}$$

In order to solve for the right-hand side of the second step, the pressure field at the $n+1$ time step must be known. This is found by solving the following Poisson's equation arising from taking the divergence of the second step and using the conservation of mass to eliminate the $\nabla \cdot \mathbf{u}^{n+1}$ term:

$$\nabla^2 P^{n+1} = \frac{\rho}{\Delta t} \nabla \cdot \mathbf{u}^* \tag{5.4}$$

This projection method requires an elliptic solve at every time step to solve for the pressure field. In applications like this where an elliptic solve is required multiple times, direct methods where one can pre-compute the solution operator and then apply it each time step greatly speeds up the computation.

5.2. The Shallow Water Equations and the Serre-Green-Naghdi Model. The shallow water equations (SWE) form a model for free surface flow derived from potential theory. The key assumption in the SWEs is that the horizontal scale is much larger than the vertical scale. Because a "shallow limit" is assumed, the velocity field in the vertical direction is a depth-averaged velocity. These types of equations are used to model tsunami waves and debris flow, among other things.

Extensions to this model done by Bonneton et al. in [1], [12], and [13] introduce a dispersive term correction in the form of a source term, leading to

$$\frac{\partial \mathbf{q}}{\partial t} + \nabla \cdot \mathbf{F}(\mathbf{q}) = \mathbf{b} + \mathbf{s} \tag{5.5}$$

where

$$\mathbf{q} = \begin{bmatrix} h \\ hu \\ hv \end{bmatrix} \quad \mathbf{F}(\mathbf{q}) = \begin{bmatrix} hu & hv \\ hu^2 & huv \\ huv & hv^2 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} 0 \\ -gh\frac{\partial b}{\partial x} \\ -gh\frac{\partial b}{\partial y} \end{bmatrix} \quad \mathbf{s} = \begin{bmatrix} 0 \\ \frac{\alpha-1}{\alpha} gh \frac{\partial \eta}{\partial x} + d_x \\ \frac{\alpha-1}{\alpha} gh \frac{\partial \eta}{\partial y} + d_y \end{bmatrix}. \tag{5.6}$$

The water column height is denoted by $h(x, y)$, with surface velocity $(u(x, y), v(x, y))$, $b(x, y)$ is a bathymetry term (elevation), $\eta(x, y)$ is the free surface height and dispersive term correction (d_x, d_y). Figure 5.1 shows the typical setup for this kind of problem.

In order to solve for the dispersive source term, one must solve the following elliptic equation

$$\big(\mathbf{I} + \alpha \mathbf{T}^b_{diag}\big)\mathbf{d} = \mathbf{f}(h, \mathbf{u}, b, \eta) \tag{5.7}$$

where the details of the right-hand side function can be found in [13]. The operator $\mathbf{T}^b_{diag}$ is an elliptic-like operator that does not change with time. So, for a static mesh, the dispersive operator can be factorized via a direct method and applied at each time step. This reduces the need to solve the elliptic PDE every time step to just an application of the solution operator every time step. If the mesh is adapted, the solution operator would need to be reformed. However, direct methods like the HPS method can allow for a local adjustment to the solution operator, further accelerating the solution.

6. Conclusion. The discretization of elliptic partial differential equations leads to linear systems that can be solved with efficient solvers, taking advantage of the structure of the problem. Discretization methods include the finite difference method, finite volume method, finite element method, spectral methods, and more. These discretizations lead to linear systems of the standard form $\mathbf{Au} = \mathbf{f}$, where $\mathbf{A}$ is sparse and structured. Efficient methods such as splitting and Krylov methods, matrix factorizations, and hierarchical domain decomposition methods can take advantage of the structure of the linear system. These solution methods can also be expanded upon in order to achieve near linear performance and are well suited for high-performance applications.

In addition, we presented various areas of study that could greatly benefit from an efficient, fast, and scalable solver for elliptic PDEs. This includes many applications in computational fluid dynamics like the incompressible Navier-Stokes and the shallow water wave equations. Using projection methods for the Navier-Stokes equations and the Serre-Green-Naghdi dispersive model leads to elliptic terms that must be solved at each time step. Fast, direct methods like the ones currently being studied and implemented can accelerate these areas as well as any area that solves an elliptic partial differential equation.

REFERENCES

[1] P. Bonneton, F. Chazel, D. Lannes, F. Marche, and M. Tissier, *A splitting approach for the fully nonlinear and weakly dispersive Green-Naghdi model*, 230 (2011), pp. 1479–1498.

[2] A. Brandt, *Multi-level adaptive solutions to boundary-value problems*, 31 (1977), pp. 333–390.

[3] W. L. Briggs, V. E. Henson, and S. F. McCormick, *A multigrid tutorial*, SIAM, 2000.

[4] A. J. Chorin, *The numerical solution of the Navier-Stokes equations for an incompressible fluid*, 73 (1967), pp. 928–931.

[5] D. Fortunato, N. Hale, and A. Townsend, *The ultraspherical spectral element method*, (2020), p. 110087.

[6] P. Geldermans and A. Gillman, *An adaptive high order direct solution technique for elliptic boundary value problems*, 41 (2019), pp. A292–A315.

[7] A. George, *Nested dissection of a regular finite element mesh*, 10 (1973), pp. 345–363.

[8] 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*, 36 (2014), pp. A2023–A2046.

[9] M. R. Hestenes, E. Stiefel, et al., *Methods of conjugate gradients for solving linear systems*, vol. 49, NBS Washington, DC, 1952.

[10] N. J. Higham, *Accuracy and stability of numerical algorithms*, SIAM, 2002.

[11] P. Kagan, A. Fischer, and P. Z. Bar-Yoseph, *New B-Spline Finite Element approach for geometrical design and mechanical analysis*, 41 (1998), pp. 435–458.

[12] D. Lannes and P. Bonneton, *Derivation of asymptotic two-dimensional time-dependent equations for surface water wave propagation*, 21 (2009), p. 016601.

[13] D. Lannes and F. Marche, *A new class of fully nonlinear and weakly dispersive Green–Naghdi models for efficient 2D simulations*, 282 (2015), pp. 238–268.

[14] R. J. LeVeque, *Finite difference methods for ordinary and partial differential equations: steady-state and time-dependent problems*, SIAM, 2007.

[15] R. J. Lipton, D. J. Rose, and R. E. Tarjan, *Generalized nested dissection*, 16 (1979), pp. 346–358.

[16] H. Luo, J. D. Baum, and R. Löhner, *A p-multigrid discontinuous Galerkin method for the Euler equations on unstructured grids*, 211 (2006), pp. 767–783.

[17] P. Martinsson, *A direct solver for variable coefficient elliptic PDEs discretized via a composite spectral collocation method*, 242 (2013), pp. 460–479.

[18] P. Martinsson, *The hierarchical Poincaré-Steklov (HPS) solver for elliptic PDEs: A tutorial*, arXiv preprint arXiv:1506.01308, (2015).

[19] P.-G. Martinsson, *Fast direct solvers for elliptic PDEs*, SIAM, 2019.

[20] S. Olver and A. Townsend, *A fast and well-conditioned spectral method*, 55 (2013), pp. 462–489.

[21] V. R. P. Martinsson, *A fast direct solver for boundary integral equations in two dimensions*, (2004).

[22] A. T. Patera, *A spectral element method for fluid dynamics: laminar flow in a channel expansion*, 54 (1984), pp. 468–488.

[23] S. Popinet, *A quadtree-adaptive multigrid solver for the Serre–Green–Naghdi equations*, 302 (2015), pp. 336–358.

[24] A. Quarteroni, R. Sacco, and F. Saleri, *Numerical mathematics*, vol. 37, Springer Science & Business Media, 2010.

[25] Y. Saad and M. H. Schultz, *GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems*, 7 (1986), pp. 856–869.

[26] A. Townsend and S. Olver, *The automatic solution of partial differential equations using a global spectral method*, 299 (2015), pp. 106–123.

[27] L. N. Trefethen and D. Bau III, *Numerical linear algebra*, vol. 50, SIAM, 1997.