LECTURES on COMPUTATIONAL NUMERICAL ANALYSIS of PARTIAL DIFFERENTIAL EQUATIONS

J. M. McDonough

Departments of Mechanical Engineering and Mathematics
University of Kentucky

Contents

1 Introduction **1**

1.1 Basic Mathematical Definitions . . . 2

1.1.1 Norms and related ideas . . . 2

1.1.2 Convergence of sequences . . . 5

1.1.3 Consistency, stability and convergence . . . 7

1.2 Classifications of Partial Differential Equations . . . 10

1.2.1 Equation type . . . 10

1.2.2 Form of nonlinearity . . . 11

1.3 Well Posedness of PDE Problems . . . 12

1.4 Discretization and Gridding of PDE Problems . . . 13

1.4.1 Discretization techniques . . . 13

1.4.2 Gridding methods . . . 15

1.5 Summary . . . 16

2 Numerical Solution of Elliptic Equations **17**

2.1 Background . . . 19

2.1.1 Iterative solution of linear systems—an overview . . . 19

2.1.2 Basic theory of linear iterative methods . . . 20

2.2 Successive Overrelaxation . . . 25

2.2.1 Jacobi iteration . . . 25

2.2.2 SOR theory . . . 28

2.2.3 Some modifications to basic SOR . . . 34

2.3 Alternating Direction Implicit (ADI) Procedures . . . 40

2.3.1 ADI with a single iteration parameter . . . 41

2.3.2 ADI: the commutative case . . . 45

2.3.3 ADI: the noncommutative case . . . 49

2.4 Incomplete LU Decomposition (ILU) . . . 49

2.4.1 Basic ideas of ILU decomposition . . . 49

2.4.2 The strongly implicit procedure (SIP) . . . 51

2.5 Preconditioning . . . 57

2.6 Conjugate-Gradient Acceleration . . . 59

2.6.1 The method of steepest descent . . . 59

2.6.2 Derivation of the conjugate-gradient method . . . 61

2.6.3 Relationship of CG to other methods . . . 63

2.7 Introduction to Multigrid Procedures . . . 64

2.7.1 Some basic ideas . . . 64

2.7.2 The $\boldsymbol{h}$-$\boldsymbol{2h}$ two-grid algorithm . . . 66

2.7.3 $\boldsymbol{\ell}$-grid multigrid methods . . . 70

2.7.4 The full multigrid method . . . 71

2.7.5 Some concluding remarks . . . 74

2.8 Domain-Decomposition Methods . . . 75
2.8.1 The alternating Schwarz procedure . . . 77
2.8.2 The Schur complement . . . 79
2.8.3 Multiplicative and additive Schwarz methods . . . 81
2.8.4 Multilevel domain-decomposition methods . . . 87
2.9 Summary . . . 89

3 Time-Splitting Methods for Evolution Equations 91
3.1 Alternating Direction Implicit Methods . . . 92
3.1.1 Peaceman–Rachford ADI . . . 92
3.1.2 Douglas–Rachford ADI . . . 95
3.1.3 Implementation of ADI schemes . . . 96
3.2 Locally One-Dimensional Methods . . . 98
3.3 General Douglas–Gunn Procedures . . . 102
3.3.1 D–G methods for two-level difference equations . . . 102
3.3.2 D–G methods for multi-level difference equations . . . 108
3.4 Summary . . . 111

4 Various Miscellaneous Topics 113
4.1 Nonlinear PDEs . . . 113
4.1.1 The general nonlinear problem to be considered . . . 113
4.1.2 Explicit integration of nonlinear terms . . . 113
4.1.3 Picard iteration . . . 114
4.1.4 The Newton–Kantorovich Procedure . . . 114
4.2 Systems of PDEs . . . 119
4.2.1 Example problem—a generalized transport equation . . . 119
4.2.2 Quasilinearization of systems of PDEs . . . 120
4.3 Numerical Solution of Block-Banded Algebraic Systems . . . 123
4.3.1 Block-banded LU decomposition—how it is applied . . . 123
4.3.2 Block-banded LU decomposition details . . . 124
4.3.3 Arithmetic operation counts . . . 126
4.4 Cell-***Re*** and Aliasing Treatments . . . 127
4.4.1 The cell-***Re*** problem—its definition and treatment . . . 127
4.4.2 Treatment of effects of aliasing . . . 136
4.5 More Advanced Spatial Discretizations . . . 142
4.5.1 Basic approximations for mixed derivatives . . . 143
4.5.2 Mixed derivatives with variable coefficients . . . 144
4.5.3 Discretization of self-adjoint form second (unmixed) derivatives . . . 147
4.6 Treatment of Advanced Boundary Conditions . . . 150
4.6.1 General linear BCs . . . 151
4.6.2 Outflow BCs . . . 152
4.7 Summary . . . 153

5 Numerical Solution of Partial Differential Equations on Irregular Domains—Grid Generation 155
5.1 Overview of Grid Structures . . . 155
5.1.1 Unstructured grids . . . 156
5.1.2 Structured grids . . . 158

References 159

List of Figures

1.1 Schematic of a contraction mapping, $[a, b] \rightarrow [f(a), f(b)]$. 7
1.2 Methods for spatial discretization of partial differential equations; (a) finite difference, (b) finite element and (c) spectral. 14

2.1 $N_x \times N_y$–point grid and mesh star for discretizations of Eq. (2.1). 18
2.2 Sparse, banded matrices arising from finite-difference discretizations of elliptic operators: (a) 5-point discrete Laplacian; (b) 9-point general discrete elliptic operator. 18
2.3 Qualitative comparison of required arithmetic for various iterative methods for 2-D elliptic problems. 20
2.4 Qualitative representation of error reduction during linear fixed-point iterations. 23
2.5 Discretization of the Laplace/Poisson equation on a rectangular grid of $N_x \times N_y$ points. . . 26
2.6 Band structure of Jacobi iteration matrix for Laplace/Poisson equation. 27
2.7 Geometric test of consistent ordering. (a) consistent ordering, (b) nonconsistent ordering. . 31
2.8 Spectral radius of SOR iteration matrix vs. ω. 32
2.9 Red-black ordering for discrete Laplacian. 35
2.10 Comparison of computations for point and line SOR showing grid stencils and red-black ordered lines. 37
2.11 Matrices arising from decomposition of A: (a) H matrix, (b) V matrix, (c) S matrix. 41
2.12 (a) 7-band finite-difference matrix; (b) corresponding mesh star. 51
2.13 Finite-difference grid for demonstrating structure of SIP matrices. 52
2.14 Level set contours and steepest-descent trajectory of 2-D quadratic form. 60
2.15 Level set contours, steepest-descent trajectory and conjugate gradient trajectory of 2-D quadratic form. 61
2.16 Comparison of h and $2h$ grids for multigrid implementations. 68
2.17 Multigrid V-cycles; (a) $\ell = 2$, and (b) $\ell = 3$. 71
2.18 Multigrid V-cycles with $\ell = 3$ and different values of γ; (a) $\gamma = 1$, (b) $\gamma = 2$ and (c) $\gamma = 3$. . 71
2.19 Four-Level, V-cycle full multigrid schematic. 72
2.20 L-shaped grid depicting basic domain-decomposition approach. 75
2.21 Keyhole-shaped domain $\Omega_1 \cup \Omega_2$ considered by Schwarz [41]. 77
2.22 Simple two-subdomain problem to demonstrate Schur complement. 80
2.23 Four-subdomain problem demonstrating types of points required for Schur complement construction. 81
2.24 Domain decomposition with two overlapping subdomains; (a) domain geometry, (b) matrix structure. 82
2.25 Schematic depicting two-level domain decomposition and approximate Schur complement. . 88

3.1 Implementation of line-by-line solves for time splitting of time-dependent problems. 97
3.2 Numerical Dirichlet–Neumann problem; points on dashed lines are "image" points needed for implementation of centered discretizations. 98

4.1 Grid point to grid point oscillations caused by cell-Re problem. 130
4.2 Dependence of z_+ on cell Re. 131
4.3 Under-sampled oscillatory function demonstrating effects of aliasing. 137
4.4 Schematic of a discrete mollifier. 140
4.5 Wavenumber response of Shuman filter for various values of filter parameter. 142
4.6 Finite-difference grid with uniform grid spacings $h_x \neq h_y$ in neighborhood of grid point (i, j).143

5.1 Geometry of (a) structured and (b) unstructured grids. 156
5.2 Simple 1-D geometry for nonuniform-grid construction of second derivative approximations. 157

Chapter 1

Introduction

The purpose of these lectures is to present a set of straightforward numerical methods with applicability to essentially any problem associated with a partial differential equation (PDE) or system of PDEs independent of type, spatial dimension or form of nonlinearity. In this first chapter we provide a review of elementary ideas from "pure" mathematics that usually are covered in a first course in numerical analysis, but which are often not introduced in "engineering computation" courses. Because many of these ideas are essential to understanding correct numerical treatments of PDEs, we include them here. We note that these can all be found in various sources, including the elementary numerical analysis lecture notes of McDonough [1].

In Chap. 2 we provide a quite thorough and reasonably up-to-date numerical treatment of elliptic partial differential equations. This will include detailed analyses of classical methods such as successive overrelaxation (SOR) as well as various modern techniques, especially multigrid and domain decomposition methods. Chapter 3 presents a detailed analysis of numerical methods for time-dependent (evolution) equations and emphasizes the very efficient so-called "time-splitting" methods. These can, in general, be equally-well applied to both parabolic and hyperbolic PDE problems, and for the most part these will not be specifically distinguished. On the other hand, we will note, via examples, some features of these two types of PDEs that make details of their treatment somewhat different, more with respect to the discretizations employed than with specific solution techniques.

Chapter 4 is devoted to presenting a number of miscellaneous topics and methods, many of which could be applicable to any of the types of problems studied in earlier chapters. These will include smoothing techniques, methods for treating nonlinear equations, approaches to be employed for systems of equations, including modifications to the required numerical linear algebra, some special discretizations for specific types of differential operators, and boundary condition implementation beyond that usually given in elementary courses.

Finally, Chap. 5 will provide an introduction to methods used for PDE problems posed on arbitrary spatial domains. There are many types of such problems and, correspondingly, many ways in which to deal with them. Herein, we will begin with a review of advantages and disadvantages of various of the approaches used to treat such problems. Then we will focus attention on two particular methods—one very old and dependable, and the other fairly new and still the topic of much investigation: namely, use of generalized coordinates and so-called immersed boundary methods, respectively.

At the conclusion of these lectures it is hoped that readers will have acquired a basic background permitting them to solve very general PDE problems, *viz.*, systems of nonlinear partial differential equations (whose solutions may not be very regular) posed on essentially arbitrary geometric domains (including those that are moving and/or deforming). Examples of such systems of equations and their associated problems arise in many different engineering, physical sciences, and biological sciences disciplines—and in various combinations of these. Our goal in these lectures is to prepare the reader to approach essentially any of these in straightforward and computationally-efficient ways.

We remark that all analyses will be presented in the context of finite-difference/finite-volume discretization of the PDE(s) involved on structured grids. We will provide more detailed justification for this choice at various junctures throughout the lectures, but we here note that this combination represents the simplest and most efficient approach to the PDE problem, in general. Nevertheless, many (in fact, most) of the algorithms we will describe can easily be employed in the context of finite-element methods, as well, and with modification—possibly significant in some cases—on unstructured grids.

The remainder of this introductory chapter includes sections in which each of the following topics is discussed:

i) some basic mathematical definitions,

ii) classifications of partial differential equations,

iii) the notion of well posedness, and

iv) an overview of methods for discretizing PDEs and gridding their domains.

1.1 Basic Mathematical Definitions

In this section we will introduce some basic definitions along with standard notation corresponding to them. We will not provide the detail here that is already available in [1] and elsewhere, but instead mainly supply a reminder of some of the mathematical notions that are crucial in the study of numerical methods for PDEs. These will include such mathematical constructs as norm, the Cauchy–Schwarz inequality, convergence, a contraction mapping principle, consistency, stability, and the Lax equivalence theorem.

1.1.1 Norms and related ideas

One of the most fundamental properties of any object, be it mathematical or physical, is its size. Of course, in numerical analysis we are always concerned with the size of the error of any particular numerical approximation, or computational procedure. There is a general mathematical object, called the *norm*, by which we can assign a number corresponding to the size of various mathematical entities.

Definition 1.1 *Let S be a (finite- or infinite-dimensional) vector space, and let $\|\cdot\|$ denote the mapping $S \to \mathbb{R}_+ \cup \{0\}$ with the following properties:*

i) $\|v\| \geq 0, \;\forall\, v \in S$ *with* $\|v\| = 0$ *iff* $v \equiv 0,$

ii) $\|av\| = |a|\;\|v\|, \quad \forall\, v \in S,\; a \in \mathbb{R},$

iii) $\|v + w\| \leq \|v\| + \|w\| \quad \forall\, v, w \in S.$

Then $\|\cdot\|$ is called a <u>norm</u> for S.

Note that we can take S to be a space of vectors, functions or even operators, and the above properties apply. It is important to observe that for a given space S there are, in general, many different mappings $\|\cdot\|$ having the properties required by the above definition. We will give a few specific examples.

If S is a finite-dimensional space of vectors with elements $v = (v_1, v_2, \ldots, v_N)^T$ then a familiar measure of the size of v is its Euclidean length,

$$\|v\|_2 = \left(\sum_{i=1}^{N} v_i^2\right)^{\frac{1}{2}} . \tag{1.1}$$

The proof that $\|\cdot\|_2$, often called the *Euclidean norm*, or simply the *2-norm*, satisfies the three conditions of the definition is straightforward, and is left to the reader. (We note here that it is common in numerical analysis to employ the subscript E to denote this norm and use the subscript 2 for the "spectral" norm of matrices. But we have chosen to defer to notation more consistent with pure mathematics.) Another useful norm often encountered in practice is the *max norm* or *infinity norm* defined as

$$\|v\|_\infty = \max_{1\le i\le N} |v_i| \;. \tag{1.2}$$

If S is an infinite-dimensional space, then we can define a particular family of norms as follows:

Definition 1.2 *Let S be the space L^p, $1 \le p < \infty$, on a domain Ω, and let $f \in L^p(\Omega)$. Then the* <u>*L^p norm of f on Ω*</u> *is*

$$\|f\|_{L^p} \equiv \left(\int_\Omega |f|^p \, d\boldsymbol{x}\right)^{1/p} , \tag{1.3}$$

where Ω is an arbitrary (usually) spatial domain, possibly possessing infinite limits, and $\boldsymbol{x}$ denotes a corresponding coordinate vector with the appropriate dimension.

We remark that, in fact, the differential associated with this coordinate vector is actually a measure (a point we will not emphasize in these lectures), and the integral appearing in the definition is that of Lebesgue—hence, the L notation for the associated spaces—rather than the familiar Riemann integral from Freshman Calculus. These points will occasionally arise in the sequel. Also, we note that by definition $f \in L^p(\Omega)$ iff $\|f\|_{L^p} < \infty$; *i.e.*, the integral in Eq. (1.3) exists.

When $p = 2$ we obtain a very special case of the L^p spaces known as a *Hilbert space*, and defined as follows:

Definition 1.3 *A* <u>*Hilbert space*</u> *is a complete, normed, linear space in which the norm is induced by an inner product.*

We note that completeness in this context means that any convergent sequence in the space converges to an element of the space (*i.e.*, to an element <u>within</u> the space). A linear space is simply one in which finite linear combinations of elements of the space are also elements of the space. Finally, the notion that the norm is "induced" by an inner product can be understood in the following way. First, we must define inner product.

Definition 1.4 *Let f and g be in $L^2(\Omega)$. Then the inner product of f and g is defined as*

$$\langle f, g\rangle \equiv \int_\Omega fg \, d\boldsymbol{x} \,, \tag{1.4}$$

where, again Ω is a prescribed domain.

Now observe that if $f = g$ we have

$$\langle f, f\rangle \equiv \int_\Omega f^2 \, d\boldsymbol{x} = \|f\|^2_{L^2} \,, \tag{1.5}$$

if f takes on values in $\mathbb{R}$. (The case when f has complex values is only slightly different and will not be needed herein.) Hence, the inner product of f with itself is just the square of the L^2 norm, and we say this norm is induced by the inner product. Because of this, the space L^2 is often termed the "canonical" Hilbert space. We mention that there are many other—usually more complicated—Hilbert spaces, but all, by definition, have a norm induced by an (often more complicated) inner product. All such spaces are typically referred to as "inner-product" spaces.

We can now introduce a very important property that relates the inner product and the norm, the *Cauchy–Schwarz inequality.*

Theorem 1.1 *(Cauchy–Schwarz) Let f and g be in $L^2(\Omega)$, $\Omega \subseteq \mathbb{R}^d$, d typically 1,2 or 3. Then*

$$\langle f, g\rangle \le \|f\|_{L^2} \|g\|_{L^2} . \tag{1.6}$$

Moreover, the L^2 norms can generally be replaced by any norm induced by an inner product, and the inequality (1.6) will hold. Many of the Sobolev spaces encountered in finite-element analysis, and generally in studies of the Navier–Stokes equations, are of this type. It is clear from (1.6) that if f and g are in L^2 (hence, their norms are bounded), then this must also be true of their inner product. This simple observation is extremely important for the theory of Fourier series because it implies existence of Fourier coefficients for all functions in L^2 (and other Hilbert spaces).

We next need to consider some corresponding ideas regarding norms of operators. The general definition of an *operator norm* is as follows.

Definition 1.5 *Let A be an operator whose domain is $\mathcal{D}$. Then the norm of A is defined as*

$$\|A\| \equiv \max_{\substack{\|x\|=1 \\ x\in\mathcal{D}(A)}} \|Ax\| . \tag{1.7}$$

It is easy to see that this is equivalent to

$$\|A\| = \max_{\substack{\|x\|\neq 0 \\ x\in\mathcal{D}(A)}} \frac{\|Ax\|}{\|x\|} ,$$

from which follows an inequality similar to the Cauchy–Schwarz inequality for vectors (or functions),

$$\|Ax\| \le \|A\|\|x\| . \tag{1.8}$$

We should remark here that (1.8) actually holds only in the finite-dimensional case when the matrix and vector norms appearing in the expression are "compatible," and this relationship is often used as the definition of *compatibility*. We will seldom need to employ this concept in the present lectures, and the reader is referred to, *e.g.*, Isaacson and Keller [2] (Chap. 1) for additional information regarding this notion.

We observe that neither (1.7) nor the expression following it is suitable for practical calculations; we now present three norms that are readily computed, at least for $M \times N$ matrices, and are widely used in numerical linear algebra. The first of these is the *2-norm*, given in the matrix case by

$$\|A\|_2 = \left(\sum_{i,j=1}^{M,N} a_{ij}^2\right)^{\frac{1}{2}} , \tag{1.9}$$

which is completely analogous to the vector 2-norm of Eq. (1.1). Moreover, the same general construction is used for norms of matrices of higher dimension.

Two other norms are also frequently employed. These are the *1-norm*

$$\|A\|_1 = \max_{1\le j\le N} \sum_{i=1}^{M} |a_{ij}| , \tag{1.10}$$

and the *infinity norm*

$$\|A\|_\infty = \max_{1\le i\le M} \sum_{j=1}^{N} |a_{ij}| . \tag{1.11}$$

These too have generalizations to higher dimensions, but which are no longer unique.

We note that although the definition of the operator norm given above was not necessarily finite-dimensional, we have here given only finite-dimensional practical computational formulas. We will see later that this is not really a serious restriction because problems involving differential operators, one of the main instances where norms of infinite-dimensional operators are needed, are essentially always solved via discrete approximations leading to finite-dimensional matrix representations.

There is a final, general comment that should be made regarding norms. It arises from the fact, mentioned earlier, that in any given vector space many different norms might be employed. A comparison of the formulas in Eqs. (1.1) and (1.2), for example, will show that the number one obtains to quantify the size of a mathematical object, a vector in this case, will change according to which formula is applied. Thus, a reasonable question is, "How do we decide which norm to use?" It turns out, for the finite-dimensional spaces we will mainly deal with herein, that it really does not matter which norm is used, provided only that the same one is used when making comparisons between similar mathematical objects. This is the content of what is known as the *norm equivalence theorem*: all norms are equivalent on finite-dimensional spaces in the sense that if a sequence converges in one norm, it will converge in any other norm (see Ref. [2], Chap. 1). This implies that in practical numerical calculations we should usually employ the norm that requires the least amount of floating-point arithmetic for its evaluation. But we note here that the situation is rather different for infinite-dimensional spaces associated with, for example, analytical studies of PDEs. In particular, for problems involving differential equations, determination of the function space in which a solution exists (and hence, the appropriate norm) is a significant part of the overall problem. We will at times need to deal with this later.

1.1.2 Convergence of sequences

It will be clear, as we proceed through these lectures, that convergence of sequences generated via iterative methods, and also by changing discretization step sizes, is a fundamental aspect of numerical solution of PDEs. Hence, we present the following elementary definitions and discussions associated with their application.

Definition 1.6 *Let $\{y_m\}_{m=1}^{\infty}$ be a sequence in $\mathbb{R}^N$. The sequence is said to* <u>*converge*</u> *to the limit $y \in \mathbb{R}^N$ if $\forall\ \epsilon > 0\ \exists\ M$ (depending on ϵ) $\ni \forall\ m \geq M$, $\|y - y_m\| < \epsilon$. We denote this by $\lim\limits_{m\to\infty} y_m = y$.*

We note here that the norm has not been specified, and we recall the earlier remark concerning equivalence of norms in finite-dimensional spaces. (More information on this can be found, for example, in Apostol [3].) We also observe that when $N = 1$, we merely replace norm $\|\cdot\|$ with absolute value $|\cdot|$.

It is fairly obvious that the above definition is not generally of practical value, for if we knew the limit y, <u>which is required to check convergence</u>, we probably would not need to generate the sequence in the first place. To circumvent such difficulties mathematicians invented the notion of a *Cauchy sequence* given in the following definition.

Definition 1.7 *Let $\{y_m\}_{m=1}^{\infty} \in \mathbb{R}^N$, and suppose that $\forall\ \epsilon > 0\ \exists\ M$ (depending on ϵ) $\ni \forall\ m, n \geq M$, $\|y_m - y_n\| < \epsilon$. Then $\{y_m\}$ is a* <u>*Cauchy sequence*</u>.

By itself, this definition would not be of much importance; but it is a fairly easily proven fact from elementary analysis (see, *e.g.*, [3]) that every Cauchy sequence in a complete metric space converges to an element of that space. It is also easy to show that $\mathbb{R}^N$ is a complete metric space $\forall\ N < \infty$. Thus, we need only demonstrate that successive iterates, for example, form a Cauchy sequence, and we can then conclude that the sequence converges in the sense of the earlier definition.

Although this represents a considerable improvement over trying to use the basic definition in convergence tests, it still leaves much to be desired. In particular, the definition of a Cauchy sequence requires that $\|y_m - y_n\| < \epsilon$ hold $\forall\ \epsilon > 0$, and $\forall\ m, n \geq M$, where M, itself, is not specified, *a priori*. For computational purposes it is completely unreasonable to choose ϵ smaller than the absolute normalized precision (the *machine* ϵ) of the floating-point arithmetic employed. It is usually sufficient to use values of $\epsilon \sim \mathcal{O}(e/10)$, where e is the acceptable error for the computed results. The more difficult part of the definition to satisfy is " $\forall\ m,n \geq M$." However, for well-behaved sequences, it is typically sufficient to choose $n = m + k$ where k is a specified integer between one and, say 100. (Often $k = 1$ is used.) The great majority of computer-implemented iteration schemes test convergence in this way; that is, the computed sequence $\{y_m\}$ is considered to be converged when $\|y_{m+1} - y_m\| < \epsilon$ for some prescribed (and often completely fixed) ϵ. In many practical calculations $\epsilon \approx 10^{-3}$ represents quite sufficient accuracy, but of course this is problem dependent.

Now that we have a means by which the convergence of sequences can be tested, we will study a systematic method for generating these sequences in the context of solving equations. This method is based on a very powerful and basic notion from mathematical analysis, the *fixed point* of a function, or mapping.

Definition 1.8 *Let $f\colon \mathcal{D} \to \mathcal{D}$, $\mathcal{D} \subset \mathbb{R}^N$. Suppose $x \in \mathcal{D}$, and $x = f(x)$. Then x is said to be a* $\underline{\textit{fixed point of } f \textit{ in } \mathcal{D}}$.

We see from this definition that a fixed point of a mapping is simply any point that is $\underline{\text{mapped back to itself}}$ by the mapping. Now at first glance this might not seem too useful—some point being repeatedly mapped back to itself, over and over again. But the expression $x = f(x)$ can be rewritten as

$$x - f(x) = 0, \tag{1.12}$$

and in this form we recognize that a fixed point of f is a *zero* (or *root*) of the function $g(x) \equiv x - f(x)$. Hence, if we can find a way to compute fixed points, we automatically obtain a method for solving equations. Indeed, intuition suggests that we might try to find a fixed point of f via the following iteration scheme:

$$\begin{aligned} x_1 &= f(x_0) \\ x_2 &= f(x_1) \\ &\cdot \\ &\cdot \\ &\cdot \\ x_m &= f(x_{m-1}) \\ &\cdot \\ &\cdot \\ &\cdot \end{aligned}$$

where x_0 is an initial guess. This procedure generates the sequence $\{x_m\}$ of approximations to the fixed point x^*, and we continue this until $\|x_{m+1} - x_m\| < \epsilon$. We remark that this approach, often called *successive approximation*, will be widely used in subsequent chapters.

The following theorem utilizes this basic idea of successive approximation to provide sufficient conditions for convergence of fixed-point iterations in finite-dimensional spaces of dimension N. Numerous similar results are known for infinite-dimensional spaces as well, but we will not consider these here.

Theorem 1.2 *(Contraction Mapping Principle) Let f be continuous on a compact subset $\mathcal{D} \subset \mathbb{R}^N$ with $f\colon \mathcal{D} \to \mathcal{D}$, and suppose $\exists$ a positive constant $L < 1 \ni$*

$$\|f(y) - f(x)\| \leq L\|y - x\| \quad \forall\ x, y \in \mathcal{D}\,. \tag{1.13}$$

Then $\exists$ a $\underline{\textit{unique}}$ $x^* \in \mathcal{D} \ni x^* = f(x^*)$, and the sequence $\{x_m\}_{m=0}^{\infty}$ generated by $x_{m+1} = f(x_m)$ converges to x^* from any (and thus, every) initial guess, $x_0 \in \mathcal{D}$.

The inequality (1.13) is of sufficient importance to merit special attention.

Definition 1.9 *The inequality,*

$$\|f(y) - f(x)\| \leq L\|y - x\|\,, \quad \forall\ x, y \in \mathcal{D}\,,$$

is called a $\underline{\textit{Lipschitz condition}}$*, and* L *is the* $\underline{\textit{Lipschitz constant}}$*. Any function* f *satisfying such a condition is said to be a* $\underline{\textit{Lipschitz function}}$*.*

There are several things to note regarding the above theorem. The first is that satisfaction of the Lipschitz condition with $L < 1$ is sufficient, but not always necessary, for convergence of the corresponding iterations. In particular, it implies the contractive properties of the function f, as illustrated in Fig. 1.1 for a simple scalar case. It is clear from this figure that $|f(b) - f(a)| < |b - a|$, so if we take $\mathcal{D} = [a, b]$ in the definition, it follows that $L < 1$ must hold. In other words, the interval $[a, b]$ is contracted when it is mapped by the function f. Second, for any set $\mathcal{D}$, and mapping f with (1.13) holding throughout, x^* is

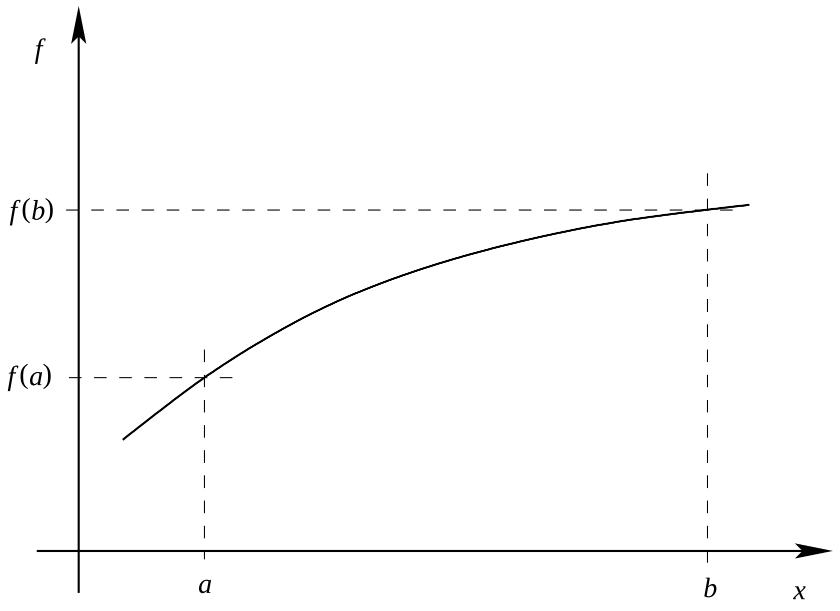

Figure 1.1: Schematic of a contraction mapping, $[a, b] \to [f(a), f(b)]$.

the $\underline{\text{unique}}$ fixed point of f in $\mathcal{D}$. Furthermore, the iterations will converge to x^* using any starting guess, whatever, so long as it is an element of the set $\mathcal{D}$. In addition, the hypothesis that f is continuous in $\mathcal{D}$ is essential. It is easy to construct examples of iteration functions satisfying all of the stated conditions except continuity, and for which the iterations fail to converge. On the other hand, the compactness requirement of the theorem statement is somewhat technical, and it is needed mainly for rigorous mathematical analyses.

1.1.3 Consistency, stability and convergence

Here, we present some basic, fairly simple, descriptions of these terms. By *consistency* we mean that the difference approximation converges to the $\underline{\text{PDE}}$ as discretization step sizes approach 0, and by *stability* we mean that the solution to the difference equation does not increase with time at a faster rate than does the solution to the differential equation. *Convergence* implies that $\underline{\text{solutions}}$ to the difference equation approach those of the PDE, again, as discretization step sizes are refined. If this does not occur, the associated numerical approximations are useless. Clearly, consistency and stability are crucial properties for a difference scheme because of the following theorem due to Lax (see Richtmyer and Morton [4]).

Theorem 1.3 *(Lax Equivalence Theorem) Given a well-posed linear initial-value problem, and a corresponding consistent difference approximation, the resulting grid functions converge to the solution of the* *differential equation(s) as spatial and temporal step sizes, respectively, $h, k \to 0$ if and only if the difference approximation is stable.*

It is important to understand the content of this theorem. First, the notion of well posedness will be treated in more detail later in this chapter; here we can associate it with situations in which the problem is guaranteed to have a well-defined solution. (This, of course, is not always the case.) One can deduce from the theorem the less than obvious fact that consistency of a difference approximation is not a sufficient condition for guaranteeing that the grid functions produced by the scheme actually converge to the solution of the original differential equation as discretization step sizes are refined. In particular, both consistency and stability are required. As will be evident in what follows in later chapters, consistency of a difference approximation is usually very straightforward though sometimes rather tedious to prove, while proof of stability can often be quite difficult.

We observe here that although the Lax equivalence theorem is extremely important, it is also quite restricted in scope. At the same time, convergence of numerically-computed grid functions is one of the most fundamental properties that must be demonstrated for any PDE approximation—and, we remark that this is too often ignored, especially by engineers and physicists. Because of its importance, which will be further stressed in the sequel, we here present some details of conducting such tests in a simple scalar environment. (The reader can find essentially the same material in [1].)

Clearly, if we know the solution to the problem we are solving ahead of time we can always exactly determine the error of the numerical solution. But, of course, if we already know the answer, we would not need a numerical solution in the first place, in general. (An important exception is the study of "model" problems when validating a new algorithm and/or computer code.)

It turns out that a rather simple test for accuracy can—and should—always be performed on solutions represented by a grid function. Namely, we employ a Cauchy convergence test on the grid function as discretization step sizes are reduced. For grid functions we generally have available additional qualitative information, derived from the numerical method itself, about the theoretical convergence <u>rate</u> of the grid functions generated by the method. In particular, we almost always have the truncation error expansion at our disposal. For example, for any sufficiently smooth u obtained via a discrete approximation using a step size h at a grid point i we would have

$$u_i^h = u(x_i) + \tau_1 h^{q_1} + \cdots = u(x_i) + \mathcal{O}\left(h^{q_1}\right),$$

and by changing the step size to rh we have

$$u_i^{rh} = u(x_i) + \tau_1 r^{q_1} h^{q_1} + \cdots .$$

The dominant error in the first case is

$$e_i^h \equiv u(x_i) - u_i^h = -\tau_1 h^{q_1} , \tag{1.14}$$

and in the second case it is

$$e_i^{rh} = u(x_i) - u_i^{rh} = -\tau_1 r^{q_1} h^{q_1} , \tag{1.15}$$

provided h is sufficiently small to permit neglect of higher-order terms in the expansions. Thus, the theoretical ratio of the errors for two different step sizes is <u>known</u> to be simply

$$\frac{e_i^{rh}}{e_i^h} = r^{q_1} . \tag{1.16}$$

Hence, for a second-order method ($q_1 = 2$) a reduction in the step size by a factor of two ($r = \frac{1}{2}$) leads to a reduction in error given by

$$r^{q_1} = \left(\frac{1}{2}\right)^2 = \frac{1}{4} ;$$

i.e., the error is reduced by a factor of four.

In practical problems we usually do not know the exact solution, $u(x)$; hence we cannot calculate the true error. However, if we obtain three approximations to $u(x)$, say $\{u_i^h\}$, $\left\{u_i^{h/2}\right\}$ and $\left\{u_i^{h/4}\right\}$, we can make good estimates of τ_1, q_1 and $u(x_i)$ at all points x_i for which elements of all three grid functions are available. This merely involves solving the following system of three equations for τ_1, q_1 and $u(x_i)$:

$$\begin{aligned} u_i^h &= u(x_i) + \tau_1 h^{q_1} \ , \\ u_i^{h/2} &= u(x_i) + 2^{-q_1}\tau_1 h^{q_1} \ , \\ u_i^{h/4} &= u(x_i) + 4^{-q_1}\tau_1 h^{q_1} \ . \end{aligned}$$

Now recall that u_i^h, $u_i^{h/2}$, $u_i^{h/4}$ and h are all known values. Thus, we can substract the second equation from the first, and the third from the second, to obtain

$$u_i^h - u_i^{h/2} = \left(1 - 2^{-q_1}\right)\tau_1 h^{q_1} \ , \tag{1.17}$$

and

$$u_i^{h/2} - u_i^{h/4} = 2^{-q_1}\left(1 - 2^{-q_1}\right)\tau_1 h^{q_1} \ . \tag{1.18}$$

Then the ratio of these is

$$\frac{u_i^h - u_i^{h/2}}{u_i^{h/2} - u_i^{h/4}} = 2^{q_1} \ , \tag{1.19}$$

which is equivalent to the result (1.16) obtained above (now with $r = 2$) using true error. Again note that q_1 should be known, theoretically; but in practice, due either to algorithm/coding errors or simply to use of step sizes that are too large, the theoretical value of q_1 may not be attained at all (or possibly at any!) grid points x_i.

This motivates us to solve Eq. (1.19) for the actual value of q_1:

$$q_1 = \frac{\log\left[\dfrac{u_i^h - u_i^{h/2}}{u_i^{h/2} - u_i^{h/4}}\right]}{\log 2} \ . \tag{1.20}$$

Then from Eq. (1.17) we obtain

$$\tau_1 = \frac{u_i^h - u_i^{h/2}}{\left(1 - 2^{-q_i}\right)h^{q_1}} \ . \tag{1.21}$$

Finally, we can now produce an even more accurate estimate of the exact solution (equivalent to Richardson extrapolation) from any of the original equations; *e.g.*,

$$u(x_i) = u_i^h - \tau_1 h^{q_1} \ . \tag{1.22}$$

In most practical situations we are more interested in simply determining whether the grid functions converge and, if so, whether convergence is at the expected theoretical rate. To do this it is usually sufficient to replace $u(x_i)$ in the original expansions with a value u_i computed on a grid much finer than any of the test grids, or a Richardson extrapolated value obtained from the test grids, say u_i^*. The latter is clearly more practical, and for sufficiently small h it leads to

$$\tilde{e}_i^h = u_i^* - u_i^h \cong -\tau_1 h^{q_1} \ ,$$

where $\tilde{e}_i^h$ denotes an approximation to e_i^h of Eq. (1.14) since u_i^* (the result of Richardson extrapolation) is only an approximation of the exact function $u(x_i)$. Similarly,

$$\tilde{e}_i^{h/2} = u_i^* - u_i^{h/2} \cong -2^{-q_1}\tau_1 h^{q_1} \ ,$$

and the ratio of these errors is

$$\frac{\tilde{e}_i^h}{\tilde{e}_i^{h/2}} \cong \frac{u_i^* - u_i^h}{u_i^* - u_i^{h/2}} = 2^{q_1}. \tag{1.23}$$

Yet another alternative (and in general, probably the best one when only grid function convergence is the concern) is simply to use Eq. (1.19), *i.e.*, employ a Cauchy convergence test. As noted above we generally know the theoretical value of q_1. Thus, the left side (obtained from numerical computation) can be compared with the right side (theoretical). Even when q_1 is not known we can gain qualitative information from the left-hand side alone. In particular, it is clear that the right-hand side is always greater than unity. Hence, this should be true of the left-hand side. If the equality in the appropriate one of (1.19) or (1.23) is not at least approximately satisfied, the first thing to do is reduce h, and repeat the analysis. If this does not lead to closer agreement between left- and right-hand sides in these formulas, it is fairly certain that there are errors in the algorithm and/or its implementation.

We note that the above procedures can be carried out for arbitrary sequences of grid spacings, and for multi-dimensional grid functions. In both cases the required formulas are more involved, but in fact generally occur in the PDE context. Finally, we must recognize that e_i^h (or $\tilde{e}_i^h$) is error at a single grid point. In most practical problems it is more appropriate to employ an error norm computed with the entire solution vector. Then (1.16), for example, would be replaced with

$$\frac{\|e^h\|}{\|e^{h/2}\|} \cong \frac{\|u^h - u^{h/2}\|}{\|u^{h/2} - u^{h/4}\|} = 2^{q_1}\,, \tag{1.24}$$

for some norm $\|\cdot\|$, say, the vector 2-norm.

1.2 Classifications of Partial Differential Equations

There are many ways in which PDEs can be classified. Here, we consider what probably are the two most important: classification by type for linear equations, and definition of the forms of nonlinearity that arise for those which are not linear.

1.2.1 Equation type

The most general form of linear second-order partial differential equations, when restricted to two independent variables and constant coefficients, is

$$au_{xx} + bu_{xy} + cu_{yy} + du_x + eu_y + fu = g(x,y)\,, \tag{1.25}$$

where g is a known forcing function; $a, b, c, \ldots$, are given constants, and subscripts denote partial differentiation. In the homogeneous case, *i.e.*, $g \equiv 0$, this form is reminiscent of the *general quadratic form* from high school analytic geometry:

$$ax^2 + bxy + cy^2 + dx + ey + f = 0\,. \tag{1.26}$$

Equation (1.26) is said to be an ellipse, a parabola or a hyperbola according as the *discriminant* $b^2 - 4ac$ is less than, equal to, or greater than zero. This same classification—*elliptic*, *parabolic*, or *hyperbolic*—is employed for the PDE (1.25), independent of the nature of $g(x,y)$. In fact, it is clear that the classification of linear PDEs depends only on the coefficients of the highest-order derivatives. This grouping of terms,

$$au_{xx} + bu_{xy} + cu_{yy}\,,$$

is called the *principal part* of the differential operator in (1.25), *i.e.*, the collection of highest-order derivative terms with respect to each independent variable; and this notion can be extended in a natural way to more complicated operators. Thus, the type of a linear equation is completely determined by its principal part.

It should be mentioned that if the coefficients of (1.25) are permitted to vary with x and y, its type may change from point to point within the solution domain. This can pose significant difficulties, both analytical and numerical. We will not specifically deal with these in the current lectures.

We next note that corresponding to each of the three types of equations there is a unique *canonical form* to which (1.25) can always be reduced. We shall not present the details of the transformations needed to achieve these reductions, as they can be found in many standard texts on elementary PDEs (*e.g.*, Berg and MacGregor [5]). On the other hand, it is important to be aware of the possibility of simplifying (1.25), since this may also simplify the numerical analysis required to construct a solution algorithm.

Elliptic. It can be shown when $b^2 - 4ac < 0$, the elliptic case, that (1.25) collapses to the form

$$u_{xx} + u_{yy} + Au = g(x, y)\,, \tag{1.27}$$

with $A = 0,\ \pm 1$. When $A = 0$ we obtain *Poisson's equation*, or *Laplace's equation* in the case $g \equiv 0$; otherwise, the result is usually termed the *Helmholtz equation.*

Parabolic. For the parabolic case, $b^2 - 4ac = 0$, we have

$$u_x - u_{yy} = g(x, y)\,, \tag{1.28}$$

which is the *heat equation*, or the diffusion equation. We remark that $b^2 - 4ac = 0$ can also imply a "degenerate" form which is only an ordinary differential equation (ODE). We will not treat this case in the present lectures.

Hyperbolic. For the hyperbolic case, $b^2 - 4ac > 0$, Eq. (1.25) can always be transformed to

$$u_{xx} - u_{yy} + Bu = g(x, y)\,, \tag{1.29}$$

where $B = 0$ or 1. If $B = 0$, we have the *wave equation*, and when $B = 1$ we obtain the linear *Klein–Gordon equation.*

Finally, we note that determination of equation type in dimensions greater than two requires a different approach. The details are rather technical but basically involve the fact that elliptic and hyperbolic operators have definitions that are independent of dimension, and usual parabolic operators can then be identified as a combination of an elliptic "spatial" operator and a first-order evolution operator.

1.2.2 Form of nonlinearity

Nonlinear differential equations can take on a number of different forms. Here, we will treat the ones that occur most often in practice. We begin by recalling the definition of a *linear operator* to provide a point of reference.

Definition 1.10 *Let S be a vector space defined on the real numbers $\mathbb{R}$ (or the complex numbers $\mathbb{C}$), and let L be an operator (or transformation) whose domain is S. Suppose for any $u, v \in S$ and $a, b \in \mathbb{R}$ (or $\mathbb{C}$) we have*

$$L(au + bv) = aLu + bLv. \tag{1.30}$$

Then L is said to be a linear operator.

The reader is encouraged to show that all of the differential operators of the previous subsection are linear.

The forms of nonlinearity we consider here are: semilinear, quasilinear and fully nonlinear. These forms are generally the same for both steady-state and evolution equations.

Semilinear. *Semilinear* PDEs represent the simplest type of nonlinearity. They are linear in all terms except those of zeroth order; hence, they can be expressed as $Lu + F(u) = s(\boldsymbol{x})$, where L is a linear (but possibly variable-coefficient) operator, in general containing both spatial derivatives and evolution terms, and F is a nonlinear function of u only; *i.e.*, there are no nonlinearities in terms containing derivatives of

u. The right-hand side term, s, is a prescribed forcing function. A typical example of a semilinear PDE would be a "reaction-diffusion" equation such as

$$u_t - D\Delta u - e^{-C/u} = 0\,, \tag{1.31}$$

with Δ denoting the Laplacian (*e.g.*, $\Delta \equiv \partial^2/\partial x^2 + \partial^2/\partial y^2$ in 2-D Cartesian coordinates); D is a constant diffusion coefficient, and C is a constant "reaction energy."

Quasilinear. PDEs are called *quasilinear* when terms containing their lowest derivatives appear as a product of the derivative and an undifferentiated factor of the dependent variable. (Note, of course, that if the equations are only first order, then their lowest derivatives are also their highest ones.) "Advective-diffusive" equations such as the Navier–Stokes equations are typical examples:

$$u_t + uu_x + vu_y - \nu\Delta u = -p_x \tag{1.32}$$

for the x-momentum equation. Observe that all terms are linear except the advective (also called convective) terms $uu_x + vu_y$. Clearly, the first of these is nonlinear as the reader may check by using the preceding definition of linearity—it fails to satisfy this definition; in particular, it is quasilinear. Moreover, we can write this in so-called "conservation-law form" as $(u^2)_x$ to obtain an alternate expression for this quasilinear form. But we emphasize that, in general, it is the first form, shown in Eq. (1.32) that represents the definition of quasilinearity. We also remark that the second advection term is also formally quasilinear if $v = v(u)$—as would be the case for the incompressible Navier–Stokes equations via the divergence-free condition. But such terms are often classified as *bilinear* because u, itself, appears explicitly in only one factor.

Fully Nonlinear. The *fully-nonlinear*, or just *nonlinear*, case is characterized by nonlinearities even in the highest derivatives of the differential operator. An example might be a heat equation containing a temperature-dependent thermal diffusivity, for example, in an advective-diffusive equation of the form

$$T_t + uT_x + vT_y - \nabla\cdot\nabla(D(T)T) = 0\,. \tag{1.33}$$

Another, often much more complicated, instance of this is use of "eddy viscosity" in Reynolds-averaged turbulence models of the Navier–Stokes equations. We comment that, in general, from a pure mathematics perspective, it is the fully-nonlinear PDEs which are least understood, as might be expected.

1.3 Well Posedness of PDE Problems

Before proceeding to introduce numerical methods for solving each of the three main classes of problems it is worthwhile to give some consideration to the question of under what circumstances these equations do, or do not, have solutions. This is a part of the mathematical concept of *well posedness.* There are many different specific definitions; the one given here is commonly used in elementary settings.

Definition 1.11 *A problem consisting of a partial differential equation and boundary and/or initial conditions is said to be well posed in the sense of Hadamard if it satisfies the following conditions:*

i) a solution exists;

ii) the solution is unique;

iii) the solution depends continuously on given data.

The well-posedness property is crucial in solving problems by numerical methods because essentially all numerical algorithms embody the tacit assumption that problems to which they apply are well posed. Consequently, a method is not likely to work correctly on an ill-posed (*i.e.*, a not well-posed) problem.

The result may be failure to obtain a solution; but a more serious outcome may be generation of numbers that have no association with reality in any sense. It behooves the user of numerical methods to sufficiently understand the mathematics of any considered problem to be aware of the possible difficulties—and symptoms of these difficulties—associated with problems that are not well posed.

We close this brief discussion of well posedness by describing a particular problem that is not well posed. This is the so-called "backward" heat equation problem. It arises in geophysical studies in which it is desired to predict the temperature distribution within the Earth at some earlier geological time by integrating backward from the (presumed-known) temperature distribution of the present. To demonstrate the difficulties that arise we consider a simple one-dimensional initial-value problem for the heat equation:

$$u_t = \kappa u_{xx} , \qquad x \in (-\infty, \infty) , \quad t \in [-T, 0) ,$$

with

$$u(x, 0) = f(x) .$$

Formally, the exact solution is

$$u(x,t) = \frac{1}{\sqrt{4\pi\kappa t}} \int_{-\infty}^{\infty} f(\xi) e^{-\frac{(x-\xi)^2}{4\kappa t}} d\xi , \tag{1.34}$$

the derivation of which (see, *e.g.*, Berg and MacGregor [5]) imposes no specific restrictions on the sign of t. But we immediately see that if $t < 0$, $u(x,t)$, if it exists at all, is imaginary (since κ, the thermal diffusivity, is always greater than zero). In fact, unless f decays to zero faster than exponentially at $\pm\infty$, there is no solution because the integral in (1.34) does not exist. It turns out that behavior of heat equation solutions places restrictions on the form of difference approximations that can be used to numerically solve the equation. In particular, schemes that are multilevel in time with a backward (in time) contribution can fail. An example of this is the well-known second-order centered (in time) method due to Richardson (see [1]); it is unconditionally unstable.

1.4 Discretization and Gridding of PDE Problems

Although in the sequel we will consider only basic finite-difference methods for approximating solutions of partial differential equations, in the present section we shall provide brief discussions of several of the most widely-used discretization techniques. We note at the outset that temporal discretizations are almost always either finite-difference or quadrature based, but many different methods, and combinations of these, may be employed for spatial approximation. In a second subsection we present a brief overview of techniques that are widely used to generate the discrete grids on which numerical approximations to PDEs are solved.

1.4.1 Discretization techniques

Discretization of any PDE consists of converting an infinite-dimensional solution and the differential operators which act on it to a finite-dimensional one, typically (but not always) in terms of grid functions and difference operators to which simple algebraic techniques can be applied to produce approximate solutions to the PDE. Figure 1.2 depicts the main features of what are the most well-known classes of such methods: *i*) finite difference, *ii*), finite element and *iii*) spectral.

Finite Difference. As we have described in considerable detail in [1], finite-difference methods are constructed by first "gridding" the solution domain as indicated in Fig. 1.2(a), and then deriving systems of algebraic equations for grid-point values which serve as approximations to the true solution of the PDE at the discrete set of points defined (typically) by intersections of the grid lines. We remark that the regular "structured" grid shown here is not the only possibility. But we will not treat finite-difference approximations on "unstructured" grids in the present lectures.

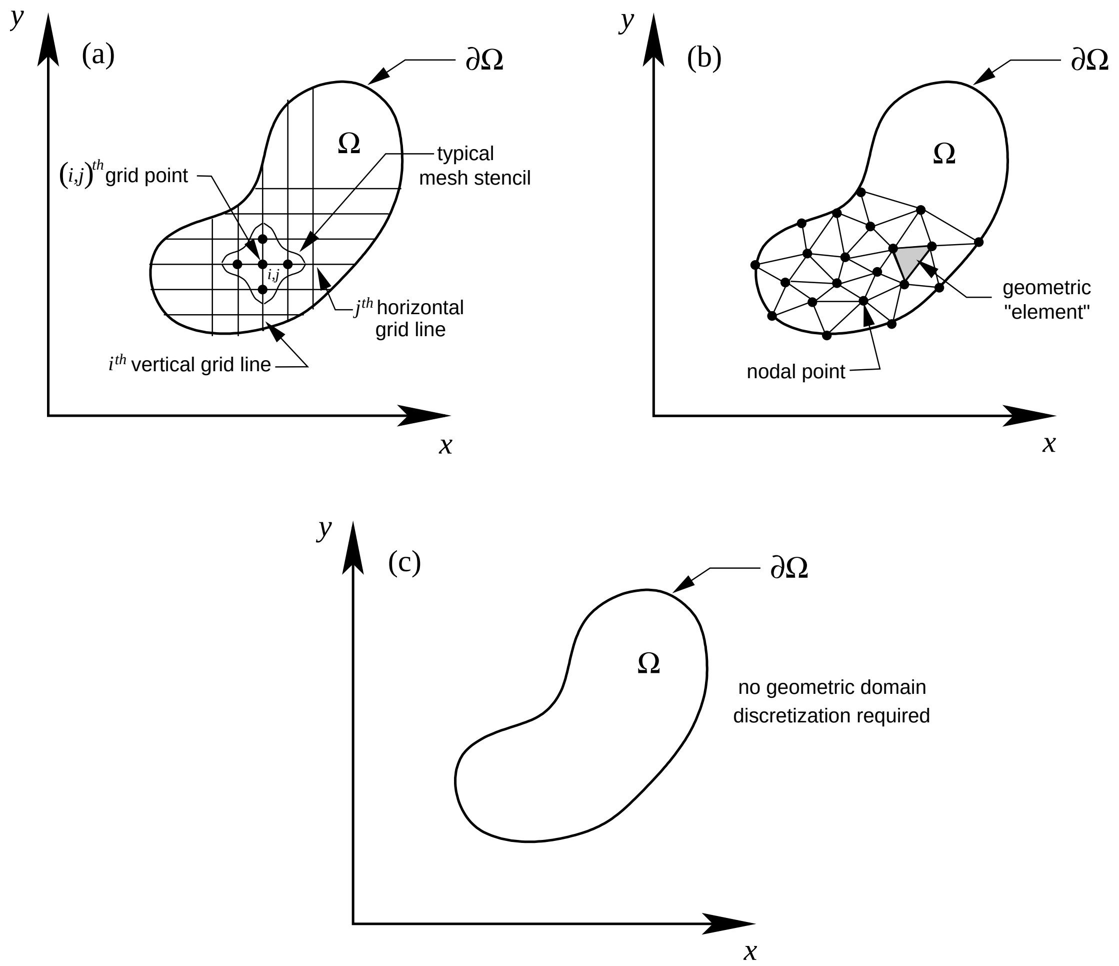

Figure 1.2: Methods for spatial discretization of partial differential equations; (a) finite difference, (b) finite element and (c) spectral.

Within this framework we will often, in the sequel, use the following difference formula notations introduced in [1] for first-order partial derivative approximation in 2-D.

$$D_{+,x}(h_x)u_{i,j} \equiv \frac{u_{i+1,j}-u_{i,j}}{h_x} = \left.\frac{\partial u}{\partial x}\right|_{(x_i,y_j)} + \mathcal{O}(h_x)\,, \qquad \text{(forward)} \tag{1.35a}$$

$$D_{-,x}(h_x)u_{i,j} \equiv \frac{u_{i,j}-u_{i-1,j}}{h_x} = \left.\frac{\partial u}{\partial x}\right|_{(x_i,y_j)} + \mathcal{O}(h_x)\,, \qquad \text{(backward)} \tag{1.35b}$$

$$D_{0,x}(h_x)u_{i,j} \equiv \frac{u_{i+1,j}-u_{i-1,j}}{2h_x} = \left.\frac{\partial u}{\partial x}\right|_{(x_i,y_j)} + \mathcal{O}(h_x^2)\,. \qquad \text{(centered)} \tag{1.35c}$$

Similarly, for second-order partial derivatives we will employ the second centered difference

$$D_{0,x}^2(h_x)u_{i,j} = \frac{u_{i-1,j}-2u_{i,j}+u_{i+1,j}}{h_x^2} = \left.\frac{\partial^2 u}{\partial x^2}\right|_{(x_i,y_j)} + \mathcal{O}(h_x^2)\,, \tag{1.36}$$

where we note, as done in [1], that the notation is formal, and in fact, we would actually construct this approximation either with a one-half grid spacing, or by the composition of first-order forward and backward difference operators, *e.g.*, $D^2_{0,x}(h_x) = D_{+,x}(h_x)D_{-,x}(h_x)$. Finally, we will typically suppress the h_x notation in these operators when the interpretation of corresponding expressions is clear.

Finite Element. Finite-element methods (FEMs) are somewhat similar to finite-difference methods, although they are typically more elaborate, often somewhat more accurate, and essentially always more difficult to implement and less efficient. As can be seen from Fig. 1.2(b) the problem domain is subdivided into small regions, often of triangular (or, in 3-D, tetrahedral) shape. On each of these subregions a polynomial is used to approximate the solution, and various degrees of smoothness of the approximate solution are achieved by requiring constructions that guarantee continuity of a prescribed number of derivatives across element boundaries. This approximation plus the subregion on which it applies is the "element." It should be remarked that the mathematics of FEMs is highly developed and is based on variational principles and weak solutions (see, *e.g.*, Strang and Fix [7]), in contrast to the Taylor-expansion foundations of finite-difference approaches. In this sense it bears some similarity to Galerkin procedures.

Spectral. Figure 1.2(c) displays the situation for spectral methods. One should observe that in this case there is no grid or discrete point set. Instead of employing what are essentially local polynomial approximations as done in finite-difference and finite-element methods, assumed forms of the solution function are constructed as (generalized) Fourier representations that are valid globally over the whole solution domain. In this case, the unknowns to be calculated consist of a finite number of Fourier coefficients leading to what amounts to a projection of the true solution onto a finite-dimensional space. In modern terminology, these are examples of so-called "grid-free" methods.

There are numerous other discretization methods that have attained fairly wide acceptance in certain areas. In computational fluid dynamics (CFD) finite-volume techniques are often used. These can be viewed as lying between finite-difference and finite-element methods in structure, but in fact they are essentially identical to finite-difference methods despite the rather different viewpoint (integral formulation of governing equations) employed for their construction. Other less often-used approaches deserving of at least mention are spectral-element and boundary-element methods. As their names suggest, these are also related to finite-element methods, and particularly in the latter case are applicable mainly for only a very restricted class of problems. Beyond these are a number of different "pseudo-spectral" methods that are similar to spectral methods, but for which the basis functions employed are not necessarily eigenfunctions of the principal part of the differential operator(s) of the problem being considered. In addition to all these individual techniques, various calculations have been performed using two or more of these methods in combination. Particular examples of this include finite-difference/spectral and finite-element/boundary-element methods. Discussion of details of such schemes is beyond the intended scope of the present lectures, and from this point onward our attention will be focused on basic finite-difference methods that have been widely used because of their inherent generality and relative ease of application to PDE approximation.

1.4.2 Gridding methods

As implied above, there are two main types of gridding techniques in wide use, corresponding to structured and unstructured gridding—with many different variations available, especially for the former of these. Here, we will briefly outline some of the general features of these approaches, and leave details (including treatment of various modifications and combinations of these) to Chap. 5.

Structured Grids. Use of structure grids involves labeling of grid points in such a way that if the indices of any one point are known, then the indices of all points within the grid can be easily determined. For many years this was the preferred (in fact, essentially only) approach utilized. It leads to very efficient, readily parallelized numerical algorithms and straightforward post processing. But generation of structured grids for complicated problem domains, as arise in many engineering applications, is very time consuming in terms of human time—and thus, very expensive.

Unstructured Grids. Human time required for grid generation has been dramatically reduced with use of unstructured grids, but this represents their only advantage. Such grids produce solutions that are far less accurate, and the required solution algorithms are less efficient, than is true for a structured grid applied to the same problem. This arises from the fact that the grid points comprising an unstructured grid can be ordered in many different ways, and knowing indexing for any one point provides no information regarding the indexing of any other points—often, even of nearest neighbors. In particular, indexing of points is handled via "pointers" which are vectors of indices. Each point possesses a unique pointer, but there is no implied canonical ordering of these pointers. This leads to numerous difficulties in essentially all aspects of the solution process, as we will describe in more detail in Chap. 5.

The interested reader is referred to Thompson *et al.* [6] for comprehensive treatments of both structured and unstructured gridding techniques, generally referred to as "grid generation."

1.5 Summary

In this introductory chapter we have presented some of the most basic material needed for a study of numerical solution of partial differential equations. Some of this has been rather abstract—the notion of Hilbert spaces, the Cauchy–Schwarz inequality, well posedness; and some has been more directly applicable despite analytical underpinnings—*e.g.*, computational formulas for norms and methods for testing grid-function convergence. Finally, in addition we have provided a brief overview of topics that are specifically numerical analytic, namely, general classes of discretization techniques and approaches to grid structure.

This set of topics is intended to provide a brief review of elementary information needed as the starting point for the lectures that follow in subsequent chapters. With this material behind us, we can now state the general PDE problem whose solution will be made possible via the various numerical algorithms to be considered in Chaps. 2–5: let

$$\boldsymbol{\mathcal{P}}(\boldsymbol{u}) = \boldsymbol{S}(\boldsymbol{x},t)\,, \quad \boldsymbol{x} \in \Omega \subseteq \mathbb{R}^d\,, \quad d = 1,2,3\,, \quad t \in (t_0, t_f]\,, \tag{1.37}$$

with initial data (if appropriate)

$$\boldsymbol{u}(\boldsymbol{x},t_0) = \boldsymbol{u}_0(\boldsymbol{x})\,, \qquad \boldsymbol{x} \in \Omega\,,$$

(and any additional required derivative conditions, as needed for higher-order equations), and boundary conditions (if appropriate)

$$\boldsymbol{\mathcal{B}}\boldsymbol{u}(\boldsymbol{x},t) = \boldsymbol{R}(\boldsymbol{x},t)\,, \qquad \boldsymbol{x} \in \partial\Omega\,,\ \ t \in (t_0,t_f]\,.$$

Here, $\mathcal{P}$ is a general vector partial differential operator with components of the form

$$\mathcal{P}_i = \mathcal{L}_i + \mathcal{N}_i\,, \qquad i = 1,2,\ldots,N_e\,, \tag{1.38}$$

where $\mathcal{L}_i$ and $\mathcal{N}_i$ are, respectively, the linear and nonlinear parts if the i^{th} component of this operator, with $\boldsymbol{\mathcal{P}} = (\mathcal{P}_1, \mathcal{P}_2 \ldots, \mathcal{P}_{N_e})^T$, and N_e is the number of PDEs in the system. The solution vector in (1.37) is $\boldsymbol{u}(\boldsymbol{x},t) = (u_1(\boldsymbol{x},t), u_2(\boldsymbol{x},t), \ldots, u_{N_e}(\boldsymbol{x},t))^T$. Finally, Ω, the problem domain and $\partial\Omega$, its boundary, may be of essentially any shape and complexity within the confines of requirements for well posedness. Moreover, different instances of the $\mathcal{P}_i$s may hold only on certain subsets of Ω, and not on others.

We remark that problems of this level of complexity arise throughout the physical, biological and engineering sciences, and with the rapid advance of computing hardware capabilities we are able to solve an ever-expanding array of these. The result is significantly improved scientific understanding of the universe around us, and optimal solutions to many engineering problems associated with everything from industrial processes to a myriad of consumer products.

Chapter 2

Numerical Solution of Elliptic Equations

In this chapter we will study the solution of linear elliptic partial differential equations (PDEs) via numerical techniques. These equations typically represent steady-state physical situations, and in two space dimensions (2D) assume the general form

$$\begin{aligned}(a_1(x,y)u_x)_x + (a_2(x,y)u_y)_y + (a_3(x,y)u_x)_y + (a_4(x,y)u_y)_x \\ + (a_5(x,y)u)_x + (a_6(x,y)u)_y + a_7 u(x,y) = f(x,y) \qquad (2.1)\end{aligned}$$

on a domain $\Omega \subseteq \mathbb{R}^2$ with appropriate boundary conditions, *e.g.*, combinations of Dirichlet, Neumann and Robin) prescribed on $\partial\Omega$. Here, subscripts denote partial differentiation; *e.g.*, $u_x = \partial u/\partial x$. It will be assumed that the coefficients of (2.1) are such as to render the PDE elliptic, uniformly in $\overline{\Omega}$ (or in $\Omega \setminus \partial\Omega$, as appropriate in the context of prescribed boundary conditions).

Throughout these lectures we will employ straightforward second-order centered finite-difference approximations of derivatives (with an occasional exception), primarily for simplicity and ease of presentation. Our emphasis in the present studies will be on solution procedures, and while it is, of course, not true that these will be completely independent of specifics of discretization, they will nevertheless prove to have fairly wide applicability, in general. Applying such discretization to Eq. (2.1) results in a system of algebraic equations,

$$\begin{aligned}A^{(1)}_{i,j}u_{i-1,j-1} + A^{(2)}_{i,j}u_{i-1,j} + A^{(3)}_{i,j}u_{i-1,j+1} + A^{(4)}_{i,j}u_{i,j-1} + A^{(5)}_{i,j}u_{i,j} \\ + A^{(6)}_{i,j}u_{i,j+1} + A^{(7)}_{i,j}u_{i+1,j-1} + A^{(8)}_{i,j}u_{i+1,j} + A^{(9)}_{i,j}u_{i+1,j+1} = f_{i,j}\,, \qquad (2.2) \\ i = 1,\ldots,N_x\,, \quad j = 1,\ldots,N_y\,.\end{aligned}$$

We note that boundary conditions are assumed to have been included in this system of equations, so this corresponds to a solution on a $N_x \times N_y$-point grid, including boundary points, as depicted in Fig. 2.1. We have also indicated in this figure the *mesh star* corresponding to Eq. (2.2).

We should comment here that while we will essentially always be concerned with 2-D problems in these lectures, this is done merely for simplicity of presentation. Nearly all of the numerical algorithms to be considered apply equally well in 3D, and this can be assumed unless we specifically note otherwise. Moreover, although as already mentioned, we will employ finite-difference discretizations, most of the solution methods to be discussed also apply for finite-element methods (FEMs).

The system of linear algebraic equations corresponding to Eqs. (2.2) is sparse and banded, as shown in Fig. 2.2. Part (a) of this figure corresponds to a second-order centered discretization of a 2-D Laplace operator, $u_{xx} + u_{yy}$, while part (b) is associated with the more general operator of Eq. (2.1). We will typically employ the concise notation

$$Au = b \qquad (2.3)$$

to represent such systems, and the focus of this chapter is study of methods for efficiently solving Eq. (2.3) on a digital computer.

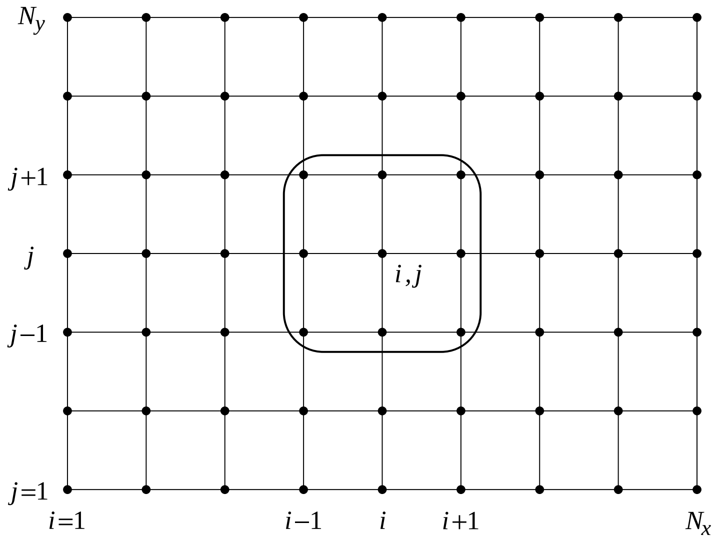

Figure 2.1: $N_x \times N_y$–point grid and mesh star for discretizations of Eq. (2.1).

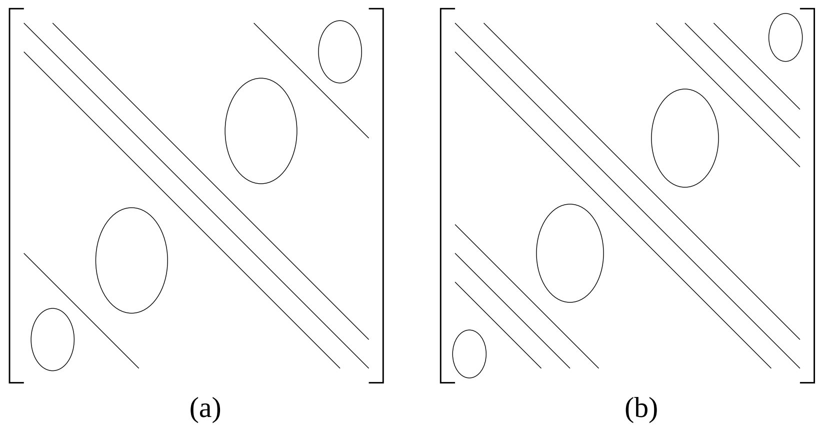

Figure 2.2: Sparse, banded matrices arising from finite-difference discretizations of elliptic operators: (a) 5-point discrete Laplacian; (b) 9-point general discrete elliptic operator.

In the following sections we will first provide a brief background discussion to motivate study of the class of methods to be considered here, namely, *iterative techniques*, and a short review of the theory of linear fixed-point iteration. This will be followed by a section devoted to the classical, but still widely-used approach known as *successive overrelaxation* (SOR). We will then present a summary treatment of a once very popular method known as the *alternating direction implicit* (ADI) method. In Sec. 2.4 we will consider *incomplete LU decomposition* (ILU) schemes, and we follow this in Sec. 2.5 with a brief discussion of what is termed *preconditioning*; Sec. 2.6 contains an introduction to the conjugate-gradient (CG) method. This will complete our description of older, classical techniques. The two final sections of the chapter will contain introductions to the two modern and extremely popular approaches for solving sparse linear systems—*multigrid* (MG) and *domain decomposition methods* (DDMs).

2.1 Background

In this section we will first present an overview of methods available for solving Eq. (2.3) and, in particular, give estimates of the total arithmetic required for each approach. We will conclude from this that now, and in the immediate future, iterative (as opposed to direct) methods are to be preferred. We will then present a short theoretical background related to such methods, in general. This will provide a natural introduction to terminology and notation to be used throughout the chapter, and it will yield some key theoretical results.

2.1.1 Iterative solution of linear systems—an overview

Linear systems of the form (2.3), with A a nonsingular matrix, can be solved in a great variety of ways. When A is not sparse, direct Gaussian elimination is usually the preferred approach. But this requires $\mathcal{O}(N^2)$ words of storage and $\mathcal{O}(N^3)$ floating-point arithmetic operations for a $N \times N$ matrix A and N-vectors b and u, with $N \equiv N_xN_y$. To put this in perspective we note that for systems arising in the manner of concern in these lectures (*viz.*, as discretizations of PDEs) N can easily be $\mathcal{O}(10^6)$, implying terabyte storage requirements and $\mathcal{O}(10^{18})$ arithmetic operations if a direct method is used. If we assume availability of teraflops performance, then $\mathcal{O}(10^6)$ seconds of CPU time will be required to obtain a solution, *i.e.*, $\sim$ 300 hours. Such requirements are clearly unacceptable in most situations, so alternative approaches must be sought.

It is essentially universally accepted that iterative techniques provide a much more efficient approach to the solution of large, sparse banded linear systems that are not *compactly banded*. While direct methods provide *exact* solutions (to machine precision, modulo rounding errors) with an exactly predetermined amount of arithmetic, iterative methods present the advantages of requiring significantly less storage (typically no more than $\mathcal{O}(N)$ words), permitting the analyst to prescribe the level of accuracy of the computed solution, and achieving a solution in no more than $\mathcal{O}(N^2)$ total arithmetic operations. We should comment that the ability to set the level of solution accuracy is especially important in the context of algebraic equations arising as discretizations of differential equations: there is an inherent truncation error in the equations themselves, so it is not generally reasonable to solve them to levels of accuracy that far exceed this. Beyond this is the fact that in physical problems, especially those arising in practical engineering contexts, data provided with the problem—initial and/or boundary conditions and prescribed forcing functions—may be quite inaccurate, often known to only a couple significant figures, at best. Hence, highly accurate solutions to the discrete equations are not appropriate.

Through the years there have been dozens (maybe hundreds) of different iterative techniques proposed. Here we will mention a few of the more widely-used ones and provide a qualitative comparison of them. It is well known that performance of essentially any of the methods treated in these lectures is problem dependent, and for that reason it is useful to distinguish some problem classes in order to facilitate more meaningful comparisons. We will consider the following three: *i*) constant-coefficient operators on a rectangular domain, *ii*) operators with smooth coefficients on a rectangular domain and *iii*) completely general operators (*e.g.*, with nonsmooth coefficients) on general domains.

As we have indicated in Fig. 2.3 below, if one is willing to expend $\mathcal{O}(N^2)$ arithmetic, a considerable number of methods can be used to solve even the most general problems. In fact, direct methods can be constructed to solve the sparse systems we consider here with this amount of arithmetic. The *capacitance matrix* methods of Buzbee [8] and many similar approaches based on *cyclic*, or nested, *reduction* (or dissection) discussed in Duff *et al.* [9] fall into this category. But $\mathcal{O}(N^2)$ is typically an unacceptably high amount of arithmetic.

There are also numerous techniques that can obtain solutions to fairly general 2-D problems in $\mathcal{O}(N^{1.5})$ arithmetic operations. These include ADI methods, some forms of ILU and various SORs. This, however, is still a burdensome amount of arithmetic, especially if the elliptic problem must be solved frequently in the course of solving a more involved overall problem, as often happens in computational fluid dynamics (CFD) and computational electromagnetics (see, *e.g.*, Fletcher [10], Umashankar and Taflove [11], respectively).

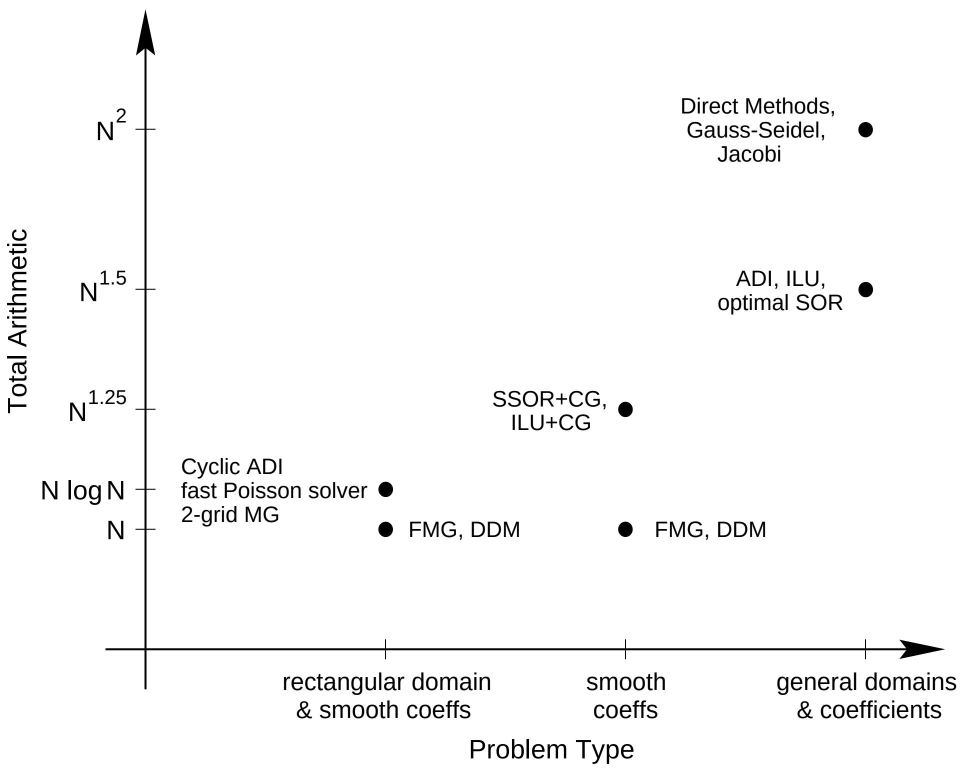

Figure 2.3: Qualitative comparison of required arithmetic for various iterative methods for 2-D elliptic problems.

If one restricts attention to problems whose differential operators possess smooth coefficients then it is possible to employ combinations of methods to reduce the required arithmetic to $\mathcal{O}(N^{1.25})$. Examples of this include use of symmetric SOR (SSOR) or ILU as *preconditioners* for a conjugate-gradient method, or some form of *Krylov-subspace* based approach (see Saad [12]). In fact, it is possible to construct multigrid and domain-decomposition techniques that can reduce required arithmetic to nearly the optimal $\mathcal{O}(N)$ in this case although, as will be evident in the sequel, these approaches are quite elaborate.

Finally, for the simple case of a rectangular domain and constant coefficients in the differential operators, cyclic ADI, *fast Poisson solvers* (which are not iterative, but of very limited applicability), and two-grid MG methods can produce solutions in $\mathcal{O}(N \log N)$ total arithmetic, and full multigrid (FMG) and DDMs can, of course, lead to solutions in $\mathcal{O}(N)$ arithmetic. But we must comment that the two conditions, rectangular domain and constant coefficients, are all but mutually exclusive in practical problems. Thus, this summary figure, similar to one first presented by Axelsson [13], indicates that much is involved in selecting a suitable solution method for any specific elliptic boundary-value problem. But we can expect that an iterative method of some type will usually be the best choice. In the following sections we will provide details of analyzing and implementing many of the abovementioned techniques.

2.1.2 Basic theory of linear iterative methods

In this section we will present a brief overview of fixed-point iteration as applied to the solution of linear systems. This will provide an opportunity to introduce terminology and notation to be used throughout the lectures of Chap. 2, and in addition to introduce some theoretical tools that are not only useful for analysis but also adaptable to numerical calculation. We begin by recalling Eq. (2.3),

$$Au = b\,,$$

and note that iterative methods for solving this system of linear equations can essentially always be expressed in the form

$$u^{(n+1)} = Gu^{(n)} + k\,. \tag{2.4}$$

In this expression (n) denotes an *iteration counter*, and G is the *iteration matrix*; it is related to the system matrix A by

$$G = I - Q^{-1}A\,, \tag{2.5}$$

where I is the identity matrix, and Q is generally called the *splitting matrix*. The vector k is constructed from the original right-hand side vector b and the inverse of the splitting matrix, as carried out below.

It is worthwhile to consider a well-known concrete example of (2.4) to motivate this terminology. Recall from elementary numerical analysis (see, *e.g.*, Stoer and Bulirsch [14]) that Jacobi iteration can be constructed as follows. First, decompose the matrix A as

$$A = D - L - U\,, \tag{2.6}$$

where D is the diagonal of A, and L and U are negatives of the lower and upper, respectively, triangles of A. Now substitute (2.6) into (2.3) to obtain

$$(D - L - U)u = b\,,$$

or

$$Du = (L + U)u + b\,. \tag{2.7}$$

In deriving Eq. (2.7) we have "split" the matrix A to isolate the trivially invertible diagonal matrix on the left-hand side. We now introduce iteration counters and write (2.7) as

$$u^{(n+1)} = D^{-1}(L + U)u^{(n)} + D^{-1}b\,, \tag{2.8}$$

which clearly is in the form of (2.4). In particular, observe from (2.6) that

$$L + U = D - A\,,$$

so

$$D^{-1}(L + U) = I - D^{-1}A\,.$$

Thus, D is the splitting matrix, and Eq. (2.8) is in the form (2.4) with

$$G \equiv D^{-1}(L + U) = I - D^{-1}A\,, \qquad \text{and} \qquad k \equiv D^{-1}b\,. \tag{2.9}$$

We see from this that the splitting matrix can be readily identified as the inverse of the matrix multiplying the original right-hand side vector of the system in the definition of k.

We should next recall that convergence of fixed-point iterations generally requires something of the nature of guaranteeing existence of a *Lipschitz condition* with Lipschitz constant less than unity. The following theorem provides the version of this basic notion that is of use for the study of linear iterative methods.

Theorem 2.1 *A necessary and sufficient condition for convergence of the iterations* (2.4) *to the solution of Eq.* (2.3) *from any initial guess is*

$$\rho(G) < 1\,, \tag{2.10}$$

where

$$\rho(G) \equiv \max_{1 \le i \le N} |\lambda_i|\,, \qquad \lambda_i \in \sigma(G)\,, \tag{2.11}$$

is the <u>*spectral radius*</u> *of the iteration matrix G, and $\sigma(G)$ is notation for the* <u>*spectrum*</u> *(set of all eigenvalues) of G.*

We remark (without proof) that this basically follows from the *contraction mapping principle* and the fact that

$$\rho(G) \le \|G\| \tag{2.12}$$

for all norms, $\|\cdot\|$. We also note that convergence may occur even when $\rho(G) \le 1$ holds, but only for a restricted set of initial guesses. It should be clear that $\rho(G)$ corresponds to the *Lipschitz constant* mentioned above.

We next present some definitions that will be of use throughout these lectures. We first consider several definitions of error associated with the iterations (2.4), and then we define *convergence rate* to quantify how rapidly error is reduced.

Definition 2.1 *The* <u>*residual*</u> *after* n *iterations is*

$$r_n = b - Au^{(n)}\,. \tag{2.13}$$

Definition 2.2 *The* <u>*exact error*</u> *after* n *iterations is*

$$e_n = u - u^{(n)}\,. \tag{2.14}$$

Definition 2.3 *The* <u>*iteration error*</u> *after* n *iterations is*

$$d_n = u^{(n+1)} - u^{(n)}\,. \tag{2.15}$$

We note that it is easy to see that r_n and e_n are related by

$$Ae_n = r_n\,. \tag{2.16}$$

Also, it follows from the general linear iteration formula (2.4) and the definition (2.14) that

$$e_n = Ge_{n-1} = G^2 e_{n-2} = \cdots = G^n e_0\,, \tag{2.17}$$

and similarly for d_n, suggesting the connection of $\|e_n\|$ and $\|d_n\|$ with $\|G\|$. This leads to the following.

Definition 2.4 *The* <u>*average convergence rate*</u> *(over* n *iterations) for iterations of Eq.* (2.4) *is given by*

$$R_n(G) \equiv -\frac{1}{n}\log\|G^n\|\,. \tag{2.18}$$

This definition is motivated by the fact that from (2.17)

$$\log\|e_n\| - \log\|e_0\| \le \log\|G^n\|$$

for *compatible* matrix and vector norms, and thus $R_n(G)$ is a measure of the average (logarithmic) error reduction over n iterations.

A second important definition associated with convergence rate is the following.

Definition 2.5 *The* <u>*asymptotic convergence rate*</u> *for the iterations* (2.4) *is defined as*

$$R_\infty(G) \equiv -\log\rho(G)\,. \tag{2.19}$$

Some remarks are in order at this point. First, it is the asymptotic convergence rate that is more important in gauging performance of iterative methods when they are to be used to produce highly-accurate solutions. As we will discuss in more detail later, it is not uncommon for a fixed-point iteration to reduce the iteration error very quickly during the first few iterations, and then proceed very slowly thereafter, as depicted qualitatively in Fig. 2.4. Thus, the asymptotic convergence rate clearly provides a better measure of performance. It should also be clear from Eq. (2.18) that specific values of average

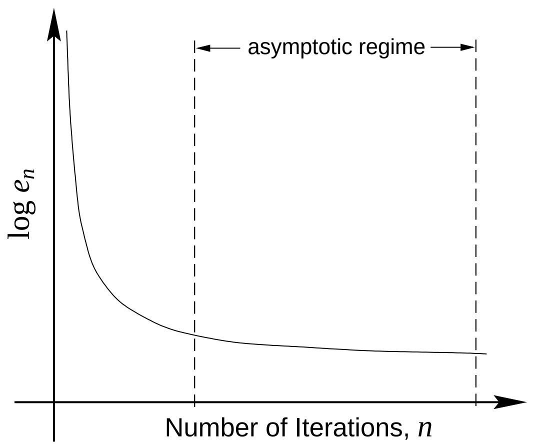

Figure 2.4: Qualitative representation of error reduction during linear fixed-point iterations.

convergence rate depend on the choice of norm, while values of asymptotic convergence rate, Eq. (2.19), depend only on the spectral radius of the iteration matrix and are thus unique. It is also important to recognize that these two measures of convergence rate are related in the expected way. Namely, it is shown by Young [15] that

$$R_\infty(G) = \lim_{n\to\infty} R_n(G)\,. \tag{2.20}$$

This relationship will be of use later in obtaining estimates of total arithmetic required by iterative methods. Finally, we note that when no confusion can arise, notation for the iteration matrix will be suppressed; furthermore, the ∞ subscript is often deleted from notation for asymptotic convergence rate. Hence, $R_\infty(G)$ becomes simply R.

We next observe that just as (2.16) provides a relationship between residual and the exact error, it can be shown (directly from the definitions) that the iteration error and exact error are related as

$$d_n = (I - G)e_n\,, \tag{2.21}$$

or

$$e_n = (I - G)^{-1} d_n\,. \tag{2.22}$$

From this it follows that

$$\|e_n\| \le \|(I - G)^{-1}\|\|d_n\| \tag{2.23}$$

for compatible matrix and vector norms. Moreover, if we take the matrix norm to be the spectral norm and the vector norm the 2-norm, then if G is diagonalizable it follows that

$$\|e_n\| \le \frac{1}{1-\rho(G)}\|d_n\| \tag{2.24}$$

for $\rho(G) < 1$.

The importance of this result should be clear. The exact error is what we would like to know, but to actually compute it requires as much work as computing the exact solution, as is clear from (2.16). On the other hand, d_n is easily computed. At the same time, $\rho(G)$ can often be estimated quite accurately (and inexpensively), either theoretically or numerically, as we will see below. It is also important to observe that Eq. (2.24) implies that $\|e_n\|$ can be very much greater than $\|d_n\|$. In particular, as $\rho(G) \to 1$, the coefficient of $\|d_n\|$ on the right-hand side of (2.24) grows unboundedly. Thus, the inexpensively computed $\|d_n\|$ may not provide a good measure of solution accuracy. This makes estimation of $\rho(G)$ quite important. The following theorem provides a very effective manner in which this can be done.

Theorem 2.2 *Let e_n and d_n be as defined in Eqs.* (2.14) *and* (2.15), *respectively, and suppose $\rho(G) < 1$ holds. Then for any norm, $\|\cdot\|$,*

$$\lim_{n\to\infty} \frac{\|d_n\|}{\|d_{n-1}\|} = \rho(G)\,, \tag{2.25}$$

and

$$\lim_{n\to\infty} \frac{\|e_n\|}{\|d_n\|} = \frac{1}{1-\rho(G)}\,. \tag{2.26}$$

Proof. We will prove these only for the 2-norm. For a complete proof the reader is referred to [15]. To prove the first of these we again recall that

$$d_n = Gd_{n-1}\,,$$

which implies

$$\|d_n\| \le \|G\|\|d_{n-1}\| \quad \Rightarrow \quad \lim \|d_n\| \le \lim \|G\|\|d_{n-1}\|$$

for compatible norms. In particular, if we take the vector norm to be the 2-norm we may use the spectral norm as the matrix norm. Then if G is diagonalizable we have

$$\|G\| = \rho(G)\,.$$

Thus,

$$\lim_{n\to\infty} \frac{\|d_n\|}{\|d_{n-1}\|} \le \rho(G)\,.$$

Next, we employ the *Rayleigh quotient* to arrive at the reverse inequality. Namely, we have since d_n is an eigenvector of G for $n \to \infty$ (this is clear from a formula for d_n analogous to (2.17), and the form of the power method for finding eigenvalues of a matrix),

$$\rho(G) = \lim \frac{\langle d_{n-1}, Gd_{n-1}\rangle}{\|d_{n-1}\|^2} = \lim \frac{\langle d_{n-1}, d_n\rangle}{\|d_{n-1}\|^2}\,.$$

But by the Cauchy–Schwarz inequality it follows that

$$\rho(G) \le \lim \frac{\|d_{n-1}\|\|d_n\|}{\|d_{n-1}\|^2} = \lim \frac{\|d_n\|}{\|d_{n-1}\|}\,,$$

thus completing the proof of (2.25) for the case of the 2-norm.

To prove (2.26) we first note that we have already shown in (2.24) that

$$\frac{\|e_n\|}{\|d_n\|} \le \frac{1}{1-\rho(G)}\,.$$

Now from (2.21) we also have

$$(I-G)e_n = d_n\,,$$

and

$$\|(I-G)e_n\| = \|d_n\|\,.$$

This implies

$$\|d_n\| \le \|(I-G)\|\|e_n\|\,,$$

again, for compatible matrix and vector norms. So using the matrix spectral norm and the vector 2-norm, respectively, gives

$$\frac{\|e_n\|}{\|d_n\|} \ge \frac{1}{1-\rho(G)}\,,$$

and taking the limit $n \to \infty$ completes the proof.

This concludes our introduction to basic linear fixed-point iteration.

2.2 Successive Overrelaxation

In this section we will provide a fairly complete description of one of the historically most-used iterative methods for solution of sparse linear systems, *viz.*, successive overrelaxation (SOR). As is well known, SOR can be developed as the last step in a sequence of methods beginning with Jacobi iteration, proceeding through Gauss–Seidel iteration, and arriving, finally, at SOR. It is also the case that much of the rigorous theory of SOR (see Young [15]) relies on the theory of Jacobi iteration. Thus, we will begin our treatment of SOR with some basic elements of Jacobi iteration theory. We will then continue with the theory of SOR, itself, culminating in the main theorem regarding optimal SOR iteration parameters. We then consider several modifications to basic "point" SOR that are widely used in implementations, devoting a subsection to each of red-black ordered SOR, symmetric SOR (SSOR) and finally successive line overrelaxation (SLOR).

2.2.1 Jacobi iteration

In this subsection we will first introduce the "model problem" that will be used extensively throughout the lectures on numerical treatment of elliptic PDEs, namely, the Laplace–Dirichlet problem on a finite domain. We present the standard second-order centered discretization of this problem and then construct the corresponding Jacobi fixed-point iteration procedure for solving it. We proceed to analytically determine the spectral radius $\rho(B)$ of the associated iteration matrix B, and we use this to determine the asymptotic convergence rate of Jacobi iterations. Finally, we demonstrate how this can be used to predict the total arithmetic required to obtain a solution of specified accuracy on a grid containing N $(= N_xN_y)$ points.

The model problem and discretization

Throughout most of our discussions we will usually be concerned with the basic mathematical problem of Laplace's (or Poisson's) equation with Dirichlet boundary conditions on a 2-D rectangular domain Ω; that is,

$$u_{xx} + u_{yy} = f(x,y)\,, \qquad (x,y) \in \Omega \subseteq \mathbb{R}^2\,, \tag{2.27a}$$

with

$$u(x,y) = g(x,y)\,, \qquad (x,y) \in \partial\Omega\,. \tag{2.27b}$$

Here, $\Omega \equiv [a_x,b_x]\times[a_y,b_y]$, and f and g are given functions assumed to be in $\mathrm{C}(\Omega)$, or $\mathrm{C}(\partial\Omega)$, respectively. We will employ a standard second-order centered discretization,

$$\frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{h_x^2} + \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{h_y^2} = f_{i,j}\,, \quad i,j = 2,3,\ldots,N_x{-}1(N_y{-}1)\,, \tag{2.28a}$$

with

$$u_{i,j} = g_{i,j}\,, \qquad \begin{array}{l} i = 1 \text{ or } N_x \text{ with } j = 2,\ldots,N_y-1\,, \\ j = 1 \text{ or } N_y \text{ with } i = 2,\ldots,N_x-1\,. \end{array} \tag{2.28b}$$

For simplicity, we will assume uniform grid spacing in each of the separate directions. This problem setup is shown in Fig. 2.5, a modification of Fig. 2.1 for the special case of the Laplacian considered here.

We calculate h_x and h_y from

$$h_x = \frac{b_x - a_x}{N_x - 1}\,, \qquad h_y = \frac{b_y - a_y}{N_y - 1}\,,$$

respectively. But we will typically set $h_x = h_y = h$ for most analyses, and in this case Eqs. (2.28) collapse to

$$u_{i-1,j} + u_{i,j-1} - 4u_{i,j} + u_{i,j+1} + u_{i+1,j} = h^2 f_{i,j}\,, \qquad i,j = 2,3,\ldots,N_x - 1(N_y - 1) \tag{2.29}$$

with boundary conditions as before.

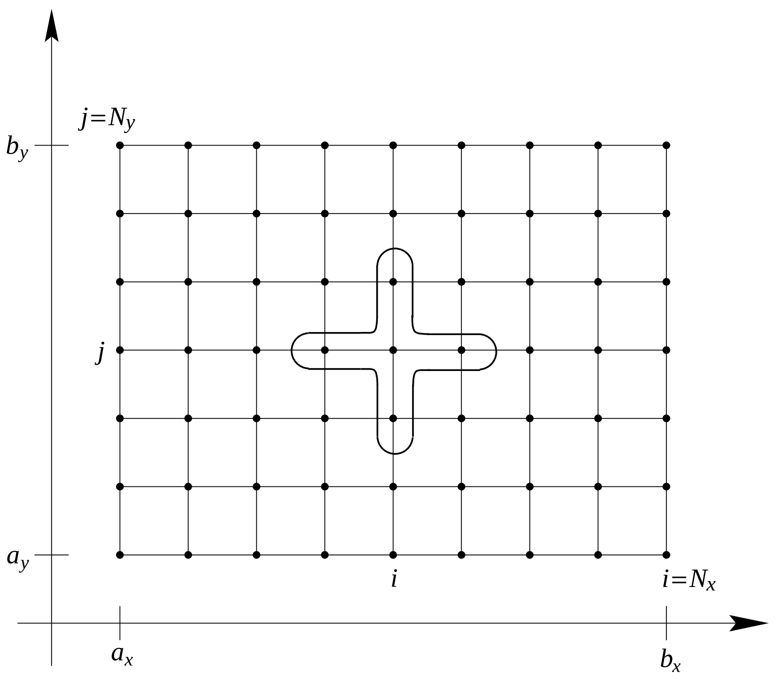

Figure 2.5: Discretization of the Laplace/Poisson equation on a rectangular grid of $N_x \times N_y$ points.

The Jacobi iteration formula

If we now "solve" (2.29) for $u_{i,j}$ and introduce iteration counters, we obtain the Jacobi iteration formula

$$u_{i,j}^{(n+1)} = \frac{1}{4}\left[u_{i-1,j}^{(n)} + u_{i,j-1}^{(n)} + u_{i,j+1}^{(n)} + u_{i+1,j}^{(n)} - h^2 f_{i,j}\right] \tag{2.30}$$

$\forall\ i,j$ in the index set of Eq. (2.29). This set of equations is clearly in fixed-point form and thus can be expressed as

$$u^{(n+1)} = Bu^{(n)} + k\,, \tag{2.31}$$

where B is the *Jacobi iteration matrix*. It should be clear from (2.30) that this matrix has the band structure shown in Fig. 2.6 for the present case of the discrete Poisson equation.

Eigenvalues of Jacobi iteration matrix

In order to investigate convergence of the iterations (2.31) we need to find $\rho(B)$, the spectral radius of the Jacobi iteration matrix B. In the present case we will do this analytically. We begin by expressing the eigenvalue problem for this matrix as

$$Bv = \mu v\,, \tag{2.32}$$

where v is an eigenvector corresponding to the eigenvalue μ. Equation (2.30) and the band structure of Fig. 2.6 imply that we can write an arbitrary equation from the system (2.32) as

$$\frac{1}{4}\Big[v_{i-1,j} + v_{i,j-1} + v_{i,j+1} + v_{i+1,j}\Big] = \mu v_{i,j}\,, \tag{2.33}$$

which we will also write in the "analytical" difference equation form

$$\frac{1}{4}\Big[v(x-h,y) + v(x,y-h) + v(x,y+h) + v(x+h,y)\Big] = \mu v(x,y)\,. \tag{2.34}$$

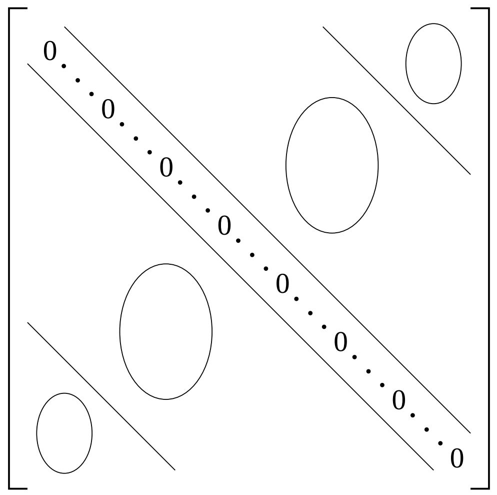

Figure 2.6: Band structure of Jacobi iteration matrix for Laplace/Poisson equation.

Now from the homogeneity of eigenvalue problems, and from the fact that the original differential equation boundary-value problem was of Dirichlet type, we conclude (by inspection) that

$$v(x,y) = \sin\frac{p\pi x}{a}\sin\frac{q\pi y}{b}\,, \tag{2.35}$$

where we have taken the domain Ω to be $(0,a)\times(0,b)$. That is, $v(x,y) = 0$ on $\partial\Omega$. Substitution of (2.35) into (2.34) followed by some standard, but somewhat tedious, manipulations of trigonometric formulas results in the expression

$$\mu = \frac{1}{2}\left(\cos\frac{p\pi h}{a} + \cos\frac{q\pi h}{b}\right) \tag{2.36}$$

for the eigenvalues of the Jacobi iteration matrix B. In (2.35) and (2.36) p and q may be any integers; but recall that B is only a $[(N_x-2)(N_y-2)]\times[(N_x-2)(N_y-2)]$ matrix, so p and q can take on values only between 1 and N_x-2 and 1 and N_y-2, respectively, before repetitions begin to occur in the values of μ calculated from (2.36).

Our task now is to find $\rho(B)$, *i.e.*, to find the maximum value of μ for p and q in the above ranges. It is easy to check by inspection that this maximum occurs for $p = q = 1$, yielding the result

$$\rho(B) = \frac{1}{2}\left(\cos\frac{\pi h}{a} + \cos\frac{\pi h}{b}\right)\,. \tag{2.37}$$

This is the exact spectral radius of the Jacobi iteration matrix for a second-order centered-difference approximation to a Dirichlet problem for Laplace's (Poisson's) equation on a $N_x\times N_y$ gridding of the rectangular domain $\Omega = (0,a)\times(0,b)$ such that $h_x = h_y = h$. Clearly, it follows that the corresponding result for Ω the unit square is

$$\rho(B) = \cos\pi h\,. \tag{2.38}$$

Asymptotic convergence rate and total arithmetic

To obtain further analytical results it is convenient to expand (2.37) in a Taylor series as

$$\rho(B) = 1 - \frac{1}{4}\left[\left(\frac{\pi}{a}\right)^2 + \left(\frac{\pi}{b}\right)^2\right]h^2 + \mathcal{O}(h^4)\,, \tag{2.39}$$

from which it follows (after yet another Taylor expansion) that the asymptotic convergence rate for Jacobi iterations is

$$R_\infty(B) = -\log\rho(B) = \frac{1}{4}\left[\left(\frac{\pi}{a}\right)^2 + \left(\frac{\pi}{b}\right)^2\right]h^2 + \mathcal{O}(h^4)\,. \tag{2.40}$$

Again, for the special case of the unit square this is

$$R_\infty(B) = \frac{1}{2}\pi^2 h^2 + \mathcal{O}(h^4)\,. \tag{2.41}$$

It is important to recognize that $R_\infty(B) \sim h^2$, so as the grid spacing of a discrete method is decreased to obtain better accuracy the convergence rate decreases as the square of h. This ultimately leads to large expenditures of floating-point arithmetic in applications of Jacobi's method, as we will now demonstrate in a more quantitative way.

We can use the preceding analysis to estimate the total arithmetic required by a Jacobi iteration procedure to reduce the error in a computed solution by a factor r. To begin we recall that the average convergence rate given in Eq. (2.18) provides a formula for the number of iterations, n:

$$n = -\frac{1}{R_n}\log\|G^n\|\,.$$

Also, from (2.17) we have

$$\frac{\|e_n\|}{\|e_0\|} \le \|G^n\|\,.$$

Now note that $\|e_n\|/\|e_0\|$ is an error reduction ratio that might be prescribed by a user of an elliptic equation solving program; we denote this by r:

$$r \equiv \frac{\|e_n\|}{\|e_0\|}\,.$$

Then

$$n \le -\frac{1}{R_n}\log r\,,$$

with $\log r \sim \mathcal{O}(1)$ to $\mathcal{O}(10)$ being typical.

At this point we recall that $R_n \to R_\infty$ as $n \to \infty$, and assume n is sufficiently large that $R_n \simeq R_\infty$. It then follows that the required number of Jacobi iterations to reduce the error by a factor r below the initial error is

$$n \le -\frac{1}{\frac{1}{2}\pi^2 h^2}\log r\,, \tag{2.42}$$

and since $h \sim 1/N_x$ this implies that $n \sim \mathcal{O}(N)$. Finally, at each iteration $\mathcal{O}(N)$ arithmetic operations will be required, so the total arithmetic for Jacobi iteration is $\mathcal{O}(N^2)$.

2.2.2 SOR theory

We recall from elementary numerical analysis that SOR is obtained as an extrapolated version of Gauss–Seidel iteration (which, itself, is derived from the preceding Jacobi iterations), and neither Gauss–Seidel nor SOR appear as fixed-point iterations in their computational forms. We have already seen the value of the analysis that is available for fixed-point iterations, so it is natural to convert SOR to this form in order to attempt prediction of its optimal iteration parameter and its required total arithmetic. We begin this section by deriving this fixed-point form. Then we introduce a series of definitions and theorems associated with convergence of Jacobi and SOR iterations, culminating in a theorem containing an explicit formula for the optimal SOR iteration parameter, ω_b, expressed in terms of the spectral radius of the <u>Jacobi iteration matrix</u>.

Fixed-point form of SOR

We again consider the linear system

$$Au = b\,, \tag{2.43}$$

where A is a sparse, nonsingular $N \times N$ matrix, and decompose A as

$$A = D - L - U\,, \tag{2.44}$$

as was done earlier in Eq. (2.6). Substitution of this into (2.43) followed by some rearrangement leads to

$$(D - L)u^{(n+1)} = Uu^{(n)} + b\,, \tag{2.45}$$

or

$$D(I - D^{-1}L)u^{(n+1)} = Uu^{(n)} + b\,,$$

and

$$u^{(n+1)} = (I - D^{-1}L)^{-1}D^{-1}Uu^{(n)} + (I - D^{-1}L)^{-1}D^{-1}b\,. \tag{2.46}$$

This is the fixed-point form of *Gauss–Seidel iteration*, and it is clearly in the usual linear fixed-point form

$$u^{(n+1)} = Gu^{(n)} + k\,.$$

In the present case we define

$$\mathcal{L} \equiv (I - D^{-1}L)^{-1}D^{-1}U\,, \tag{2.47a}$$

$$k \equiv (I - D^{-1}L)^{-1}D^{-1}b\,, \tag{2.47b}$$

and write

$$u^{(n+1)} = \mathcal{L}u^{(n)} + k\,. \tag{2.48}$$

We now recall that successive overrelaxation is obtained from Gauss–Seidel iteration by introducing the *relaxation parameter* ω via the extrapolation

$$u^{(n+1)} = (1 - \omega)u^{(n)} + \omega u^{(n+1)^*}\,, \tag{2.49}$$

where $u^{(n+1)^*}$ has been obtained from (2.48). This leads us to the fixed-point formula for SOR iterations:

$$u^{(n+1)} = (1 - \omega)u^{(n)} + \omega\left[D^{-1}Lu^{(n+1)} + D^{-1}Uu^{(n)} + D^{-1}b\right]\,,$$

or

$$u^{(n+1)} = (I - \omega D^{-1}L)^{-1}\left[\omega D^{-1}U + (1 - \omega)I\right]u^{(n)} + \omega(I - \omega D^{-1}L)^{-1}D^{-1}b\,. \tag{2.50}$$

We note that the equation preceding (2.50) is easily obtained from the "computational" form of Gauss–Seidel iterations,

$$u^{(n+1)^*} = D^{-1}Lu^{(n+1)} + D^{-1}Uu^{(n)} + D^{-1}b\,, \tag{2.51}$$

a rearrangement of Eq. (2.45). If we now define

$$\mathcal{L}_\omega \equiv (I - \omega D^{-1}L)^{-1}\left[\omega D^{-1}U + (1 - \omega)I\right]\,, \tag{2.52a}$$

$$k_\omega \equiv \omega(I - \omega D^{-1}L)^{-1}D^{-1}b\,, \tag{2.52b}$$

we can write Eq. (2.50) as

$$u^{(n+1)} = \mathcal{L}_\omega u^{(n)} + k_\omega\,, \tag{2.53}$$

the *fixed-point form of SOR*. It is important to note that although this form is crucial for analysis of SOR, it is not efficient for numerical calculations. For this purpose the combination of (2.51) and (2.49) should always be used.

Consistent ordering and property A

We now introduce some important definitions and theorems leading to the principal theorem in SOR theory containing formulas for the optimal SOR parameter and the spectral radius of $\mathcal{L}_\omega$. There are two crucial notions that ultimately serve as necessary hypotheses in the majority of theorems associated with SOR: *consistently ordered* and *property A*. To motivate the need for the first of these we recall that in contrast to Jacobi iterations, in SOR the order in which individual equations of the system are evaluated influences the convergence rate, and even whether convergence occurs. This is easily recognized by recalling the computational formulas for SOR, written here for a second-order centered finite-difference approximation of the Poisson equation:

$$u_{i,j}^{(n+1)^*} = \frac{1}{4}\left[u_{i-1,j}^{(n+1)} + u_{i,j-1}^{(n+1)} + u_{i,j+1}^{(n)} + u_{i+1,j}^{(n)} - h^2 f_{i,j}\right], \tag{2.54a}$$

$$u_{i,j}^{(n+1)} = \omega u_{i,j}^{(n+1)^*} + (1-\omega)u_{i,j}^{(n)}, \tag{2.54b}$$

for grid point (i,j). Obviously, reordering the sequence of evaluation of the $u_{i,j}$s will change which of the values are known at the advanced iteration level on the right-hand side of Eq. (2.54a). In turn, this will have an effect on the matrix representation of the fixed-point form of this procedure, and thus also on the spectral radius. In particular, there are problems for which it is possible for some orderings to be convergent and others divergent. In order to make precise statements regarding convergence of SOR we will need the notion of a *consistently-ordered matrix* given in the following definition.

Definition 2.6 *The $N \times N$ matrix A is* $\underline{\textit{consistently ordered}}$ *if for some K $\exists$ disjoint subsets $S_1, S_2, \ldots, S_K \subset W = \{1,2,\ldots,N\} \ni \cup_{k=1}^K S_k = W$, and for $i \in S_k$ with either $a_{i,j} \neq 0$ or $a_{j,i} \neq 0$, then $j \in S_{k+1}$ for $j > i$, and $j \in S_{k-1}$ for $j < i$.*

Application of this definition is not especially easy although it can be formulated as a computational algorithm (see Young [15] for details). Here, we will give a simple example. Consider the 4×4 matrix

$$A = \begin{bmatrix} 4 & 0 & 0 & -1 \\ -1 & 4 & -1 & 0 \\ 0 & -1 & 4 & 0 \\ -1 & 0 & 0 & 4 \end{bmatrix}.$$

In the notation of the above definition we have $W = \{1,2,3,4\}$. We will somewhat arbitrarily choose to set $K = 3$, and then check whether this leads to satisfaction of the conditions of the definition. Let $S_1 = \{1\}$, $S_2 = \{2,4\}$ and $S_3 = \{3\}$ so that $\cup_k S_k = W$, and the S_ks are disjoint. Now we check the condition on the relationships between (i,j) and the S_ks for all matrix elements $a_{i,j}, a_{j,i} \neq 0$, and $i \neq j$. For example, for the matrix element a_{14}, $i \in S_1$, $j \in S_2 \Rightarrow j \in S_{k+1}$; for a_{21}, $i \in S_2$ and $j \in S_1$. Hence, $j \in S_{k-1}$, and we see that the conditions are satisfied for these two elements. We leave verification of this for the remaining elements of A as an exercise for the reader.

For the case of the 5-band discrete Laplacian in 2-D, and the corresponding 7-band 3-D case, there is a simple geometric approach that can be used to check consistent ordering. Figure 2.7 demonstrates this. In part (a) of the figure we present a grid with the order in which equations of the iteration scheme are to be evaluated indicated adjacent to the corresponding grid points. The geometric test for consistent ordering is carried out by inserting arrows pointing from the lower index to the higher one along each grid line segment. After this has been done, one checks that the number of clockwise-pointing arrows equals the number of counterclockwise-pointing ones in each grid cell. This turns out to be true for part (a) of the figure. To show that the notion of consistent ordering is nontrivial, we have in part (b) provided an example of an ordering that is not consistent (see two lower right-hand corner grid cells). We remark that proof of equivalence of the formal definition of consistent ordering and the geometric test is straightforward, but tedious. We leave this as an interesting exercise for the reader.

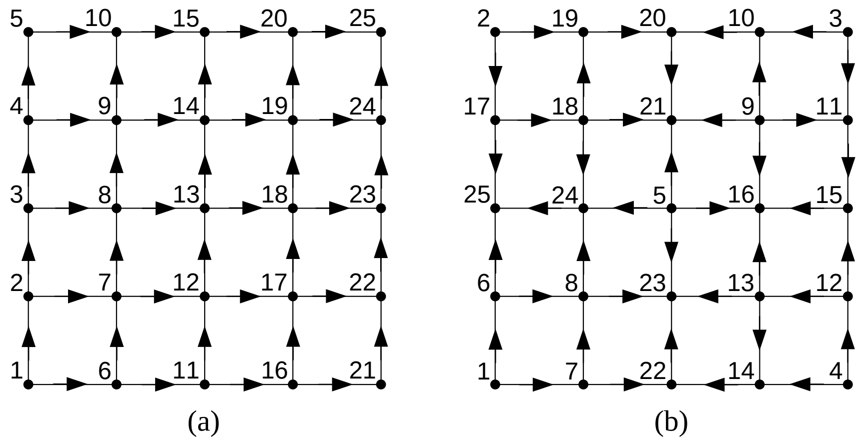

Figure 2.7: Geometric test of consistent ordering. (a) consistent ordering, (b) nonconsistent ordering.

We also remark that the ordering in part (a) of the figure is one of two *natural orderings*, the other one corresponding to increasing the index first in the horizontal direction. Both orderings are widely used in practice.

The consistent-ordering property has numerous characterizations. Here, we present an additional one that is widely quoted, and sometimes used as the *definition* of consistently ordered.

Theorem 2.3 *If the matrix A is consistently ordered, then* $\det(\alpha L + \frac{1}{\alpha}U - \kappa D)$ *is independent of* α $(\alpha \neq 0)$ $\forall\ \kappa$.

Our first result to make use of the consistent-ordering property is contained in the following.

Theorem 2.4 *Let A be a symmetric, consistently-ordered matrix with positive diagonal elements. Then $\rho(B) < 1$ iff A is positive definite.*

This theorem provides a very strong result concerning convergence of Jacobi iterations.

The definition of consistently ordered given above is clearly quite tedious to apply, as is highlighted by the simple example. We have seen that in some cases it is possible to employ a fairly easy geometric test, but this cannot be applied to the general 9-point discrete operator considered in Eq. (2.2) and Fig. 2.1. This motivates the search for at least a nearly equivalent property that is easier to test, leading us to consider the characterization known as *property* A.

Definition 2.7 *A $N \times N$ matrix A has* $\underline{\text{property A}}$ *if $\exists$ two disjoint subsets $S_1, S_2 \subset W \equiv \{1, 2, \dots, N\}$ $\ni S_1 \cup S_2 = W$, and if $i \neq j$ and either $a_{i,j} \neq 0$ or $a_{j,i} \neq 0$, then $i \in S_1 \Rightarrow j \in S_2$, or $i \in S_2 \Rightarrow j \in S_1$.*

The importance of property A is that it slightly widens the class of matrices to which SOR theory may be applied, and at the same time it provides a more readily checkable characterization of these matrices. In particular, not every matrix having property A is consistently ordered. However, we have the following theorem, which we state without proof (see [15] for a proof) that connects these two matrix properties.

Theorem 2.5 *Let the matrix A have property A. Then $\exists$ a permutation matrix $P \ni A' = P^{-1}AP$ is consistently ordered.*

It is clear that the matrix P generates a *similarity transformation*, and hence A' which is consistently ordered has the same spectrum as A, which has only property A. Thus, any spectral properties (the spectral radius, in particular) that hold for a consistently-ordered matrix also hold for a matrix having property A. But the similarity transformation of the theorem does not lead to consistent ordering for all permutation matrices P, so analyses involving these ideas must include finding an appropriate matrix P.

Optimal SOR parameter

In this subsection we will present a formula for the optimal SOR relaxation parameter, denoted ω_b. Our treatment will rely on the preceding results, and follows that found in [15]. But we note that similar results can be obtained from basically geometric arguments, as also given in [15] and elsewhere, *e.g.*, Mitchell and Griffiths [16] and Varga [17]. We begin by stating a pair of theorems that are needed in the proof of the main theorem concerning the value of the optimal SOR parameter.

For consistently-ordered matrices it can be shown (see *e.g.*, [16]) that the eigenvalues of the SOR iteration matrix are related to those of the Jacobi iteration matrix (which, as we have already shown, can often be calculated exactly) by the formula given in the following theorem.

Theorem 2.6 *Suppose the matrix A is consistently ordered, and let B and $\mathcal{L}_\omega$ be the Jacobi and SOR, respectively, iteration matrices associated with A. Let $\mu \in \sigma(B)$ and $\lambda \in \sigma(\mathcal{L}_\omega)$, ω being the SOR iteration parameter. Then*

$$\lambda - \omega\mu\lambda^{1/2} + \omega - 1 = 0\,. \tag{2.55}$$

Our first result regarding the SOR parameter ω is the following.

Theorem 2.7 *Suppose A is consistently ordered with nonvanishing diagonal elements, and such that the Jacobi iteration matrix B has real eigenvalues. Then $\rho(\mathcal{L}_\omega) < 1$ iff $0 < \omega < 2$, $\underline{\text{and}}$ $\rho(B) < 1$.*

Proof. The proof follows directly from the following lemma applied to (2.55).

Lemma *If $b, c \in \mathbb{R}$, then both roots of*

$$x^2 - bx + c = 0$$

have modulus less than unity iff $|b| < 1 + c$ and $|c| < 1$.

Proof of the lemma follows from a direct calculation, which we omit. (It can be found in [15], pg. 171.)

Now take $b = \omega\mu$, and $c = \omega - 1$, viewing (2.55) as a quadratic in $\lambda^{1/2}$. Since $\mu \in \sigma(B)$, proof of the theorem is immediate.

We now display in Fig. 2.8 a summary of results associated with the behavior of the SOR iteration parameter as a function of spectral radius of the SOR iteration matrix, and we state without proof the main result regarding the optimal SOR parameter, ω_b.

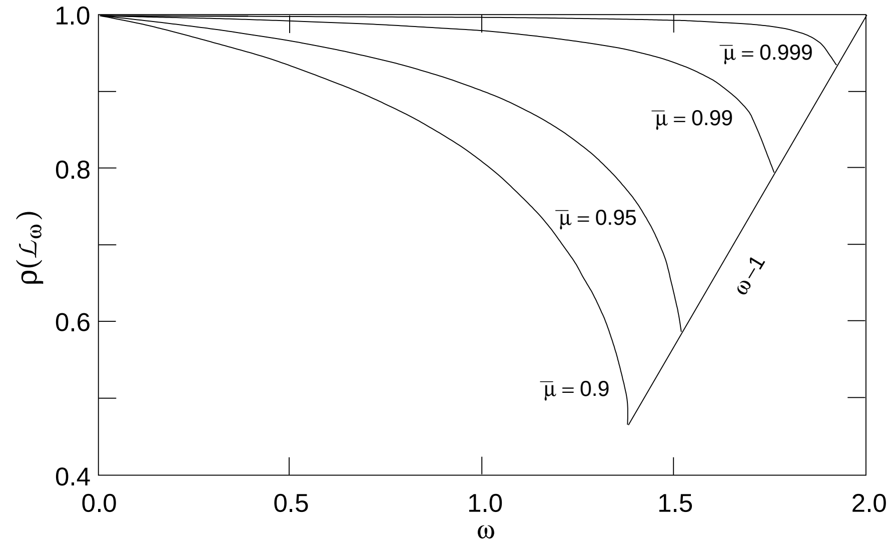

Figure 2.8: Spectral radius of SOR iteration matrix vs. ω.

Theorem 2.8 (Optimal SOR parameter) *Suppose A is a consistently-ordered matrix, and the Jacobi iteration matrix has $\sigma(B)\in\mathbb{R}$ with $\bar\mu \equiv \rho(B) < 1$. Then the optimal SOR iteration parameter is given by*

$$\omega_b = \frac{2}{1+(1-\bar\mu^2)^{1/2}} \, . \tag{2.56}$$

Moreover, $\forall\ \omega \in (0,2)$

$$\rho(\mathcal{L}_\omega) = \begin{cases} \left[\dfrac{\omega\bar\mu + (\omega^2\bar\mu^2 - 4(\omega-1))^{1/2}}{2}\right]^2 & 0 < \omega \le \omega_b \, , \\ \omega - 1 & \omega_b \le \omega < 2 \, . \end{cases} \tag{2.57}$$

Furthermore, $\rho(\mathcal{L}_\omega)$ is a strictly decreasing function of ω for $\omega \in (0,\omega_b)$, and $\rho(\mathcal{L}_\omega) > \rho(\mathcal{L}_{\omega_b})$ for $\omega \ne \omega_b$.

The behavior of $\rho(\mathcal{L}_\omega)$ as a function of ω for various values of $\bar\mu$ is presented, above, in Fig. 2.8. The point where the nonlinear portion of each curve meets the line corresponding to $\omega - 1$ is the location of ω_b for the particular value of $\bar\mu$. It is important to note that $\rho(\mathcal{L}_\omega)$ is obviously not differentiable at this point; as a consequence, it is not possible to derive the formula for the optimal parameter ω_b by simple minimization techniques from Freshman calculus. This leads to considerable difficulty in obtaining the formula (2.56), and in proving the theorem; but as noted earlier such proofs are available in the references, *e.g.*, [15].

We can see from this figure that $\rho(\mathcal{L}_\omega)$ is fairly sensitive to ω, and consequently so is the convergence rate of SOR. It is thus important to be able to predict ω_b with reasonable accuracy. We also see that the nonlinear portion of the curve increases somewhat more rapidly for $\omega < \omega_b$ than does the linear right-hand portion for $\omega > \omega_b$. This leads to the recommendation that if one cannot predict ω accurately, it is better to chose a somewhat high value rather than a low one. On the other hand, it is known that some of the eigenvalues of $\mathcal{L}_\omega$ are no longer real when $\omega > \omega_b$, so this makes testing convergence of a solution more difficult due to oscillations in the iteration error induced by the complex eigenvalues. Finally, we observe from the figure that ω_b increases monotonically with $\bar\mu$ $(= \rho(B))$, which itself increases monotonically with decreasing grid spacing h, as is clear from Eqs. (2.37) and (2.39).

SOR convergence rate

In this subsection we will provide an estimate of the convergence rate for optimal SOR and use this to predict total floating-point arithmetic requirements, just as we were able to do earlier for Jacobi iterations. The key result needed for this is the following theorem which is proved in [15].

Theorem 2.9 *Let A be a consistently-ordered matrix with nonvanishing diagonal elements $\ni \sigma(B) \in \mathbb{R}$, and $\bar\mu \equiv \rho(B) < 1$. Then*

$$2\bar\mu[R(\mathcal{L})]^{1/2} \le R(\mathcal{L}_{\omega_b}) \le R(\mathcal{L}) + 2[R(\mathcal{L})]^{1/2} \, ,$$

with the right-hand inequality holding when $R(\mathcal{L}) \le 3$. Furthermore,

$$\lim_{\bar\mu\to 1^-} [R(\mathcal{L}_{\omega_b})/2(R(\mathcal{L}))^{1/2}] = 1 \, . \tag{2.58}$$

We will use this result to calculate the asymptotic convergence rate for optimal SOR for a Laplace–Dirichlet problem posed on the unit square. Recall from Eq. (2.38) that the spectral radius of the Jacobi iteration matrix is $\cos\pi h$ for this case, with h being the uniform finite-difference grid spacing. Furthermore, Eq. (2.41) gives $R(B) = \frac{1}{2}\pi^2h^2 + \mathcal{O}(h^4)$. Now it is easy to show that Gauss–Seidel iterations converge exactly twice as fast as do Jacobi iterations for this case; *i.e.*, $R(\mathcal{L}) = 2R(B)$. To see this recall Eq. (2.55) in Theorem 2.6, and set $\omega = 1$ in that formula. It follows that $\lambda = \mu^2$, and from this we would intuitively expect that

$$\rho(\mathcal{L}) = \bar\mu^2 \, ,$$

from which the result follows. We note however, that there are some missing technical details that we will mention, but not attempt to prove. In particular, although the correspondence $\lambda = \mu^2$ follows easily, providing a relationship between eigenvalues of the Gauss–Seidel and Jacobi iteration matrices, this alone does not imply that their spectral radii are similarly related—in a sense, this is a "counting" (or ordering) problem: the eigenvalue that corresponds to the spectral radius of B is not necessarily related by the above formula to the eigenvalue corresponding to the spectral radius of $\mathcal{L}$. We comment that this is one of the questions that must be dealt with in proving the main SOR theorem. Hence, since proofs of this theorem are well known, the desired result may be accepted.

Now from (2.58), as $h \to 0$ ($\bar{\mu} \to 1^-$) we have

$$R(\mathcal{L}_{\omega_b}) \to 2[R(\mathcal{L})]^{1/2} \simeq 2\pi h\,. \tag{2.59}$$

It is of interest to consider the practical consequences of this result. Using the above together with $R(\mathcal{L}) = 2R(B) = \pi^2 h^2$ yields

$$\frac{R(\mathcal{L}_{\omega_b})}{R(\mathcal{L})} \simeq \frac{2}{\pi h}\,,$$

which implies that for even very coarse gridding, say $h = 1/20$, this ratio is greater than 12. This indicates that the convergence rate of optimal SOR is more than an order of magnitude greater than that of Gauss–Seidel, even on a coarse grid. The above formula clearly shows that as $h \to 0$ the ratio of improvement obtained from using optimal ω becomes very large.

Related to this is the fact (from Eq. (2.59)) that $R(\mathcal{L}_{\omega_b}) \sim \mathcal{O}(h)$; that is, the convergence rate decreases only linearly with N_x (or N_y) when $\omega = \omega_b$, in contrast to a rate of decrease proportional to N $(= N_x N_y)$ found for Jacobi and Gauss–Seidel iterations. This immediately implies that the required total arithmetic on a $N_x \times N_y$ finite-difference grid is $\mathcal{O}(N^{1.5})$ for SOR with the $\underline{\text{optimal}}$ relaxation parameter. We leave as an exercise for the reader demonstration that in 3D the total arithmetic for optimal SOR is $\mathcal{O}(N^{1.33\ldots})$.

2.2.3 Some modifications to basic SOR

In this section we will briefly describe some modifications that can be made to the basic SOR procedure considered to this point. There are many such modifications, and various of these are treated in detail in, *e.g.*, Hageman and Young [18] and elsewhere. Here we will study only the following three: *i*) red-black ordering, *ii*) symmetric SOR and *iii*) line SOR. Each of these provides specific advantages not found in basic point SOR; but, in general, it is not always clear that these advantages are sufficient to offset the attendant increased algorithmic complexities of these approaches.

Red-black ordering

Red-black ordering, so named because of the "checkerboard" pattern of indexing of the unknowns, offers the advantage that the resultant coefficient matrix is (for the discrete Laplacian) automatically consistently ordered. Numerical experiments indicate somewhat higher convergence rates than are achieved with natural orderings, although this seems to be problem dependent.

For a solution vector

$$u = (u_{1,1}, u_{1,2}, \ldots, u_{1,N_y}, u_{2,1}, \ldots, \ldots, u_{N_x,N_y})^T,$$

which is ordered in one of the two natural orderings, the *red-black* ordering is obtained as follows. We decompose this solution vector into two subvectors, a "red" one and a "black" one, defined such that the sums of their (i, j) indices are, respectively even and odd. That is, for the red vector $i + j$ is even, and for the black vector it is odd.

It is of value to consider a simple example to more clearly demonstrate this decomposition. The mesh shown in Fig. 2.9 includes the interior grid points corresponding to a Laplace–Dirichlet problem on the unit square for which a 5-point centered discretization with uniform grid spacing $h = h_x = h_y = 1/6$ has

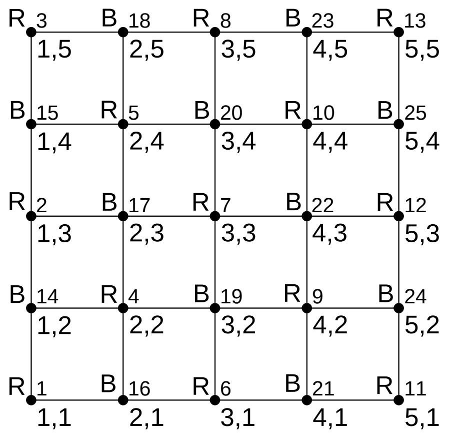

Figure 2.9: Red-black ordering for discrete Laplacian.

been constructed, and boundary-point indexing has been omitted. Hence, all equations will be of the form (2.54).

In this figure, the usual indexing is shown below and to the right of each grid point; whether the grid point is red or black is indicated above and to the left of the point, and the single-index ordering of evaluation is provided above and to the right of each point. The natural ordering in this case is

$$u = (u_{1,1}, \ldots, u_{1,5}, u_{2,1}, \ldots, u_{2,5}, \ldots, \ldots, u_{4,1}, \ldots, u_{4,5}, u_{5,1}, \ldots, u_{5,5})^T ,$$

and the corresponding red-black ordering is

$$u = (\underbrace{u_{1,1}, u_{1,3}, u_{1,5}, u_{2,2}, u_{2,4}, u_{3,1}, \ldots, u_{5,1}, \ldots, u_{5,5}}_{red}, \overbrace{u_{1,2}, u_{1,4}, u_{2,1}, u_{2,3}, u_{2,5}, \ldots, \ldots, u_{5,4}}^{black})^T .$$

We leave as an exercise to the reader construction of the geometric test of consistent ordering of the matrix corresponding to this red-black vector. We also note that the programming changes required to convert a standard SOR code to red-black ordering are relatively minor. Furthermore, extension of these ideas to 3D is straightforward.

Symmetric SOR

We next briefly describe a modification to the basic SOR point iteration procedure known as *symmetric SOR* (SSOR). It is easily checked that the SOR iteration matrix $\mathcal{L}_\omega$ is not symmetric, even when the system matrix A is symmetric. Moreover, from the fact that the eigenvector of a linearly convergent iteration scheme corresponding to the dominant eigenvalue (the spectral radius, up to a sign) is precisely e_n, we expect convergence to be more rapid when the iteration matrix is symmetric in light of properties of the power method iteration procedure for finding the dominant eigenvector (see Isaacson and Keller [2]). The SOR procedure can be easily symmetrized by merely reversing the order in which the equations are evaluated on each successive iteration. More specifically, the symmetric SOR matrix $\mathcal{S}_\omega$ which we will construct below can be shown to be similar to a symmetric matrix (see [18]), and hence it will have the same eigenstructure.

We develop the SSOR procedure as follows. First, recall that the fixed-point form of the usual SOR iteration procedure is Eq. (2.50):

$$u^{(n+1)} = (I - \omega D^{-1}L)^{-1} \left[\omega D^{-1}U + (1-\omega)I\right] u^{(n)} + \omega(I - \omega D^{-1}L)^{-1}D^{-1}b .$$

We can define the backward procedure by interchanging the L and U triangular matrices. Thus, with $\mathcal{L}_\omega$ defined as

$$\mathcal{L}_\omega \equiv (I - \omega D^{-1}L)^{-1}\left[\omega D^{-1}U + (1-\omega)I\right] ,$$

we can define the backward SOR matrix as

$$\mathcal{U}_\omega \equiv (I - \omega D^{-1}U)^{-1}\left[\omega D^{-1}L + (1-\omega)I\right] .$$

Then SSOR can be viewed as a two-step procedure carried out as follows. First calculate

$$u^{(n+\frac{1}{2})} = \mathcal{L}_\omega u^{(n)} + k_{\omega,F} ,$$

where

$$k_{\omega,F} = \omega(I - \omega D^{-1}L)^{-1}D^{-1}b .$$

Then calculate

$$u^{(n+1)} = \mathcal{U}_\omega u^{(n+\frac{1}{2})} + k_{\omega,B} ,$$

with

$$k_{\omega,B} = \omega(I - \omega D^{-1}U)^{-1}D^{-1}b .$$

Substitution of the first of these into the second yields

$$\begin{aligned} u^{(n+1)} &= \mathcal{U}_\omega\left(\mathcal{L}_\omega u^{(n)} + k_{\omega,F}\right) + k_{\omega,B} \\ &= \mathcal{U}_\omega\mathcal{L}_\omega u^{(n)} + \mathcal{U}_\omega k_{\omega,F} + k_{\omega,B} \\ &= \mathcal{S}_\omega u^{(n)} + k_\omega , \end{aligned} \tag{2.60}$$

with $\mathcal{S}_\omega$ and k_ω having obvious definitions.

These constructions are important for analysis of the method; but, as noted previously, efficient implementations do not employ the SOR matrix. Hence, to implement SSOR we need only be able to run the usual Do-loops of a computer-coded implementation both forward and backward.

It is interesting that point SSOR can be shown theoretically to converge twice as fast as does the usual point SOR. However, twice as much arithmetic is performed per iteration in SSOR, so there is no advantage in this regard. It is thus argued that the main thing to be gained is symmetry of the iteration matrix, which can be of value when SSOR is used in conjunction with other iterative methods such as conjugate-gradient acceleration to be treated later.

Line SOR

From the standpoint of wide applicability, successive line overrelaxation (SLOR) is probably the must robust and often-used form of SOR. Its robustness stems from the "more implicit" construction used for SLOR. All of the forms of SOR discussed to this point were implemented so as to obtain an updated solution value at a single point with each application of the iteration formula—hence, the terminology *point SOR*. In SLOR, as the name suggests, a complete grid line of solution values is computed, simultaneously, with each application of the iteration formula. Figure 2.10 depicts this situation.

There are several features to note regarding the SLOR procedure. The first is that the mesh star for any individual grid point is the same as in the SOR case, as the figure indicates, since this is determined by the discretization, and not by the solution procedure. Second, some form of tridiagonal solver is applied to simultaneously compute all grid point values on a given line. Thus, all points are coupled in the direction of the solve, inducing a fairly high degree of implicitness, as alluded to above. This in turn propagates boundary condition information across the entire row of points in a single solve, and because of this it is generally recommended that these solves be done in the direction containing the larger number of points. (We will see below, however, that an alternative is to sweep through all grid points using solves in a

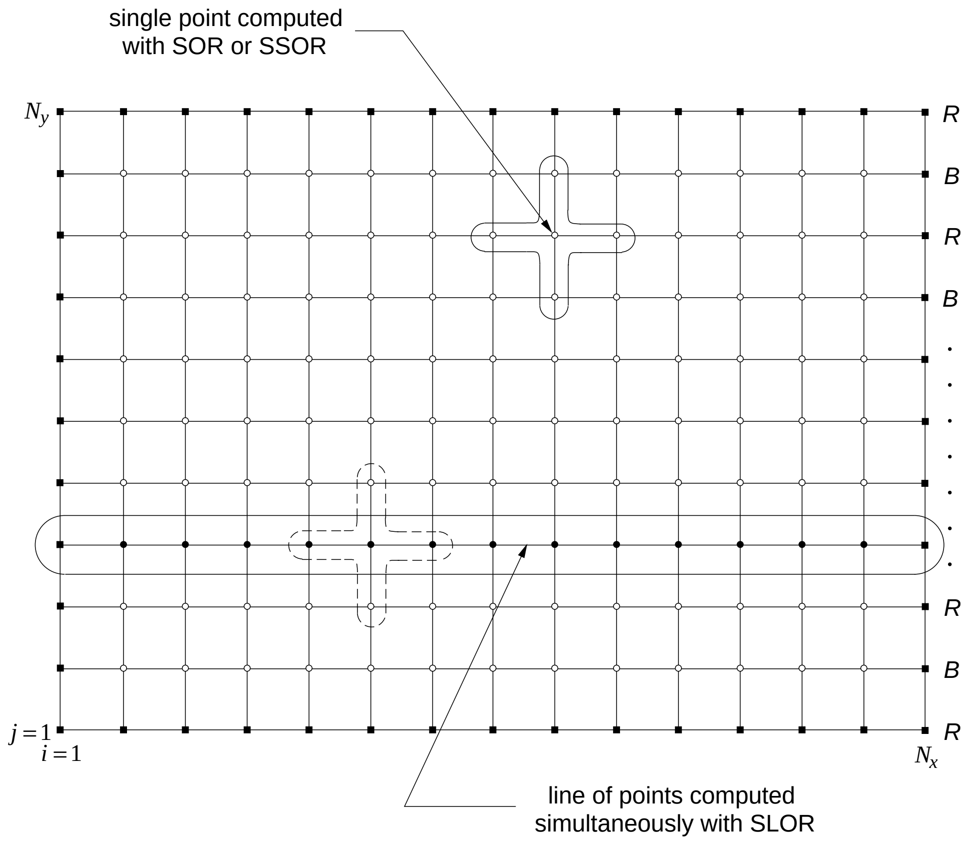
single point computed with SOR or SSOR
N_y
R
B
R
B
R
B
R
$j=1$
$i=1$
N_x
line of points computed simultaneously with SLOR

Figure 2.10: Comparison of computations for point and line SOR showing grid stencils and red-black ordered lines.

particular direction, and then perform a second sweep with solves in the orthogonal direction.) Finally, we comment that there are several possible ways to implement SLOR, and we will discuss two of these here.

We begin with a general 5-point discrete operator,

$$A_1u_{i-1,j} + A_2u_{i,j-1} + A_3u_{i,j} + A_4u_{i,j+1} + A_5u_{i+1,j} = b_{i,j}\,. \tag{2.61}$$

We remark that while this is a simplification of Eq. (2.2), it is more general than a discrete Laplacian because discretization of first-order derivatives and inclusion of zeroth-order terms can be accomodated within this representation.

If we have chosen to solve along horizontal lines, as indicated in Fig. 2.10, we rearrange Eq. (2.61) as

$$A_1u_{i-1,j} + A_3u_{i,j} + A_5u_{i+1,j} = b_{i,j} - A_2u_{i,j-1} - A_4u_{i,j+1}\,. \tag{2.62}$$

It is clear that the $u_{i,j-1}$ component on the right-hand side of this equation will already be known at the current iteration level if we are traversing the lines in the direction of increasing j index. (If not, then $u_{i,j+1}$ will be known.) Thus, we can write the above as

$$A_1u^{(n+1)}_{i-1,j} + A_3u^{(n+1)}_{i,j} + A_5u^{(n+1)}_{i+1,j} = b_{i,j} - A_2u^{(n+1)}_{i,j-1} - A_4u^{(n)}_{i,j+1}\,, \tag{2.63}$$

for each fixed j and $i = 1, 2, \ldots, N_x$. For each such fixed j this is clearly a tridiagonal linear system which can be solved by efficient sparse LU decompositions, or by *cyclic reduction* methods (see *e.g.*, Birkhoff and Lynch [19]). In either case, $\mathcal{O}(N_x)$ arithmetic operations are needed for each line, so $\mathcal{O}(N)$ total arithmetic is required for each iteration, just as in point SOR. As is fairly easy to determine, the arithmetic per line for SLOR is somewhat higher than for the usual point SOR except when A is symmetric. For this special case (which arises, *e.g.*, for Laplace–Dirichlet problems) Cuthill and Varga [20] have provided a numerically stable tridiagonal elimination requiring exactly the same arithmetic per line as used in point SOR.

Before giving details of the implementation of SLOR we will first provide a brief discussion of further generalizations of SOR, usually termed “block” SOR, of which SLOR is a special case. Extensive discussions can be found in Young [15] and Hageman and Young [18]. A brief description similar to the treatment to be given here can be found in Birkhoff and Lynch [19].

Rather than consider a single line at a time, as we have done above, we might instead simultaneously treat multiple lines (thus gaining even further implicitness) by defining subvectors $u_1, u_2, \ldots, u_m$ of the solution vector u and writing the original system as

$$Au = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1m} \\ A_{21} & A_{22} & \cdots & \vdots \\ \vdots & & \ddots & \vdots \\ A_{m1} & \cdots & \cdots & A_{mm} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_m \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix} . \tag{2.64}$$

Here each A_{ii} is a square nonsingular submatrix of A of size $N_i \times N_i$ such that $\sum_{i=1}^{m} N_i = N$. If $N_i = N_x$ or if $N_i = N_y$, then we obtain SLOR.

As noted in [19] it is possible to define block consistently-ordered and block property A analogous to our earlier definitions for point SOR. It is further noted in that reference that Parter [21] has shown for k-line SOR applied to discrete equations corresponding to a Poisson equation on the unit square that the spectral radius of the SLOR iteration matrix is

$$\rho(\mathcal{L}_{\omega_b}^{(k-line)}) \simeq 1 - 2\pi\sqrt{2k}h\,, \tag{2.65}$$

and correspondingly, the asymptotic convergence rate is given by

$$R(\mathcal{L}_{\omega_b}^{(k-line)}) \simeq 2\pi\sqrt{2k}h\,. \tag{2.66}$$

Thus, typical single-line SLOR ($k = 1$) has a convergence rate that is a factor $\sqrt{2}$ larger than that of ordinary point SOR. We should also comment that because the block formalism is constructed in the same manner as is point SOR, and in addition because the block structure has no influence on the Jacobi iteration matrix, the optimal parameter for such methods is expected to be the same as for point SOR.

We are now prepared to consider two specific implementations of SLOR. The first is a direct line-by-line application of Eq. (2.63). This is embodied in the following pseudo-language algorithm.

Algorithm 2.1 (Successive line overrelaxation) *Suppose n iterations have been completed. To perform iteration $n+1$ carry out the following steps.*

```
Do $j = 1, N_y$
   1. Load matrix coefficients $A_{1,ij}, A_{3,ij}, A_{5,ij}$ and right-hand side vector $b_{ij} - A_{2,ij}u_{i,j-1}^{(n+1)} - A_{4,ij}u_{i,j+1}^{(n)}$
   2. Call tridiagonal solver to compute $\{u_{ij}^*\}_{i=1}^{N_x}$
   3. Update solution vector:
         $u_{ij}^{(n+1)} = (1-\omega)u_{ij}^{(n)} + \omega u_{ij}^*\,, \qquad i = 1, 2, \cdots, N_x$
Repeat $j$
```

While this approach can be quite effective on single-processor vector machines, especially if a cyclic reduction algorithm is used for the tridiagonal solver, it is not particularly efficient on parallel processing hardware because each successive line solve depends on results from the preceding one. This dependence can be broken in at least two ways: *i*) wait until all line solves for a given iteration are complete before performing the update in step *3* of the preceding algorithm, or *ii*) employ red-black ordering of the lines, as indicated in Fig. 2.10. The first of these is effectively a block Jacobi iteration (but with SOR-like acceleration at the end of each sweep through all grid points), but because potentially all line solves can be done simultaneously (depending on details of the parallel hardware and system software available) it can be a very efficient procedure. The second, although somewhat more intricate to implement, is preferred because it is somewhat less hardware and system dependent.

It is clear from Fig. 2.10 that in the red-black ordered implementation, dependence of successive red (or black) lines on preceding ones is completely broken, and from a parallelization standpoint the only difficulty is dealing with the fact that, *e.g.*, successive red lines must access a common black line to perform their calculations. This is not, however, a major issue on most modern parallel hardware utilizing either MPI (Message Passing Interface) or the shared-memory version of this, OpenMP. This permits calculation on all red lines simultaneously (if there are sufficient processors), followed by a similar treatment of all black lines to complete an iteration. This makes it possible to employ as many as $N_y/2$ (or $N_x/2$) processors in a straightforward, efficient way—it is easily seen that this is an instance of block Gauss–Seidel iteration, which with $\omega > 1$ becomes block SOR. Clearly, good load balancing occurs automatically; but we must emphasize that a truly effective implementation can depend on details of the hardware and system software available, and on problem size—in particular, the amount of arithmetic needed for a line solve compared with the communication required to set up the solve.

The following pseudo-language algorithm contains the main ideas embodied in this approach (but with no specific notation for parallelization commands).

Algorithm 2.2 (Red-black ordered SLOR) *Suppose n iterations have been completed. To perform iteration $n+1$ carry out the following steps.*

```
A. Calculation of red vectors

  Do $j = 1, N_y, 2$
    1. Load matrix coefficients $A_{1,ij}$, $A_{3,ij}$, $A_{5,ij}$ and right-hand side vector $b_{ij} - A_{2,ij}u^{(n+1)}_{i,j-1} - A_{4,ij}u^{(n)}_{i,j+1}$
    2. Call tridiagonal solver to compute $\{u^*_{ij}\}_{i=1}^{N_x}$
  Repeat j

    3. Update red vectors:

      Do $j = 1, N_y, 2$
        Do $i = 1, N_x$
          $u^{(n+1)}_{ij} = (1-\omega)u^{(n)}_{ij} + \omega u^*_{ij}$
        Repeat i
      Repeat j

B. Calculation of black vectors

  Do $j = 2, N_y, 2$
    Repeat steps 1. and 2. of part A.
  Repeat j

    3. Update black vectors:

      Do $j = 2, N_y, 2$
        Do $i = 1, N_x$
          $u^{(n+1)}_{ij} = (1-\omega)u^{(n)}_{ij} + \omega u^*_{ij}$
        Repeat i
    Repeat j
```

Observe that this algorithm structure lies between the two possibilities indicated above because SOR accelerations are carried out for all red vector components prior to calculating any black vector components, and conversely. This approach is expected to exhibit the same basic performance characteristics as found for red-black ordered point SOR—of course, *modulo* convergence rate increases due to the improved implicitness obtained from line solves. This is due to the fact that, as noted earlier, all red (or black) components are computed first using only black (red) vector information. Typically, these are updated as they are computed, but this has no specific effect because they are not used until all have been calculated.

Finally we note that, specifically within the context of parallelization, there is yet another possible ordering that can completely eliminate potential message-passing problems arising from the need to share data during red-black ordering calculations. It is use of red-green-blue (RGB) ordering. It is easy to see that data required for each individual red (or green, or blue) line are completely independent of those needed for any other red line. Hence, although the potential for employing large numbers of processors is reduced by a factor of three, efficiency can be significantly improved. Moreover, coding the corresponding parallelization instructions is far easier because of complete absence of data overlaps. We leave to the reader the exercise of developing a pseudo-language algorithm for this case similar to those presented above.

2.3 Alternating Direction Implicit (ADI) Procedures

In this section we will treat a class of methods first introduced by Peaceman and Rachford [22] for solving the time-dependent heat equation in two space dimensions. It was quickly recognized that the unconditional stability of the method might render it effective as a steady-state (hence, elliptic) solver due to the possibility of employing large time steps for *pseudo-time* marching to a steady state. At each pseudo-time step (iteration) the discrete equations are implicitly (line-by-line) solved first in one spatial direction, and then in the other, leading to the terminology *alternating direction implicit.* A development along these lines is given in [16]. Here we will use a somewhat different, more mathematically-oriented, approach based on the theory of linear fixed-point iteration, as has already been used in the study of SOR. Our treatment closely follows that found in [17] with some results from [15] also used.

We should remark at the outset that at one time ADI schemes were considered to be among the most efficient possible methods for solving elliptic equations. As we have indicated in Fig. 2.3, the total arithmetic can be as low as $\mathcal{O}(N\log N)$, which until the middle to late 1980s was as close to optimal as could be obtained. It is important to recognize, however, that since that time methods have been developed by which elliptic equations can be solved in only $\mathcal{O}(N)$ total arithmetic and in addition, ADI in a form that can be rigorously treated by the full theory presented here and in the cited references is applicable only to 2-D problems. Thus, in recent years it has come to be seldom used. Our purpose for presenting a fairly detailed treatment here arises from two considerations. First, it is a very natural procedure—one that might readily be developed by researchers not familiar with the history and proven theoretical performance of the method. Second, in developing certain parts of the theory we will introduce ideas that are useful elsewhere.

There are three main topics associated with study of ADI in a mathematical context, and which we will treat in subsequent subsections. The first is convergence (and convergence rate) in the case of using a constant iteration parameter (analogous to a pseudo-time step size), and selection of an optimal parameter. Second is convergence for the case of variable iteration parameters (so-called *cyclic ADI*) and prediction of (nearly) optimal sequences of such parameters. Finally, we will consider the "commutative" and "noncommutative" cases of ADI. This is important because the former occurs only in *separable* problems which can be solved analytically, or at least with a fast Poisson solver. The noncommutative case, which therefore is more important for applications, unfortunately does not have the rigorous theoretical foundation of the commutative case. Nevertheless, ADI tends to perform rather well, even for the more practical noncommutative problems in two space dimensions.

2.3.1 ADI with a single iteration parameter

In this subsection we will begin by presenting the form of problem to be considered throughout this treatment of ADI. We will see that it is slightly more restrictive than was the case for SOR, but recall that our theory for SOR held rigorously only for 5-point (7-point in 3D) discrete operators. Following this we will derive the basic fixed-point iteration formula for single-parameter ADI, and then state and prove a theorem regarding convergence of the iterations. We will then derive the optimal parameter value for this iteration scheme and from this compute the asymptotic convergence rate for the method.

The form of the PDE to be studied in this section is

$$-(au_x)_x - (au_y)_y + su = f\,, \qquad (x,y) \in \Omega\,, \tag{2.67}$$

where $\Omega \subset \mathbb{R}^2$ is a bounded rectangle. We will also assume that Dirichlet boundary conditions are applied on all of $\partial\Omega$, although we observe that the theory can be developed for other typical boundary conditions. Also note that a, s and f may all be functions of (x, y).

When Eq. (2.67) is discretized in the usual way (see Chap. 3 for methods to discretize the *self-adjoint form* operators in this expression) with centered difference approximations the resulting system of linear algebraic equations takes the form

$$Au = b\,, \tag{2.68}$$

which is again just Eq. (2.3). Thus, A has the structure of a discrete Laplacian, but it is not necessarily constant coefficient.

ADI as a fixed-point iteration

We now decompose the matrix A as

$$A = H + V + S\,, \tag{2.69}$$

where each of the matrices on the right-hand side is $N\times N$, and each comes from a specific term in (2.67). The matrix H arises in the discretization of the x-derivative term, V from the y-derivative term and S from the zeroth-order term. For a natural ordering of A in which the vertical index is varied first, these matrices have structures as depicted in Fig. 2.11. We note that the form of H can be transformed to the

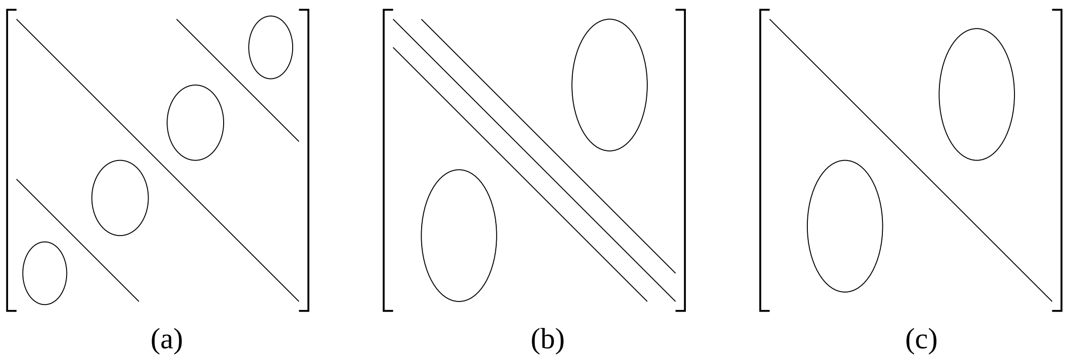

Figure 2.11: Matrices arising from decomposition of A: (a) H matrix, (b) V matrix, (c) S matrix.

standard compact tridiagonal form shown for V by suitable permutations of rows and columns. However, this is not necessary for implementations because grid function values in only one row (or one column) of the grid are found at a time (as in SLOR), and typical algorithms then automatically use the form of the matrix in part (b) of the figure. Furthermore, it is possible to construct a form of sparse LU decomposition that directly treats the matrix structure shown in part (a) of the figure, but this is seldom done.

If we now introduce (2.69) into (2.68) we obtain

$$(H + V + S)u = b\,.$$

We wish to construct an iteration scheme for solving this system. An obvious one corresponding to a rearrangement of the *Richardson method* (see, *e.g.*, Hackbusch [23]) is

$$u = (I - H - V - S)u + b\,. \tag{2.70}$$

But to make use of our decomposition into tridiagonal matrices we write this as

$$(I + H + \theta S)u = (I - V - (1-\theta)S)u + b\,.$$

Furthermore, it is clear from the form of (2.70) that we can easily introduce a factor r multiplying I which is analogous to a time-step parameter in a pseudo-transient formalism. In the present case this will be our iteration parameter. Also, we note that $\theta = 1/2$ is usually employed in the present context, so the above can be written as

$$\left(rI + H + \frac{1}{2}S\right)u = \left(rI - V - \frac{1}{2}S\right)u + b\,. \tag{2.71}$$

But we can write an analogous equation that employs the vertical operator on the left-hand side; namely,

$$\left(rI + V + \frac{1}{2}S\right)u = \left(rI - H - \frac{1}{2}S\right)u + b\,. \tag{2.72}$$

If we were to use only one or the other of (2.71), (2.72) we would obtain a method quite similar (but not identical) to SLOR. By using both expressions alternately (from one iteration to the next) we obtain Peaceman–Rachford ADI [22]. In particular, we define an intermediate result u^* obtained from (2.71), and write the complete iteration scheme as

$$\left(rI + H + \frac{1}{2}S\right)u^* = \left(rI - V - \frac{1}{2}S\right)u^{(n)} + b\,, \tag{2.73a}$$

$$\left(rI + V + \frac{1}{2}S\right)u^{(n+1)} = \left(rI - H - \frac{1}{2}S\right)u^* + b\,. \tag{2.73b}$$

It can readily be seen from the pseudo-transient viewpoint that this scheme is convergent for and $r > 0$ due to the unconditional stability of the *Crank–Nicolson method* to which it is equivalent. We will prove this here in a different manner, making use of the theory of linear fixed-point iteration as we have done for SOR, because this will lead us to a formula for optimal r. This is not available from analysis of the pseudo-transient formalism. We also note that consistency of Eqs. (2.73) with the original PDE follows directly from the construction of Eq. (2.70). This too is more difficult to obtain from a pseudo-transient analysis.

Convergence of ADI iterations

We begin study of convergence of ADI by defining matrices to simplify notation:

$$H_1 \equiv H + \frac{1}{2}S\,, \qquad V_1 \equiv V + \frac{1}{2}S\,,$$

and write Eqs. (2.73) as

$$(rI + H_1)u^* = (rI - V_1)u^{(n)} + b\,,$$
$$(rI + V_1)u^{(n+1)} = (rI - H_1)u^* + b\,.$$

Formal solution of these equations followed by substitution of the former into the latter yields

$$u^{(n+1)} = (rI + V_1)^{-1}(rI - H_1)\left\{(rI + H_1)^{-1}\left[(rI - V_1)u^{(n)} + b\right]\right\} + (rI + V_1)^{-1}b\,,$$

or

$$u^{(n+1)} = T_r u^{(n)} + k_r\,, \tag{2.74}$$

where

$$T_r \equiv (rI + V_1)^{-1}(rI - H_1)(rI + H_1)^{-1}(rI - V_1) \tag{2.75}$$

is the Peaceman–Rachford iteration matrix, and

$$k_r \equiv (rI + V_1)^{-1}\left[(rI - H_1)(rI + H_1)^{-1} + I\right]b\,. \tag{2.76}$$

We see that (2.74) is of exactly the same form as all of the basic iteration schemes considered so far, now with $G = T_r$.

Thus, we would expect that to study convergence of the iterations of (2.74) we need to estimate $\rho(T_r)$. To do this we first use a similarity transformation to define

$$\begin{aligned}\widetilde{T}_r &\equiv (rI + V_1)T_r(rI + V_1)^{-1}\\ &= (rI - H_1)(rI + H_1)^{-1}(rI - V_1)(rI + V_1)^{-1}\,,\end{aligned}$$

which is similar to T_r and thus has the same spectral radius. Hence, we have

$$\begin{aligned}\rho(T_r) = \rho(\widetilde{T}_r) &\le \|\widetilde{T}_r\|\\ &\le \|(rI - H_1)(rI + H_1)^{-1}\|\|(rI - V_1)(rI + V_1)^{-1}\|\,.\end{aligned}$$

We can now state the basic theorem associated with single-parameter ADI iterations.
Theorem 2.10 *Let H_1 and V_1 be $N\times N$ Hermitian non-negative definite matrices with at least one being positive definite. Then $\rho(T_r) < 1\ \forall\ r > 0$.*

Proof. Since H_1 is Hermitian its eigenvalues are real, and H_1 is diagonalizable. The same is true for $rI - H_1$ and $rI + H_1$. Moreover, since H_1 is non-negative definite, $\lambda_j \ge 0$ holds $\forall\ \lambda_j \in \sigma(H_1)$. Now if $\lambda_j \in \sigma(H_1)$ it follows that $r - \lambda_j \in \sigma(rI - H_1)$, and $r + \lambda_j \in \sigma(rI + H_1)$. Furthermore, since $rI + H_1$ is diagonalizable, $(r + \lambda_j)^{-1} \in \sigma\left((rI + H_1)^{-1}\right)$. Finally, it follows from this and a direct calculation that the eigenvalues of $(rI - H_1)(rI + H_1)^{-1}$ are $(r - \lambda_j)/(r + \lambda_j)$. Thus, taking $\|\cdot\|$ to be the spectral norm leads to

$$\|(rI - H_1)(rI + H_1)^{-1}\| = \max_{1\le j\le N}\left|\frac{r - \lambda_j}{r + \lambda_j}\right|\,.$$

Clearly, the quantity on the right is less than unity for any $\lambda_j > 0$ and $\forall\ r > 0$. The same arguments and conclusions apply for $\|(rI - V_1)(rI + V_1)^{-1}\|$, completing the proof.

ADI optimal parameter

We now consider the problem of choosing an optimal parameter value r for the ADI iterations. In the context of a pseudo-transient formalism, and in light of the manner in which r enters the iteration formulas, *e.g.*, (2.73), we would expect r to be rather small since it is roughly the reciprocal of the pseudo-time step. In principle this would seem to produce the fastest convergence to steady state.

To permit a direct comparison of convergence rate (to be obtained as part of the optimal parameter analysis) with results obtained earlier for SOR, we restrict attention to the Laplace–Dirichlet problem. But we emphasize, as remarked above, that the analyses presented here can be conducted for somewhat more general problems; we leave this as an exercise for the reader.

We will employ a uniform grid in both directions on the unit square so that $N_x = N_y$, leading to $h_x = h_y = h = 1/(N_x - 1)$. We will also assume that the Dirichlet conditions have been eliminated from the discrete equations resulting in only $N_x - 2$ points in each direction at which solutions are sought. This implies that the system matrix A will be of size $(N_x - 2)^2$, and correspondingly this number of eigenvectors will be needed to span the space on which this matrix acts.

Now suppose that $\alpha^{(k,\ell)}$ is one such eigenvector, and that its $(i,j)^{th}$ component can be represented as

$$\alpha_{i,j}^{(k,\ell)} = \gamma_{k,\ell} \sin\left(\frac{(k-1)(i-1)\pi}{N_x - 1}\right) \sin\left(\frac{(\ell-1)(j-1)\pi}{N_x - 1}\right), \tag{2.77}$$

$$2 \le i,j \le N_x - 1\,, \quad 2 \le k,\ell \le N_x - 1\,.$$

Here, $\gamma_{k,\ell}$ is a constant that can ultimately be determined via normalization, but in analyses such as presented here, it actually cancels.

If we form the eigenvalue problems for H_1 and V_1 and substitute $\alpha_{i,j}^{(k,\ell)}$ for the eigenvector, we can easily show that the eigenvalues of H_1 ($= H$ in this case) are

$$\lambda_k = 2 - 2\cos\left(\frac{(k-1)\pi}{N_x - 1}\right) \qquad k = 2, 3, \ldots, N_x - 1\,, \tag{2.78}$$

and those of V_1 are

$$\lambda_\ell = 2 - 2\cos\left(\frac{(\ell-1)\pi}{N_x - 1}\right) \qquad \ell = 2, 3, \ldots, N_x - 1\,. \tag{2.79}$$

We note here that the choice of Eq. (2.77) for the form of the eigenvector component is motivated in the same way is in the Jacobi iteration analysis done earlier.

We can use a trigonometric identity to express these as, for example,

$$\lambda_k = 4\sin^2\left(\frac{(k-1)\pi}{2(N_x - 1)}\right).$$

It then follows from the ADI convergence theorem that the eigenvalues of the Peaceman–Rachford iteration matrix T_r are

$$\lambda_{k,\ell} = \left[\frac{r - 4\sin^2\left(\frac{(k-1)\pi}{2(N_x-1)}\right)}{r + 4\sin^2\left(\frac{(k-1)\pi}{2(N_x-1)}\right)}\right] \left[\frac{r - 4\sin^2\left(\frac{(\ell-1)\pi}{2(N_x-1)}\right)}{r + 4\sin^2\left(\frac{(\ell-1)\pi}{2(N_x-1)}\right)}\right], \tag{2.80}$$

and from this we see (for our special case of $N_x = N_y$ and uniform h) that

$$\rho(T_r) = \left[\max_{2 \le k \le N_x - 1} \left|\frac{r - 4\sin^2\left(\frac{(k-1)\pi}{2(N_x-1)}\right)}{r + 4\sin^2\left(\frac{(k-1)\pi}{2(N_x-1)}\right)}\right|\right]^2. \tag{2.81}$$

We leave as an exercise to the reader derivation of the analogous formula for the more general cases.

We now want to choose r so as to minmize $\rho(T_r)$. To do this we consider the function

$$g(\xi, r) = \frac{r - \xi}{r + \xi}\,, \qquad r > 0\,, \quad \xi \in [\xi_1, \xi_2]\,,$$

for $\xi_1 > 0$. For the given r and ξ intervals, a direct calculation (left as an exercise for the reader) shows that $\partial g/\partial \xi < 0$, strictly; so g is monotone decreasing. Thus, $\max |g|$ occurs at one of the endpoints of the ξ interval. That is,

$$\max_{\xi \in [\xi_1, \xi_2]} |g(\xi, r)| = \max\left[\left|\frac{r - \xi_1}{r + \xi_1}\right|, \left|\frac{r - \xi_2}{r + \xi_2}\right|\right] \tag{2.82}$$

for each fixed r. Using this, it is easily checked that

$$\max_{\xi\in[\xi_1,\xi_2]} |g(\xi,r)| = \begin{cases} \frac{\xi_2-r}{\xi_2+r} & 0 < r \le \sqrt{\xi_1\xi_2} \\ \frac{r-\xi_1}{r+\xi_1} & r \ge \sqrt{\xi_1\xi_2}\,. \end{cases} \tag{2.83}$$

This provides an expression for $\rho(T_r)$ as a function only of r, and a direct calculation shows that

$$\min_{r>0}\max_{\xi\in[\xi_1,\xi_2]} |g(\xi,r)| = \frac{1-\sqrt{\xi_1/\xi_2}}{1+\sqrt{\xi_1/\xi_2}}\,, \tag{2.84}$$

which implies (upon multiplying the numerator and the denominator by ξ_2) that

$$r_{opt} = \sqrt{\xi_1\xi_2}\,. \tag{2.85}$$

Now recall from the definition of $g(\xi,r)$ and the form of $\rho(T_r)$ that $[\xi_1,\xi_2]$ is the spectral interval for the eigenvalues in each direction for the given problem. Thus, in the present case we have

$$\xi_1 = 4\sin^2\left(\frac{\pi}{2(N_x-1)}\right) = 4\sin^2\left(\frac{\pi}{2}h\right) \tag{2.86}$$

and

$$\xi_2 = 4\sin^2\left(\frac{(N_x-2)\pi}{2(N_x-1)}\right) = 4\cos^2\left(\frac{\pi}{2(N_x-1)}\right) = 4\cos^2\left(\frac{\pi}{2}h\right) \tag{2.87}$$

We can now substitute these into Eq. (2.85) to obtain an explicit expression for the optimal iteration parameter value in terms of the spatial step size:

$$r_{opt} = 4\sin\left(\frac{\pi}{2}h\right)\cos\left(\frac{\pi}{2}h\right)\,. \tag{2.88}$$

In the limit $h\to 0$ this is approximately $2\pi h$, which is small; so our intuition regarding the size of pseudo-time steps is roughly correct.

We can also use the above values of the ξs in (2.84) to find that

$$\min_{r>0}\rho(T_r) = \left[\frac{1-\tan\left(\frac{\pi}{2}h\right)}{1+\tan\left(\frac{\pi}{2}h\right)}\right]^2\,,$$

which can be rearranged to yield

$$\rho(T_r) = \frac{1-2\sin\frac{\pi}{2}h\cos\frac{\pi}{2}h}{1+2\sin\frac{\pi}{2}h\cos\frac{\pi}{2}h} \tag{2.89}$$

for optimal iteration parameter r. One can check that for small h (neglecting $\mathcal{O}(h^3)$ terms) we have

$$\rho(T_r) \simeq \frac{1-\pi h}{1+\pi h} \simeq 1-2\pi h+\mathcal{O}(h^2)\,. \tag{2.90}$$

But since $\log(1-2\pi h)\cong -2\pi h$ for sufficiently small h, it follows from the definition of asymptotic convergence rate that $R(T_r)\cong 2\pi h$ for optimal r; this is precisely the same result obtained for optimal SOR. But one must recognize that the amount of work per iteration is greater by more than a factor of two for ADI; indeed, it is almost exactly a factor of two greater than that required by SLOR—which has a higher convergence rate, and can be extended to the 3-D case in a natural way. We see from this why Peaceman–Rachford ADI is no longer widely used, despite its rather intuitively appealing traits.

2.3.2 ADI: the commutative case

In this section we will outline the analysis of Peaceman–Rachford ADI for the special case when $H_1V_1 = V_1H_1$. We will see that under this assumption (and not otherwise, in general) it is possible to generate (nearly) optimal sequences of iteration parameters whose use embodies the *cyclic ADI* procedures. We will see that these *nonstationary* iteration methods (ones whose iteration parameters change from one iteration to the next) are significantly more efficient than is the single-parameter version discussed in the preceding section. We will begin the treatment with the basic iteration formula construction followed by a major theorem associated with such procedures. We will then sketch the process of determining the optimal parameter sequences and give two widely-used specific examples of these.

Cyclic ADI iteration formula

We begin by noting that since $\rho(T_r) < 1 \ \forall\ r > 0$ in our single-parameter formulation of ADI, there is no reason to use a single parameter in Eqs. (2.73). Thus, in place of these equations we here consider

$$(r_{n+1}I + H_1)u^* = (r_{n+1}I - V_1)u^{(n)} + b\,, \tag{2.91a}$$

$$(r_{n+1}I + V_1)u^{(n+1)} = (r_{n+1}I - H_1)u^* + b\,. \tag{2.91b}$$

In fact, we could even use different values of r in the individual steps of these equations, as might also have been done in the *stationary* iteration procedure of the previous section (see, *e.g.*, Axelsson [24] for an example of this).

After n iterations of this procedure the corresponding representation of the iteration matrix is

$$\prod_{k=1}^{n} T_{r_k}\,,$$

rather than simply T_r^n as would be the case for the stationary single-parameter case. Intuitively, we would expect

$$\rho\left(\prod_{k=1}^{n} T_{r_k}\right) < 1 \quad \forall\ n \ge 1$$

since we have shown that $\rho(T_r) < 1 \ \forall\ r > 0$. But this is a naive expectation, and there are subtleties that prevent proving such a result. In fact, counterexamples for which convergence does not occur can be found when the matrices H_1 and V_1 do not commute (see [15]). To guarantee convergence it is necessary that the same orthonormal basis of eigenvectors spans the eigenspace of both H_1 and V_1. In this case, the following theorem can be proven (see Varga [17]).

Theorem 2.11 *Let H_1 and V_1 be $N\times N$ Hermitian positive definite matrices with $H_1V_1 = V_1H_1$. Then for any set $\{r_k\}_{k=1}^n$ of positive real numbers we have*

$$\left\|\prod_{k=1}^{n} T_{r_k}\right\| = \rho\left(\prod_{k=1}^{n} T_{r_k}\right) = \max_{1<i,j\le N} \prod_{k=1}^{n} \left|\frac{r_k - \sigma_i}{r_k + \sigma_i}\right| \left|\frac{r_k - \tau_j}{r_k + \tau_j}\right| < 1\,, \tag{2.92}$$

where $\sigma_i \in \sigma(H_1)$, $\tau_j \in \sigma(V_1)$. Moreover, if $r_{n+1} > 0$, then

$$\left\|\prod_{k=1}^{n+1} T_{r_k}\right\| < \left\|\prod_{k=1}^{n} T_{r_k}\right\|\,.$$

The norm in (2.92) and following is the spectral norm.

The last inequality above implies that the error is decreased by increasing amounts for each succeeding iteration within a parameter sequence. In fact, if $\sigma(H_1)$ and $\sigma(V_1)$ were completely known, it would be possible to construct a direct (non-iterative) solution procedure by choosing an appropriate sequence $\{r_k\}_{k=1}^n$. We remark that this also occurs for *Chebychev acceleration* (not treated herein, but see *e.g.*, [24]), and it is our first indication that cyclic ADI represents a significant improvement over the single-parameter version.

In general, we seldom have available the complete spectra for H_1 and V_1. There are, however techniques whereby we can obtain bounds on the eigenvalues, leading to useful approximations of the parameter sets $\{r_k\}_{k=1}^n$. In particular, it is often possible to estimate α and β such that

$$0 < \alpha \le \sigma_i, \tau_j \le \beta\,, \qquad i,j = 1,2,\ldots,N\,.$$

Then we have

$$\begin{aligned}\max_{1\le i,j\le N} \prod_{k=1}^n \left|\frac{r_k-\sigma_i}{r_k+\sigma_i}\right| \left|\frac{r_k-\tau_j}{r_k+\tau_j}\right| &\le \left[\max_{1\le i\le N} \prod_{k=1}^n \left|\frac{r_k-\sigma_i}{r_k+\sigma_i}\right|\right]\cdot\left[\max_{1\le j\le N} \prod_{k=1}^n \left|\frac{r_k-\tau_j}{r_k+\tau_j}\right|\right] \\ &\le \left[\max_{\xi\in[\alpha,\beta]} \prod_{j=1}^n \left|\frac{r_j-\xi}{r_j+\xi}\right|\right]^2 .\end{aligned}$$

From this it follows that

$$\left\|\prod_{j=1}^n T_{r_j}\right\| \le \left[\max_{\xi\in[\alpha,\beta]} |g_n(\xi,r)|\right]^2 ,$$

where

$$g_n(\xi,r) \equiv \prod_{j=1}^n \left(\frac{r_j-\xi}{r_j+\xi}\right) .$$

It is clear that this is now in the same form as that treated earlier in the single-parameter case, and analogous to what was done there, we now seek a set of r_js that will minimize the maximum of $|g_n|$ with respect to $\xi \in [\alpha,\beta]$. We remark that the exact solution to this min-max problem is not known in general; however, when $n = 2^k$ for any $k \ge 0$, Wachspress [25] has provided an elegant exact result, as described in [17]. Here, we will present results from an approximate solution to this problem in the form of "nearly" optimal r_js as well as the required number of them—all as a function of α, β and the mesh size h.

To begin, we divide the spectral interval $[\alpha,\beta]$ into n subintervals (noting that n has not as yet been prescribed) such that

$$\alpha = \xi_0 < \xi_1 < \xi_2 < \cdots < \xi_n = \beta\,.$$

Then on each of these subintervals we consider the <u>one-parameter</u> min-max problem

$$\min_{r_j}\left[\max_{\xi\in[\xi_{j-1},\xi_j]} \left|\frac{r_j-\xi}{r_j+\xi}\right|\right] ,$$

which is identical to the single-parameter problem of the preceding section, except that it is posed on only a subset of the spectral interval. Hence, we already know the solution to this problem is $r_j = \sqrt{\xi_{j-1}\xi_j}$ for each j^{th} subinterval. Our task now is to find the ξ_js and n, the number of r_js.

First, analogous to Eq. (2.84) we have

$$\min_{r_j}\left[\max_{\xi\in[\xi_{j-1},\xi_j]} \left|\frac{r_j-\xi}{r_j+\xi}\right|\right] = \frac{1-\sqrt{\xi_{j-1}/\xi_j}}{1-\sqrt{\xi_{j-1}/\xi_j}} \equiv \delta \tag{2.93}$$

Now define

$$\gamma \equiv \left(\frac{\beta}{\alpha}\right)^{\frac{1}{2n}} , \tag{2.94}$$

and set

$$\delta = \frac{\gamma-1}{\gamma+1}\,. \tag{2.95}$$

The last of these arises naturally by factoring $\sqrt{\xi_{j-1}/\xi_j}$ from both numerator and denominator of (2.93), and then with $n = 1$ setting $\xi_{j-1} = \alpha$ and $\xi_j = \beta$.

We can now use these formulas recursively to find the ξ_js and then the r_js. In particular, from eq93:ch2, if $n \neq 1$ but is known and the spectral interval bounds are known, then we can calculate δ and starting with $j = 1$ (for which $\xi_{j-i} = \xi_0 = \alpha$ is known), we can solve for ξ_1. Then we recursively apply this procedure to find the remaining ξ_js. It is is possible to derive a very simple formula that accomplishes this directly:

$$\xi_j = \alpha\gamma^{2j} \,. \tag{2.96}$$

We leave this derivation as an exercise for the reader.

The only remaining unknown associated with the sequence of iteration parameters is the number of them, n. This is derived in [17], and here we will only give the result; but we note that the basic idea underlying the derivation is to choose n so as to maximize the average convergence rate over a cycle of length n. This leads to the formula

$$n \geq \frac{1}{2}\frac{\ln\left(\frac{\beta}{\alpha}\right)}{\ln\left(\frac{1}{\sqrt{2}-1}\right)} \,. \tag{2.97}$$

We can now find the (nearly) optimal parameter sequence using the formula

$$r_j = \alpha\gamma^{2j-1} \,, \qquad j = 1, 2, \ldots, n \,, \tag{2.98}$$

which follows directly from $r_j = \sqrt{\xi_{j-1}/\xi_j}$. This sequence is due to Peaceman and Rachford [22]. There are numerous similar sequences, derived via modifications of the technique outlined above. One of the more successful ones is due to Wachspress [25]:

$$n \geq 1 + \frac{1}{2}\frac{\ln\left(\frac{\beta}{\alpha}\right)}{\ln\left(\frac{1}{\sqrt{2}-1}\right)} \,, \tag{2.99}$$

and

$$r_j = \beta\left(\frac{\alpha}{\beta}\right)^{(j-1)/(n-1)} \,, \qquad j = 1, 2, \ldots, n \,, \tag{2.100}$$

In both of these expressions for n, the right-hand side will obviously not produce an integer result. But Theorem 1.11 suggests that longer sequences are preferred to shorter ones; so the rule is to use the first integer larger than the decimal number produced by the formula.

There is one final calculation that must be demonstrated before it is possible to actually produce the above iteration parameters. It is the determination of α and β. The approach is actually equivalent to what we did earlier in the single parameter case, but we will present it somewhat more formally here.

Theorem 2.12 *Let A be a $(N-2)\times(N-2)$ tridiagonal matrix arising from discretization of a Laplace–Dirichlet problem on a grid of uniform spacing $h = 1/(N-1)$. Let the main diagonal of A consist of the nonzero elements a and the two off-diagonals contain elements b. Then the eigenvalues of A are*

$$\lambda_j = a + 2b\cos((j-1)\pi h) \,, \qquad j = 2, 3, \ldots, N-1 \,. \tag{2.101}$$

The proof of this is the same as the construction employed earlier for eignenvalues of the Jacobi iteration matrix.

Mitchell and Griffiths [16] state, without proof, the following result for eigenvalues of nonsymmetric matrices.

$$\lambda_j = a + 2\sqrt{bc}\cos((j-1)\pi h) \,, \tag{2.102}$$

where b and c are off-diagonal elements, and the rest of the problem statement is the same as in the theorem above.

2.3.3 ADI: the noncommutative case

In the preceding section it was necessary to use the assumption $H_1V_1 = V_1H - 1$ in order to prove convergence of cyclic ADI, and we recall that it was mentioned that ADI may not converge without satisfaction of this requirement. In this section we will briefly consider under what conditions this commutativity assumption can be expected to hold. All of our detailed analyses to this point have involved the relatively simple Laplace–Dirichlet problem. We will now return to Eq. (2.67) which we here generalize slightly and write as

$$-(a_1(x,y)u_x)_x - (a_2(x,y)u_y)_y + s(x,y)u = f(x,y)\,, \qquad (x,y) \in \Omega \subset \mathbb{R}^2\,,$$

with Dirichlet conditions

$$u(x,y) = g(x,y)\,, \qquad (x,y) \in \partial\Omega\,.$$

It will be assumed that $a_1, a_2 \in \mathrm{C}^1(\Omega)$, $a_1, a_2, s \in \mathrm{C}(\overline{\Omega})$, and $a_1, a_2, s > 0$ in $\overline{\Omega}$. The following theorem is the main result on commutativity with respect to ADI procedures.

Theorem 2.13 *Let a_1, a_2 and s be as above. Then except for the case $a_1(x,y) = \varphi_1(x)$, $a_2(x,y) = \varphi_2(y)$, $s(x,y) \equiv$ const. and $\overline{\Omega}$ a rectangle, the matrices H_1 and V_1 $\underline{\text{fail to commute}}$.*

This is a well-known result; the proof can be found in [15] and [17], for example, and we will not repeat it here. It is clear from this that the entire theory of cyclic ADI no longer holds in most practical cases. In particular, even if we begin with a constant-coefficient problem on a nonrectangular domain, by the time a coordinate transformation is performed to obtain a rectangular domain (see Chap. 4 for discussions of this), as required by the theorem, we will in general no longer have the "separable" coefficients that are required.

This, and the fact that ADI does not have a natural extension to 3-D problems, makes it difficult to recommend its use. As stated at the beginning of this section, our main goal here has been to provide sufficient information on this rather natural approach to prevent its being re-invented. Indeed, even if the commutativity problem were not so severe, the $\mathcal{O}(N \log N)$ arithmetic required by cyclic ADI still is more than needed for modern methods such as multigrid and domain decomposition.

2.4 Incomplete LU Decomposition (ILU)

In this section we will treat a class of methods that at one time appeared to show great promise for solving elliptic PDEs, especially in the context of practical engineering problems. However, as more demanding problems were attempted, especially when using highly-refined gridding, it became apparent that the method had no better convergence rate than either point SOR or single-parameter ADI, although it was typically more robust; but it required significantly more arithmetic per iteration. With the increased understanding that has accumulated in recent years, it is evident that *incomplete LU decompositions* (ILUs) are useful mainly as preconditioners (to be discussed in the next section) for other iterative techniques. They should generally not be used as "stand-alone" solvers.

In what follows we will begin with a basic general treatment of ILUs. Following this we provide details for the first (and probably best-known in the engineering literature) of these methods. We remark that there are many variants of what we will present. For a rather different approach, but embodying the same basic ideas, the reader is referred to Manteuffel [26] and Van der Voorst [27]; also see [23] and [24].

2.4.1 Basic ideas of ILU decomposition

We first observe that all of the methods we have considered up to this point could, in a general way, be classified as ILUs. But as will be evident as we proceed, most would be rather trivial in this context, and they are not usually included within the present framework. The basic ideas underlying construction of ILUs are as follows.

Suppose we wish to solve the linear system Eq. (2.3) repeated here as

$$Au = b\,, \tag{2.103}$$

where A is a sparse banded (but not compactly) $N \times N$ nonsingular matrix. Because A is not compactly banded, at least $\mathcal{O}(N^2)$ arithmetic operations would be required by an elimination procedure, and if standard methods (*e.g.*, Gaussian elimination) are employed $\mathcal{O}(N^3)$ operations would be needed. Moreover, $\mathcal{O}(N^2)$ words of storage would be necessary to handle the fill-in between bands during the elimination process.

We can avoid both the arithmetic and storage problems by replacing A with a matrix M that is "nearly as sparse" as A, but which admits a $\underline{\text{sparse}}$ LU decomposition, and then construct a fixed-point iteration within this context. In particular, we set

$$M = A + E\,, \tag{2.104}$$

where E is an error (sometimes termed "defect") matrix. Then the original system (2.103) can be written as

$$(M - E)u = b$$

or

$$Mu = Eu + b\,,$$

which immediately suggests the fixed-point form

$$Mu^{(n+1)} = Eu^{(n)} + b\,. \tag{2.105}$$

Now since M has a sparse LU decomposition, we can express the above as

$$LUu^{(n+1)} = Eu^{(n)} + b\,,$$

or

$$u^{(n+1)} = U^{-1}L^{-1}Eu^{(n)} + U^{-1}L^{-1}b\,, \tag{2.106}$$

and, in principle, efficiently perform the indicated iterations. Clearly, this is in our standard linear fixed-point form

$$u^{(n+1)} = Gu^{(n)} + k\,,$$

with

$$G \equiv U^{-1}L^{-1}E\,, \qquad k \equiv U^{-1}L^{-1}b\,.$$

It follows that any such procedure will be convergent from arbitrary initial guesses if, and only if,

$$\rho(U^{-1}L^{-1}E) < 1\,. \tag{2.107}$$

As noted earlier, there are many different variants of ILU decomposition, but all are constructed so as to exhibit the following properties:

i) M should be "nearly" as sparse as A and should have an easily constructed sparse LU decomposition.

ii) E should be easily constructed and should be at least as sparse as M.

iii) $\rho(M^{-1}E) < 1$ must hold.

Clearly, the first two of these guarantee that arithmetic per iteration will remain manageable while the last implies convergence of the iterations.

2.4.2 The strongly implicit procedure (SIP)

Although some of the basic ideas if ILU can be traced to the Russian literature of the early 1960s, the first practical implementation was due to Stone [28] in the U. S. in 1968. This technique came to be known as the *strongly implicit procedure* (SIP) and generated considerable research during the 1970s and 1980s. Rubin and various co-workers (see, *e.g.*, Rubin and Khosla [29] and Khosla and Rubin [30]) applied the method to numerous problems associated with the Navier–Stokes equations of fluid motion, and Schneider and Zedan [31] introduced modifications in applications to heat transfer problems. In our treatment here we will present Stone's original version of SIP. The method used by Rubin is obtained by choosing a zero value for the iteration parameter α.

We now consider a general elliptic problem of the form

$$a_1(x,y)u_{xx} + a_2(x,y)u_{yy} + a_3(x,y)u_x + a_4(x,y)u_y + a_5(x,y)u = f(x,y)\,, \qquad (x,y)\in\Omega\,, \tag{2.108}$$

$a_1, a_2 < 0$, and $a_5 > 0$ uniformly in Ω, with boundary conditions

$$\alpha(x,y)u_x + \beta(x,y)u_y + \gamma(x,y)u = g(x,y)\,, \qquad (x,y)\in\partial\Omega\,. \tag{2.109}$$

We first observe that this problem is somewhat more general than those explicitly treated by earlier methods, but at least for SOR a theory can be developed for problems of the above form. It is also important to recall that there is no sparse LU decomposition corresponding to the matrix arising from the usual 5-point, second-order approximation to (2.108):

$$A^{(1)}_{i,j}u_{i-1,j} + A^{(2)}_{i,j}u_{i,j-1} + A^{(3)}_{i,j}u_{i,j} + A^{(4)}_{i,j}u_{i,j+1} + A^{(5)}_{i,j}u_{i+1,j} = f_{i,j}\,. \tag{2.110}$$

On the other hand, it is possible to construct a sparse LU decomposition for the 7-band matrix shown in Fig. 2.12, part (a). The two bands shown as dashed lines correspond to grid points marked as open circles

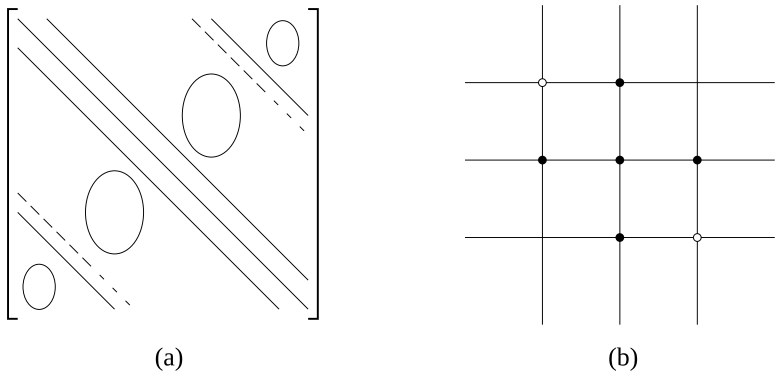

Figure 2.12: (a) 7-band finite-difference matrix; (b) corresponding mesh star.

for the mesh star in part (b) of the figure.

We will now construct Stone's SIP for a small number of grid points to show in detail how ILUs are formulated. We consider the 3×4 grid displayed in Fig. 2.13 in which all points are taken to be in the interior of the solution domain.

The matrix A obtained by collecting all equations of (2.110) and re-indexing as in Fig 2.13 appears as

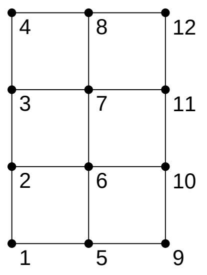

Figure 2.13: Finite-difference grid for demonstrating structure of SIP matrices.

follows:

$$A = \begin{bmatrix}
A_1^{(3)} & A_1^{(4)} & 0 & 0 & A_1^{(5)} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
A_2^{(2)} & A_2^{(3)} & A_2^{(4)} & 0 & 0 & A_2^{(5)} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & A_3^{(2)} & A_3^{(3)} & A_3^{(4)} & 0 & 0 & A_3^{(5)} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & A_4^{(2)} & A_4^{(3)} & 0 & 0 & 0 & A_4^{(3)} & 0 & 0 & 0 & 0 \\
A_5^{(1)} & 0 & 0 & 0 & A_5^{(3)} & A_5^{(4)} & 0 & 0 & A_5^{(3)} & 0 & 0 & 0 \\
0 & A_6^{(1)} & 0 & 0 & A_6^{(2)} & A_6^{(3)} & A_6^{(4)} & 0 & 0 & A_6^{(5)} & 0 & 0 \\
0 & 0 & A_7^{(1)} & 0 & 0 & A_7^{(2)} & A_7^{(3)} & A_7^{(4)} & 0 & 0 & A_7^{(5)} & 0 \\
0 & 0 & 0 & A_8^{(1)} & 0 & 0 & A_8^{(2)} & A_8^{(3)} & 0 & 0 & 0 & A_8^{(5)} \\
0 & 0 & 0 & 0 & A_9^{(1)} & 0 & 0 & 0 & A_9^{(3)} & A_9^{(4)} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & A_{10}^{(1)} & 0 & 0 & A_{10}^{(2)} & A_{10}^{(3)} & A_{10}^{(4)} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & A_{11}^{(1)} & 0 & 0 & A_{11}^{(2)} & A_{11}^{(3)} & A_{11}^{(4)} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & A_{12}^{(1)} & 0 & 0 & A_{12}^{(2)} & A_{12}^{(3)}
\end{bmatrix} .$$

Corresponding to this matrix we define the lower and upper triangular matrices shown below. It is important to note that in SIP (but not necessarily in all ILUs) these matrices have the same structure as the corresponding part of the original matrix.

$$L = \begin{bmatrix}
d_1 & 0 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & 0 \\
c_2 & d_2 & 0 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & 0 \\
0 & c_3 & d_3 & \ddots & \ddots & & & & & & \vdots & \vdots \\
0 & \ddots & c_4 & d_4 & \ddots & \ddots & & & & & \vdots & \vdots \\
b_5 & \ddots & \ddots & c_5 & d_5 & \ddots & \ddots & & & & \vdots & \vdots \\
0 & b_6 & \ddots & \ddots & c_6 & d_6 & \ddots & \ddots & & & \vdots & \vdots \\
\vdots & 0 & b_7 & \ddots & \ddots & c_7 & d_7 & \ddots & \ddots & & \vdots & \vdots \\
\vdots & \vdots & \ddots & b_8 & \ddots & \ddots & c_8 & d_8 & \ddots & \ddots & \vdots & \vdots \\
\vdots & \vdots & & \ddots & b_9 & \ddots & \ddots & c_9 & d_9 & \ddots & 0 & 0 \\
\vdots & \vdots & & & \ddots & b_{10} & \ddots & \ddots & c_{10} & d_{10} & 0 & 0 \\
0 & 0 & \cdots & \cdots & \cdots & 0 & b_{11} & \ddots & \ddots & c_{11} & d_{11} & 0 \\
0 & 0 & \cdots & \cdots & \cdots & \cdots & 0 & b_{12} & 0 & 0 & c_{12} & d_{12}
\end{bmatrix} ,$$

and

$$
U = \begin{bmatrix}
1 & e_1 & 0 & 0 & f_1 & 0 & \cdots & \cdots & \cdots & \cdots & 0 & 0 \\
0 & 1 & e_2 & 0 & 0 & f_2 & 0 & \cdots & \cdots & \cdots & 0 & 0 \\
0 & 0 & 1 & e_3 & 0 & 0 & f_3 & \ddots & & & \vdots & \vdots \\
\vdots & 0 & \ddots & 1 & e_4 & 0 & 0 & f_4 & \ddots & & \vdots & \vdots \\
\vdots & \vdots & \ddots & \ddots & 1 & e_5 & 0 & 0 & f_5 & \ddots & \vdots & \vdots \\
\vdots & \vdots & & \ddots & \ddots & 1 & e_6 & 0 & 0 & f_6 & 0 & \vdots \\
\vdots & \vdots & & & \ddots & \ddots & 1 & e_7 & 0 & 0 & f_7 & 0 \\
\vdots & \vdots & & & & \ddots & \ddots & 1 & e_8 & 0 & 0 & f_8 \\
\vdots & \vdots & & & & & \ddots & \ddots & 1 & e_9 & 0 & 0 \\
\vdots & \vdots & & & & & & \ddots & 0 & 1 & e_{10} & 0 \\
0 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & 0 & 1 & e_{11} \\
0 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & 0 & 1
\end{bmatrix} .
$$

We now construct the product LU and compare its structure with that of the original matrix A. As expected, we find the two matrices have the same band structure except for two additional bands immediately inside the outer bands in the LU product. In particular, the structure of this matrix is the same as that indicated in Fig. 2.12 part (a).

$$
LU = \begin{bmatrix}
d_1 & d_1e_1 & 0 & 0 & d_1f_1 & 0 & 0 & 0 & \cdots \\
c_2 & d_2 + c_2e_1 & d_2e_2 & 0 & c_2f_1 & d_2f_2 & 0 & 0 & \cdots \\
0 & c_3 & d_3 + c_3e_2 & d_3e_3 & 0 & c_3f_2 & d_3f_3 & 0 & \cdots \\
0 & 0 & c_4 & d_4 + c_4e_3 & d_4e_4 & 0 & c_4f_3 & d_4f_4 & \ddots \\
b_5 & b_5e_1 & 0 & c_5 & d_5 + c_5e_4 + b_5f_1 & d_5e_5 & 0 & c_5f_4 & \ddots \\
0 & b_6 & b_6e_2 & 0 & c_6 & \ddots & \ddots & 0 & \ddots \\
\vdots & 0 & b_7 & b_7e_3 & \ddots & \ddots & \ddots & \ddots & \ddots \\
\vdots & \vdots & 0 & b_8 & b_8e_4 & \ddots & \ddots & \ddots & \ddots \\
\vdots & \vdots & \vdots & 0 & b_9 & b_9e_5 & \ddots & \ddots & \ddots \\
 & & & & & & & &
\end{bmatrix} .
$$

We can now determine the elements of L and U by matching components of LU with those of A. First, it is clear that

$$
\begin{aligned}
b_i &= A_i^{(1)} , \qquad \forall\ i = 5, \ldots, 12 , \\
c_i &= A_i^{(2)} , \qquad \forall\ i = 2, \ldots, 12 .
\end{aligned}
$$

Then

$$
d_1 = A_1^{(3)} ,
$$

and

$$
e_1 = A_1^{(4)}/d_1 , \qquad f_1 = A_1^{(5)}/d_1 .
$$

Next, we find

$$
d_2 + c_2e_1 = A_2^{(3)} \ \Rightarrow\ d_2 = A_2^{(3)} - c_2e_1 .
$$

In general, the d_is are given by

$$
d_i = A_i^{(3)} - c_ie_{i-1} - b_if_{i-N_y} ,
$$

where, as usual, N_y is the number of points in the y direction. (Note that if we had used row-wise natural ordering, rather than the column-wise ordering employed here, N_y would be replaced with N_x.) The third term in this expression should be evaluated only for $i > N_y$. With this general formula for d_i we are now able to calculate the remaining matrix elements, namely, those of the matrix U:

$$e_i = A_i^{(4)}/d_i\,, \qquad f_i = A_i^{(5)}/d_i\,.$$

Once all of the elements of L and U have been evaluated, the elements of the error matrix can be constructed. These are given by

$$E_i^{(1)} = b_i e_{i-N_y} \qquad \text{and} \qquad E_i^{(2)} = c_i f_{i-1}\,.$$

We now observe that the difference equation (2.110) must be replaced with

$$A_i^{(1)}u_{i-N_y} + E_i^{(1)}u_{i-N_y+1} + A_i^{(2)}u_{i-1} + A_i^{(3)}u_i + A_i^{(4)}u_{i+1} + E_i^{(2)}u_{i+N_y-1} + A_i^{(5)}u_{i+N_y} = f_i\,. \tag{2.111}$$

From our above constructions it is clear that the corresponding matrix $M = A + E$ has a sparse, easily-constructed LU decomposition, and the corresponding error matrix E has only two easily-constructed bands. However, the iterations

$$u^{(n+1)} = M^{-1}Eu^{(n)} + M^{-1}b$$

converge rather slowly. Stone [28] reasoned (heuristically) that if the effects of the error terms could be minimized, the convergence rate might be improved.

Thus, we now present the analysis provided in [28]. This is based on attempting to estimate u_{i-N_y+1} and u_{i+N_y-1} via a crude Taylor series expansion, and using points already contained in the mesh star of Fig. 2.12, part (b). We will carry this out using the original two-index notation; so the grid function values to be approximated are $u_{i-1,j+1}$ and $u_{i+1,j-1}$. Stone [28] provides the following approximations:

$$\begin{aligned} u_{i-1,j+1} &= -u_{i,j} + u_{i,j+1} + u_{i-1,j}\,, \qquad (= -u_i + u_{i+1} + u_{i-N_y}) \\ u_{i+1,j-1} &= -u_{i,j} + u_{i+1,j} + u_{i,j-1}\,, \qquad (= -u_i + u_{i-1} + u_{i+N_y})\,, \end{aligned}$$

which are accurate only to $\mathcal{O}(h)$. Because of this Stone [28] introduces the parameter α (in a sense, trying to improve the mean value theorem approximation) when inserting these into Eq. (2.111). Thus, this equation is replace by

$$\begin{aligned} A_i^{(1)}u_{i-N_y} + E_i^{(1)}\left[u_{i-N_y+1} - \alpha(-u_i + u_{i+1} + u_{i-n_y})\right] + A_i^{(2)}u_{i-1} + A_i^{(3)}u_i \\ + A_i^{(4)}u_{i+1} + E_i^{(2)}\left[u_{i+N_y-1} - \alpha(-u_i + u_{i-1} + u_{i+n_y})\right] + A_i^{(5)}u_{i+N_y} = f_i\,, \end{aligned}$$

or after rearrangement

$$\begin{aligned} \left(A_i^{(1)} - \alpha E_i^{(1)}\right)u_{i-N_y} + E_i^{(1)}u_{i-N_y+1} + \left(A_i^{(2)} - \alpha E_i^{(2)}\right)u_{i-1} + \left(A_i^{(3)} + \alpha E_i^{(1)} + \alpha E_i^{(2)}\right)u_i \\ + \left(A_i^{(4)} - \alpha E_i^{(1)}\right)u_{i+1} + E_i^{(2)}u_{i+N_y-1} + \left(A_i^{(5)} - \alpha E_i^{(2)}\right)u_{i+N_y} = f_i\,. \end{aligned} \tag{2.112}$$

From Eq. (2.112) we see that the original matrix A has been altered in all of its bands when $\alpha \neq 0$. Hence, construction of L and U must be carried out in a manner reflecting the fact that elements of the original error matrix are now present in the matrix undergoing incomplete decomposition. In particular, we can no longer calculate all elements of L and U before calculating those of E; elements of E must be calculated simultaneously with those of L and U. Nevertheless, the matrix band structure is still the same as before, and there is still an explicit, well-defined procedure by which this can be carried out.

This is done as follows. We first observe that $E_i^{(2)} = 0$ for $i = 1$, and $E_i^{(1)} = 0 \ \forall \ i \le N_y$. Thus, we can calculate d_1, e_1 and f_1 as before:

$$\begin{aligned} d_1 &= A_1^{(3)} \\ e_1 &= A_1^{(4)}/d_1 \\ f_1 &= A_1^{(5)}/d_1 \,. \end{aligned}$$

Next, we have $c_2 = A_2^{(2)} - \alpha E_2^{(2)} = A_2^{(2)} - \alpha c_2 f_1$, with E_2 being of the same form as when $\alpha = 0$. It follows that

$$c_2 = \frac{A_2^{(2)}}{1 + \alpha f_1} \,.$$

Hence, we can now calculate $E_2^{(2)} = c_2 f_1$, as in the previous case. This now permits evaluation of d_2:

$$d_2 = \left(A_2^{(3)} + \alpha E_2^{(2)}\right) - c_2 e_1 \,.$$

With d_2 thus obtained, we can calculate e_2 and f_2 in the usual way:

$$e_2 = A_2^{(4)}/d_2 \,, \qquad f_2 = \left(A_2^{(5)} - \alpha E_2^{(2)}\right) \,.$$

We are now ready to start calculation of the next row (in each of L and U) of matrix elements. We have $c_3 = A_3^{(2)} - \alpha E_3^{(2)} = A_3^{(2)} - \alpha c_3 f_2$. But f_2 is already known from preceding calculations; so we can solve the above for c_3:

$$c_3 = \frac{A_3^{(2)}}{1 + \alpha f_2} \,.$$

As was the case on the previous line of the matrices, we can now calculate $E_3^{(2)} = c_3 f_2$, since $E_3^{(1)} = 0$ we obtain d_3, and following this e_3 and f_3. We can proceed in exactly this way until $i = N_y + 1$. At this point we will first encounter contributions from the lowermost left band of the matrix, and associated with this nonzero values of $E_i^{(1)}$. The calculations must then proceed as follows. Since

$$b_i = A_i^{(1)} - \alpha E_i^{(1)} = A_i^{(1)} - \alpha b_i e_{i-N_y} \,,$$

it follows that

$$b_i = \frac{A_i^{(1)}}{1 + \alpha e_{i-N_y}} \,, \tag{2.113a}$$

$$E_i^{(1)} = b_i e_{i-N_y} \,, \tag{2.113b}$$

$$c_i = \frac{A_i^{(2)}}{1 + \alpha f_{i-1}} \,, \tag{2.113c}$$

$$E_i^{(2)} = c_i f_{i-1} \,, \tag{2.113d}$$

$$d_i = A_i^{(3)} + \alpha\left(E_i^{(1)} + E_i^{(2)}\right) - c_i e_{i-1} - b_i f_{i-N_y} \,, \tag{2.113e}$$

$$e_i = \left(A_i^{(4)} - \alpha E_i^{(1)}\right)/d_i \,, \tag{2.113f}$$

$$f_i = \left(A_i^{(5)} - \alpha E_i^{(2)}\right)/d_i \,. \tag{2.113g}$$

This is continued for all i to $i = N_x N_y$. It should be noted here that for $i \le N_y$ several of the factors in the above expressions have not yet been calculated. It is best to set these equal to zero until $i > N_y$.

It should be observed that the above procedure results in replacing the original system matrix A with a matrix M of the form $M = A + E$ where both M and E contain seven bands, compared with only five bands in A. On the other hand, M is readily decomposable, so (assuming $\rho(M^{-1}E) < 1$) the requirements set down earlier for an ILU decomposition procedure have basically been met.

The specific form of iteration procedure employed by Stone [28] is of the *residual form* which we have not previously discussed. It is a convenient and widely-used approach that is applicable rather generally, so we will develop it here. Starting with the original system of difference equations, $Au = b$, we replace A with $A = M - E = LU - E$ as before to obtain

$$LUu = Eu + b\,.$$

If we now add and subtract Au on the right-hand side of this equation, we find

$$\begin{aligned} LUu^{(n+1)} &= (A + E)u^{(n)} - (Au^{(n)} - b) \\ &= LUu^{(n)} + r^{(n)}\,, \end{aligned}$$

or

$$LU\delta^{(n+1)} = r^{(n)}\,. \tag{2.114}$$

The solution to this system is easily carried out in the manner just presented, after which we calculate

$$u^{(n+1)} = u^{(n)} + \delta^{(n+1)}\,.$$

We now consider the choice of iteration parameter α. We have already seen several approaches by which this might be done, and in particular we need to decide whether to find a single optimal parameter, corresponding to a stationary iteration procedure, or whether we should use a sequence of different parameters in a manner similar to what is done in ADI. It should be noted that Stone [28] indicated that using a cyclic implementation of SIP seemed to be far superior to employing a single optimal parameter. But no rigorous analysis is provided in [28], and the computations were performed on grids containing too few points to allow an assessment of the true asymptotic convergence rate. Nevertheless, we will follow the treatment of [28] in the current lectures, leaving as an open question whether better approaches might be found.

Stone [28] suggests that the values of α near unity tend to remove low-wavenumber error components while those near zero remove the high-wavenumber contributions. We comment that at least the latter seems to be confirmed by the behavior of the version of SIP used by Rubin and Khosla [29] in which α is set to zero. Thus, it appears that something might be gained by employing a parameter sequence. The following construction was introduced in [28] for equations of the form

$$-\left(a_1(x,y)u_x\right)_x - \left(a_2(x,y)u_y\right)_y = f(x,y)\,.$$

First, determine the maximum value of α (note that $\alpha \le 1$ must hold) from

$$\alpha_{max} = 1 - 2\min\left[\overline{\left(\frac{h_x^2}{1 + \frac{a_2(x,y)}{a_1(x,y)}\frac{h_x^2}{h_y^2}}\right)}, \overline{\left(\frac{h_y^2}{1 + \frac{a_1(x,y)}{a_2(x,y)}\frac{h_y^2}{h_x^2}}\right)}\right]\,. \tag{2.115}$$

The "bar" notation indicates averages taken over all grid point values, so there are only two arguments in the minimization function. There is no theoretical method for determining the number of iteration parameters to be included in the cycle, but once a number M is selected by whatever means, the parameters are chosen to satisfy

$$\alpha_m = 1 - (1 - \alpha_{max})^{m/(M-1)}\,, \qquad m = 0, 1, \dots, M-1\,. \tag{2.116}$$

Clearly, this procedure is ad hoc, and it appears from the descriptions in [28] that the parameter sequence is not necessarily used in order, with certain parameters repeated before the next one is employed.

As alluded to earlier, Stone [28] presents results that imply SIP produces solutions with number of iterations independent of the grid spacing h. But this was an artifact of computing on very coarse grids, and when SIP and related ILUs were applied to larger problems it was found that in general their asymptotic convergence rates were no better than that of optimal SOR, although they have sometimes been found to be more robust. Furthermore, it must be noted that a separate LU decomposition must be performed for each value of α, as is clear from Eqs. (2.113), and this entails a considerable amount of extra arithmetic over the single optimal α case—and this must be repeated cyclically if the corresponding Ls and Us are not stored.

2.5 Preconditioning

It is often noted (see, *e.g.*, Saad [12]) that *preconditioning* is somewhat difficult to define, and that it is probably best viewed as "any process that makes the system easier to solve." This is basically the viewpoint we will take here. We comment before starting that one of the reasons preconditioning is not well defined is that it is used in many different ways, for a considerable number of classes of problems, and implementations tend to vary with the class of problems. It is also worth pointing out that use of the term, preconditioning is relatively modern. Little use was made of the terminology prior to the early 1980s despite the fact that the basic underlying ideas have been known almost from the beginning of analysis of linear iterative procedures. Associated with this, at least peripherally, is the wide use of *condition number* in place of spectral radius in the modern numerical linear algebra literature. In particular, one way to look at a process of preconditioning is to require that it decrease the condition number of the matrix associated with the solution process (*e.g.*, the iteration matrix). It is clear that condition number and spectral radius are related since in the spectral norm

$$\kappa(A) = \|A\|\|A^{-1}\| = \frac{|\lambda|_{max}}{|\lambda|_{min}}\,,$$

where $\lambda \in \sigma(A)$, and κ is the condition number. Then, if A is diagonalizable we also have

$$\rho(A) = |\lambda|_{max}\,,$$

with $\rho(A)$ being the spectral radius of the matrix A. In specific cases additional properties of the matrices can be used to obtain particularly useful relationships between ρ and κ. Nevertheless, the fact remains that to calculate κ generally requires more effort than is needed to calculate ρ.

We begin by observing that any system whose matrix is the identity matrix can be solved trivially (as, of course, is also true for any diagonal matrix). This implies that a good preconditioner for the system given in Eq. (2.3),

$$Au = b\,,$$

would be any matrix M such that M^{-1} is "close to" A^{-1} in some appropriate sense, and hence the condition number of the resulting system is close to unity. With this in mind, we see that M chosen in this way can be used to produce the preconditioned system

$$M^{-1}Au = M^{-1}b\,, \tag{2.117}$$

and we would expect this to be relatively easy to solve since by construction $M^{-1}A \sim A^{-1}A = I$. The formulation given here is known as *left preconditioning.*

As one might expect, there is also a *right preconditioning*, constructed as follows. Obtain a matrix M as above, and set

$$M^{-1}v = u\,; \tag{2.118}$$

then substitution of this into the original system leads to

$$AM^{-1}v = b\,, \tag{2.119}$$

which is easily solved because $AM^{-1} \sim I$. Then the desired solution is obtained by solving

$$Mu = v\,.$$

As noted in [12] there seems to be no clear-cut advantage between left and right preconditioning in terms of overall effectiveness, but one or the other may prove to be preferred in the context of specific algorithms.

All of the methods we have previously discussed in these lectures can be viewed as preconditioners. We will demonstrate this here with two specific examples that will, in addition, highlight further requirements that one must consider when constructing preconditioners. We will first revisit Jacobi iteration for which the fixed-point formulation was first given in Eq. (2.8), and which we write here as

$$u^{(n+1)} = Bu^{(n)} + k\,, \tag{2.120}$$

with $B \equiv D^{-1}(L+U)$, and $k = D^{-1}b$. Recall that in this case L and U are, respectively, (negatives of) the lower and upper triangles of the original system matrix A (they do <u>not</u> arise from factorization), and

$$L + U = D - A\,.$$

Thus, we have

$$B = D^{-1}(D - A) = I - D^{-1}A\,,$$

a result we have obtained previously when discussing splitting matrices. Then (2.120) can be expressed as

$$u^{(n+1)} = (I - D^{-1}A)u^{(n)} + D^{-1}b\,, \tag{2.121}$$

which is nothing more than the Richardson iteration for the left-preconditioned system

$$D^{-1}Au = D^{-1}b\,,$$

as is easily checked. That is, the diagonal of the original system matrix A has been used as the preconditioner. Clearly, in general, D^{-1} is not very close to A^{-1} so we would expect that this would not provide a very effective method. On the other hand, construction and application of the preconditioner in this case is very inexpensive—a highly-desirable trait.

Our second example will also correspond to preconditioning of Richardson iteration, but here we will use a more elaborate matrix M. In particular recall that incomplete LU factorization applied to the original system $Au = b$ results in

$$Mu^{(n+1)} = Eu^{(n)} + b\,, \tag{2.122}$$

where M and E are related to the system matrix A via

$$M = A + E\,. \tag{2.123}$$

Furthermore, M admits a sparse LU decomposition consisting of lower and upper triangular matrices L and U such that

$$M = LU\,. \tag{2.124}$$

Now solving (2.123) for E and substituting the result into (2.122) yields

$$\begin{aligned} Mu^{(n+1)} &= (M - A)u^{(n)} + b \\ &= M(I - M^{-1}A)u^{(n)} + b\,, \end{aligned}$$

or

$$u^{(n+1)} = (I - M^{-1}A)u^{(n)} + M^{-1}b\,. \tag{2.125}$$

Again, this is precisely a Richardson iteration for the system

$$M^{-1}Au = M^{-1}b\,,$$

but now M^{-1} is obtained as the incomplete LU factorization $M^{-1} = U^{-1}L^{-1}$. Clearly, if M is close to A (implying E should be "small"), then M^{-1} would be expected to be close to A^{-1}, and $M^{-1}A \sim I$ should hold.

As might be expected, there are numerous other examples of preconditioners and their application. Indeed, once the splitting matrix Q in Eq. (2.5) has been identified, we can consider any of the linear fixed-point iterations treated here as a Richardson iteration of the original system preconditioned with Q^{-1}. We refer the reader to the cited references, especially [12], for further discussions.

2.6 Conjugate-Gradient Acceleration

In this section we will present an abbreviated treatment of the conjugate gradient (CG) method originally proposed by Hestenes and Stiefel [32]. It is of interest to note that when the method was first introduced it was presented as a direct solver for systems with symmetric, nonsingular $N\times N$ matrices A because in the $\underline{\text{absence}}$ $\underline{\text{of}}$ $\underline{\text{round-off}}$ $\underline{\text{errors}}$ it can be shown to produce an *exact* solution in at most N steps. This implies $\mathcal{O}(N^2)$ total arithmetic for this case, which is competitive with non-optimal SOR, Gauss–Seidel and Jacobi iterations—essentially the only viable alternatives at that time. Unfortunately, rounding errors seriously degrade the (already marginal) performance, and little immediate use was made of the method. In the mid to late 1960s the conjugate-gradient method was beginning to see application in nonlinear optimization where it provided a reasonably effective improvement over steepest-descent methods, as we will indicate below. By the 1980s, with the rise of procedures consisting of combinations of preconditioning and acceleration, it was found that the CG method could provide an effective accelerator, and in this context it is closely related to the Chebyschev *semi-iterative* procedures described, *e.g.*, in [18]. Furthermore, in recent years the connection between such polynomial accelerators and the Krylov-subspace methods has been recognized, providing yet another interpretation of the CG method.

In this section we will first follow the development presented in [18]. Thus, we begin with basic notions associated with steepest descent minimization, and then use this to motivate and derive the conjugate-gradient method. We then demonstrate the relationship between this procedure and polynomial acceleration, and finally indicate the connection to Krylov-subspace methods, basically following parts of Saad [12].

2.6.1 The method of steepest descent

Following Hageman and Young [18] we will assume the system matrix A is a $N\times N$ symmetric positive-definite matrix. We begin derivation of the *steepest-descent method* by constructing the quadratic form

$$F(u) = \frac{1}{2}\langle u, Au\rangle - \langle b, u\rangle\,, \tag{2.126}$$

corresponding to the system

$$Au = b\,.$$

Now since A is positive definite, we know $F(u)$ possesses a minimum. Indeed, it can be seen that $Au = b$ corresponds to a *critical point* of $F(u)$ independent of whether A is positive definite. Namely, minimizing (or, more generally, finding the critical point of) $F(u)$ is equivalent to solving $Au = b$ because the minimum (or critical point) occurs at $\nabla_u F(u) = 0$, and

$$\nabla_u F(u) = Au - b\,.$$

To minimize F starting from an approximation $u^{(n)}$ we calculate a new estimate by moving along a line in the negative gradient direction until $F(u^{(n+1)})$ is a minimum. This is done using the formula

$$u^{(n+1)} = u^{(n)} - \lambda_n \nabla F(u^{(n)}) = u^{(n)} + \lambda_n r_n \tag{2.127}$$

where, as defined earlier,

$$r_n = b - Au^{(n)}\,, \tag{2.128}$$

and λ_n is a *step length* given by

$$\lambda_n = \frac{\langle r_n, r_n \rangle}{\langle r_n, Ar_n \rangle}\,.$$

We can obtain this formula by substituting (2.127) into (2.126), using (2.128) to eliminate $u^{(n)}$, and then formally minimizing the result with respect to the scalar λ_n. The details of carrying this out are left to the interested reader.

Figure 2.14 provides a schematic of the geometry of steepest descent for a 2×2 case. It should be noticed that low eccentricity of the elliptic contours comprising the *level sets* in this figure is equivalent to the matrix A being well conditioned. In particular, in such a case the condition number $\kappa(A)$ should be close to unity, and this in turn implies that the smallest and largest eigenvalues must be relatively close together since these provide the scalings for the axes of the ellipse.

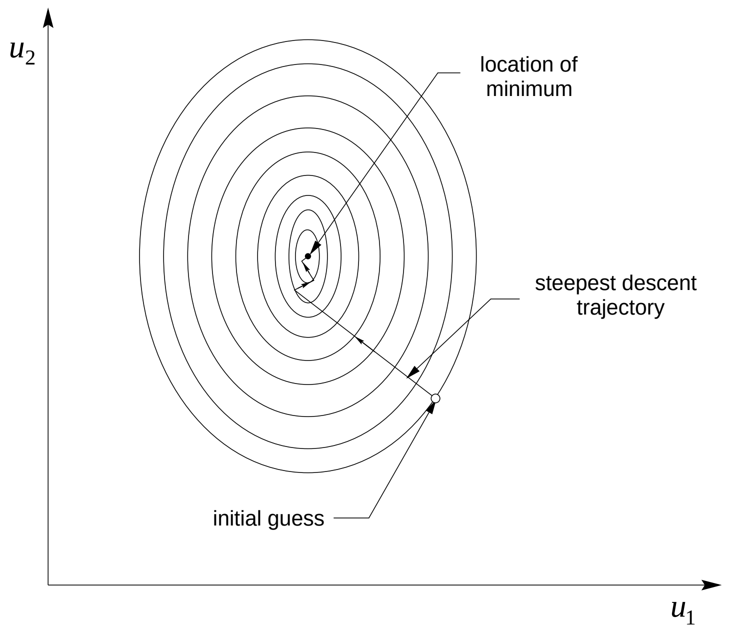

Figure 2.14: Level set contours and steepest-descent trajectory of 2-D quadratic form.

We observe that this method performs reasonably well in this case, but this is largely because the gradient direction of the first step took the trajectory very close to the neighborhood of the minimum point. Moreover, since the ellipses were not very eccentric, successive gradient directions proved to be reasonably good *search directions*. It is not too difficult to predict that this approach will not work well in the case of highly eccentric ellipses, and correspondingly large condition numbers for the matrix A. Figure 2.15 displays a case of this. In this figure we show both the steepest-descent convergence trajectory and the trajectory corresponding to the conjugate-gradient method. It is clear that the latter is far superior, and indeed because it utilizes information specific to the system matrix to select search directions, it does not suffer from the ill effects of highly eccentric elliptic search regions. In particular, it is well known that the steepest-descent method has a tendency to "ham stitch" when its trajectory enters a region of highly-elongated contours. This is specifically because it always follows the gradient direction, rather than the "natural" directions provided by the eigenvectors of the particular system matrix.

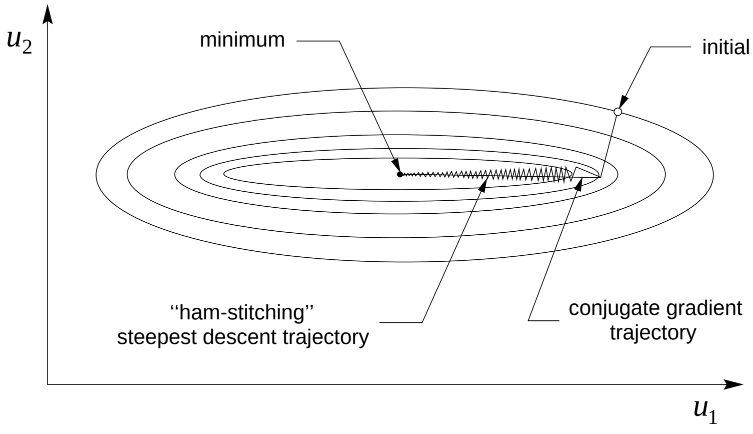
u_2
minimum
initial guess
"ham-stitching" steepest descent trajectory
conjugate gradient trajectory
u_1

Figure 2.15: Level set contours, steepest-descent trajectory and conjugate gradient trajectory of 2-D quadratic form.

2.6.2 Derivation of the conjugate-gradient method

The conjugate-gradient method can be derived in a number of different ways, as we have already implied. Here, we will use the same approach as applied for the steepest-descent method, but provide more details in the present case. Thus, as before, the basic idea is to minimize the functional

$$F(u) = \frac{1}{2}\langle u, Au\rangle - \langle b, u\rangle\,,$$

given in Eq. (2.126). We again assume the matrix A is symmetric and positive definite to guarantee that a minimum (in contrast to a general critical point) actually exists, and we propose to approach this minimum starting with an initial guess $u^{(0)}$ using the following formula:

$$u^{(n+1)} = u^{(n)} + \alpha_n p^{(n)}\,. \tag{2.129}$$

In this formula $p^{(n)}$ is a search direction calculated from step n information, and α_n is a similarly calculated step length. Clearly, the structure of these iterations is very similar to that of steepest descent, but we will derive the search direction so as to (at least partially) avoid the ham stitching exhibited by the steepest-descent algorithm when the condition number is large. In particular, we want the search direction to specifically incorporate information regarding the matrix A in a manner that is more effective than simply using the system residual vector as is done in steepest-descent methods. This is done by defining

$$p^{(n)} = r_n + \beta_n p^{(n-1)} \tag{2.130}$$

with the scalars β_n chosen so that the search directions $p^{(n)}$ are A-*conjugate*; that is, they are orthogonal with respect to the A-weighted inner product:

$$\langle p^{(n)}, Ap^{(n-1)}\rangle = 0\,. \tag{2.131}$$

Using (2.130) in this expression yields

$$\langle r_n, Ap^{(n-1)}\rangle + \beta_n\langle p^{(n-1)}, Ap^{(n-1)}\rangle = 0\,,$$

or

$$\beta_n = -\frac{\langle r_n, Ap^{(n-1)}\rangle}{\langle p^{(n-1)}, Ap^{(n-1)}\rangle}\,. \tag{2.132}$$

We observe that the form of the denominator in this expression requires positive definiteness of A to guarantee existence of β_n, just as was true for λ_n in the steepest-decent method.

There are at least two different (but equivalent) ways to determine the α_ns. One is to require that successive residuals always be orthogonal (in the usual inner product), and the other is to directly minimize $F(u)$ with respect to α_n. The first of these is trivial; we will apply the second because this will at the same time provide some of the missing details of deriving the steepest-descent formula. Thus, we substitute Eq. (2.129) into Eq. (2.126) to obtain

$$\begin{aligned} F(u^{(n+1)}) &= \frac{1}{2}\langle (u^{(n)} + \alpha_n p^{(n)}), A(u^{(n)} + \alpha_n p^{(n)})\rangle - \langle b, (u^{(n)} + \alpha_n p^{(n)})\rangle \\ &= \frac{1}{2}\left[\langle u^{(n)}, Au^{(n)}\rangle + \alpha_n\langle p^{(n)}, Au^{(n)}\rangle + \alpha_n\langle u^{(n)}, Ap^{(n)}\rangle + \alpha_n^2\langle p^{(n)}, Ap^{(n)}\rangle\right] \\ &\quad -\langle b, u^{(n)}\rangle - \alpha_n\langle b, p^{(n)}\rangle\,. \end{aligned}$$

To minimize this expression with respect to α_n we set

$$\frac{\partial F(u^{(n+1)})}{\partial \alpha_n} = 0\,,$$

which leads to

$$\frac{1}{2}\left[\langle p^{(n)}, Au^{(n)}\rangle + \langle u^{(n)}, Ap^{(n)}\rangle + 2\alpha_n\langle p^{(n)}, Ap^{(n)}\rangle\right] = \langle b, p^{(n)}\rangle\,.$$

But since A is symmetric this can be simplified to

$$\langle p^{(n)}, Au^{(n)}\rangle + \alpha_n\langle p^{(n)}, Ap^{(n)}\rangle = \langle b, p^{(n)}\rangle\,.$$

It is worthwhile to mention here that symmetry, *per se*, is not essential to the construction of α_n. On the other hand, if A is not symmetric, and thus not necessarily diagonalizable, the favorable effects of A conjugacy may be seriously degraded with poorer performance resulting.

Now recall that $Au^{(n)} = b - r_n$ to express the above as

$$\alpha_n = \frac{\langle p^{(n)}, r_n\rangle}{\langle p^{(n)}, Ap^{(n)}\rangle}\,. \tag{2.133}$$

The preceding formulas can be cast in a slightly different form, as shown by Saad [12]. In particular, from (2.129) and the form of the basic linear system being solved, it can be shown that the residual can be calculated from the recursion

$$r_{n+1} = r_n - \alpha_n Ap^{(n)}$$

as an alternative to directly calculating it from the definition. Then, with this formula we can use required orthogonality of the residuals to write

$$\langle r_n - \alpha_n Ap^{(n)}, r_n\rangle = 0\,,$$

from which follows an alternative formula for α_n:

$$\alpha_n = \frac{\langle r_n, r_n\rangle}{\langle Ap^{(n)}, r_n\rangle}\,. \tag{2.134}$$

Via analogous manipulations one can also obtain a simplified formula for β_n:

$$\beta_{n+1} = -\frac{\langle r_{n+1}, r_{n+1}\rangle}{\langle r_n, r_n\rangle}\,. \tag{2.135}$$

It is indicated in Hageman and Young [18] that these formulations are equivalent and are obtained from the following identities proven in Hestenes and Stiefel [32]:

$$\begin{aligned}\langle r_n, r_m\rangle &= 0 \qquad \forall\, m \neq n\\ \langle p^{(n)}, Ap^{(m)}\rangle &= 0 \qquad \forall\, m \neq n\\ \langle r_n, Ap^{(m)}\rangle &= 0 \qquad \forall\, m \neq n \quad \text{and} \quad m \neq n+1\,.\end{aligned}$$

We can now provide a pseudo-language algorithm corresponding to the preceding equations, in this case following the presentation in [18].

Algorithm 2.3 (CG method) *Compute $r_0 = b - Au^{(0)}$, and set $p^{(0)} = r_0$. Then carry out the following steps:*

```
Do $n = 0, maxit$
  1. Calculate step length $\alpha_n$
       $\alpha_n = \langle p^{(n)}, r_n\rangle / \langle p^{(n)}, Ap^{(n)}\rangle$
  2. Update solution values
       $u^{(n+1)} = u^{(n)} + \alpha_n p^{(n)}$
  3. Calculate residual based on updated solution
       $r_{n+1} = b - Au^{(n+1)}$
     If residual is sufficiently small, stop
  4. Calculate conjugate direction for next iteration
       $\beta_{n+1} = -\langle r_{n+1}, Ap^{(n)}\rangle / \langle p^{(n)}, Ap^{(n)}\rangle$ ,
       $p^{(n+1)} = r_{n+1} + \beta_{n+1} p^{(n)}$
Repeat $n$
```

It should be clear that this is not a stationary iterative procedure; indeed, it cannot be formulated as a linear fixed-point iteration. Thus, none of our standard analysis techniques can be applied directly.

2.6.3 Relationship of CG to other methods

We hinted in the opening paragraph of this section that there are relationships between the conjugate-gradient method and other widely-studied approaches for solving sparse linear systems. In this section we will briefly consider these. The first is Chebyshev acceleration, which is a so-called *polynomial* acceleration method that has been thoroughly studied. From its relation to the CG method we will be able to deduce the convergence rate of CG iterations. The second related class of methods is that associated with use of Krylov subspaces. These are also formally equivalent to polynomial accelerations, and thus also to the CG method. One of the currently most-often used of such techniques is the generalized minimum residual (GMRES) method.

Chebyshev acceleration is often proposed as a method for accelerating other iterative procedures because it is able to make use of more information than that simply from the most recent iterate. Indeed, the Chebyshev polynomials satisfy a three term recursion relation leading to a natural acceleration technique involving two iteration levels to produce a third. It turns out that the CG method can be recast into this form (see *e.g.*, [12], [18] or [19]). We will not provide the details here since we do not plan to develop this further in these lectures, but an important aspect of this is that if bounds for the eigenvalues of the corresponding iteration matrix are known, the convergence rate can be predicted. This leads to the following result for the CG method:

$$\|e_n\|_A \le 2\left[\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right]^n \|e_0\|_A\,, \tag{2.136}$$

where e_n is the exact error defined earlier in (2.14), $\|\cdot\|_A$ is the norm induced by the A-weighted inner product, and κ is the condition number of the system matrix. It should be observed that the form of Eq. (2.136) is similar to that of any other linearly-convergent procedure; on the other hand, it depends only on the square root of the condition number, rather than on the condition number, itself, as is often the case.

To demonstrate the connection of conjugate-gradient methods to those based on use of Krylov subspaces, we first remind the reader how a Krylov subspace is defined. As usual we consider the system of equations $Au = b$ to be solved iteratively starting with an initial guess $u^{(0)}$ and corresponding residual r_0. We then construct the sequence of vectors $r_0, Ar_0, A^2r_0, \ldots, A^{m-1}r_0$ and define the Krylov subspace $u_0 + \mathcal{K}_m$ by

$$\mathcal{K}_m(A, r_0) = \mathrm{span}\{r_0, Ar_0, A^2r_0, \ldots, A^{m-1}r_0\}\,. \tag{2.137}$$

It is assumed that this set of vectors can be used to approximate the basis needed to construct the solution u to the system whose matrix is A. Thus,

$$u \simeq c_0r_0 + c_1Ar_0 + c_2A^2r_0 + \cdots + c_{m-1}A^{m-1}r_0\,.$$

This represents a polynomial acceleration method of the sort we have already discussed in the context of Chebyshev acceleration, and it shows that all three of conjugate-gradient, Chebyshev and Krylov subspace methods are related.

2.7 Introduction to Multigrid Procedures

In this section we will introduce one of the most important modern approaches to solving sparse linear systems arising from discretization of elliptic PDEs, *viz.*, multigrid methods. We begin with a somewhat general overview of the procedure, follow this with details of the so-called "two-grid method," and end with additional developments needed for construction of "full multigrid" methods.

2.7.1 Some basic ideas

The basic notions underlying construction of multigrid (MG) methods have been known for a very long time; they are the following:

i) use of relaxation-type methods as smoothing operators on fine grids;

ii) recursive application of coarse-grid (defect) corrections, and

iii) nested iteration.

Items *i*) and *iii*) have been used together for a wide range of problems over a long period of time, and it turns out that *iii*) is the last step in what is now known as the *full multigrid* (FMG) *procedure.* The first application of *i*) and *ii*) together is apparently due to Fedorenko [33] in 1961. In 1966 Bachvalov [34] suggested combining this with *iii*), but he did not actually implement the scheme. It is the proper combination of all three elements that leads to the extremely good theoretical convergence rates for multigrid methods, as demonstrated by Brandt [35], and by Bramble *et al.* [36].

Of all sparse matrix techniques, only the various multigrid and domain decomposition (to be discussed in the next section) procedures exhibit convergence rates that approach being independent of grid spacing h (*i.e.*, independent of the number of equations in the system). Moreover, multigrid methods can be constructed so that the total arithmetic required to solve a system of N equations is only $\mathcal{O}(N)$. Hence, as a class, these methods are optimal.

It is important to observe that convergence rate independent of h is not, alone, sufficient to guarantee efficiency. In particular, direct elimination methods require only a single "iteration," independent of h, to obtain the exact (to machine precision) solution. But we are well aware that such methods cannot be readily applied to large sparse systems because each such "iteration" requires $\mathcal{O}(N^3)$ arithmetic operations. Thus, to obtain an optimally efficient method we must seek approaches requiring a number of iterations that is independent of the number of equations (recall that Newton's method provides an example of this for nonlinear equations), and this number must be relatively small—$\sim \mathcal{O}(1)$. Then if only $\mathcal{O}(N)$ arithmetic operations are needed at each such iteration, the total arithmetic will scale as $\mathcal{O}(N)$.

In this introduction we will closely follow the treatment of Stüben and Trottenberg [37] (note that this same approach can also be found in the more recent monograph by Briggs [38]), presenting first a fairly detailed discussion of the two-grid case, and then a few aspects of the ℓ-grid methods, followed by a brief discussion of the full multigrid procedure.

We begin by developing the notions corresponding to *i*) and *ii*) above. It is interesting to note that this is analogous to use of *deferred corrections*, or in the context of linear systems treated here, *iterative improvement* (see Böhmer and Stetter [39] for further discussions). In particular, suppose we approximately solve the system

$$A_h u_h = b_h\,, \tag{2.138}$$

and obtain the iterate $u_h^{(n)}$. We comment that while the algorithms of previous sections in some cases did not specifically make use of the fact that the sparse matrix systems to which they were applied arose by discretizing a PDE, our presentation of MG methods does reflect this. In particular, the h subscripts appearing in the above equation indicate that this system is a discrete approximation formulated with a grid of spacing h. This will be particularly important in the development of restriction and prolongation operators discussed below.

Now observe that if we could calculate the exact error $e_h^{(n)}$ by some means, we could add this to $u_h^{(n)}$ to obtain the exact solution (to within machine precision); *i.e.*, we have

$$u_h = u_h^{(n)} + e_h^{(n)}\,.$$

To obtain an equation for $e_h^{(n)}$ we rearrange this as

$$e_h^{(n)} = u_h - u_h^{(n)}\,,$$

and multiplication by A_h yields

$$A_h e_h^{(n)} = A_h u_h - A_h u_h^{(n)} = b_h - A_h u_h^{(n)} = r_h^{(n)}\,.$$

Hence,

$$A_h e_h^{(n)} = r_h^{(n)}\,, \tag{2.139}$$

as given earlier in Eq. (2.16). This is usually called the *defect equation*; and, as we earlier observed, accurately solving this requires the same amount of work as is needed to calculate u_h itself.

But an important observation is that if $e_h^{(n)}$ is smooth, it is not necessary to determine it exactly in order to obtain a significant improvement in the accuracy of $u_h^{(n)}$. Namely, we could solve the defect equation on a coarser grid, say of spacing $2h$, and still obtain a useful improvement in the solution $u_h^{(n)}$. In particular, we might solve

$$A_{2h} e_{2h}^{(n)} = r_{2h}^{(n)}\,. \tag{2.140}$$

and then form

$$u_h^{(n+1)} = u_h^{(n)} + e_{2h}^{(n)}\,. \tag{2.141}$$

Equation (2.140) can be solved much more easily than can (2.139) simply because the solution vector will contain approximately a factor of four fewer components (for a 2-D problem). On the other hand, if $2h$ is the coarsest grid employed (as will be the case in the next section), then Eq. (2.140) must be solved "exactly" in the sense that iteration errors should be smaller than the minimum of the required iteration tolerance and truncation errors on the finest (*i.e.*, h) grid in order to avoid introduction of additional error.

It is clear from (2.140) and (2.141) that there are incompatibilities in vector lengths. For example, since $e_{2h}^{(n)}$ is roughly four times shorter than $r_h^{(n)}$ the latter must be appropriately shortened by applying what is termed a *restriction operator*. If we let $\mathcal{S}_h$ denote the set of points in the grid of spacing h and similarly let $\mathcal{S}_{2h}$ denote those of the $2h$ grid, in general it will be true that $\mathcal{S}_h \cap \mathcal{S}_{2h} \neq \emptyset$ on a structured grid (see Chap. 5). Thus, we could (and sometimes do) define the restriction operator so as to merely pick out the points lying in this intersection; this is termed *injection*. However, for theoretical reasons we will discuss in more detail later, this is not always done. We will consider a more widely-used restriction operator below.

Similarly, it is also easy to see that $e_{2h}^{(n)}$ is too short for use in Eq. (2.141) by a factor of about four. This is remedied by applying a *prolongation operator* to $e_{2h}^{(n)}$ to obtain $e_h^{(n)}$; this merely amounts to interpolation between the $\mathcal{S}_{2h}$ and $\mathcal{S}_h$ point sets. As with restriction, there are many alternatives for constructing prolongation operators, and we shall treat this in more detail later.

By this time we have accumulated enough information to make a first attempt at constructing a basic multigrid algorithm. This will be the topic of the next section.

2.7.2 The h-$2h$ two-grid algorithm

In this subsection we will first present a formal algorithm corresponding to the two-grid h-$2h$ multigrid method. We will follow this with a fairly detailed discussion of each of the main steps of the procedure, and we will then derive the general fixed-point form of this method.

Algorithm 2.4 (Two-grid MG) *Suppose n iterations have been completed. Compute the $n+1^{th}$ iterate by carrying out the following steps.*

1. *Obtain an approximate solution $u_h^{(n+1)^*}$ to $A_h u_h = b_h$ using a relaxation-like method to accomplish fine-grid smoothing starting from the (current) "initial guess" $u_h^{(n)}$, and compute the residual,*

$$r_h^{(n+1)} = b_h - A_h u^{(n+1)^*}$$

2. *Test convergence: if $\|r_h^{(n+1)}\| < \epsilon$, then stop; else continue to step 3.*

3. *Restrict residual to coarse grid (fine-to-coarse grid transfer) using a restriction operator $\mathcal{R}_h^{2h} : r_h \rightarrow r_{2h}$*

$$r_{2h}^{(n+1)} = \mathcal{R}_h^{2h} r_h^{(n+1)}$$

4. *"Exactly" solve the defect equation*

$$A_{2h} e_{2h}^{(n+1)} = r_{2h}^{(n+1)}$$

5. *Prolongate (interpolate) correction $e_{2h}^{(n+1)}$ (coarse-to-fine grid transfer) using a prolongation operator $\mathcal{P}_{2h}^h : e_{2h} \rightarrow e_h$*

$$e_h^{(n+1)} = \mathcal{P}_{2h}^h e_{2h}^{(n+1)}$$

6. *Compute new approximation to u_h,*

$$u_h^{(n+1)} = u_h^{(n+1)^*} + e_h^{(n+1)}$$

and go to 1.

It is important to consider each step of this algorithm in some detail; we do this in the following subsections.

Fine-grid smoothing

The first step requires computation of an approximate solution on the finest grid. But the underlying philosophy in carrying this out is now considerably different than would be the case in a single-grid algorithm. In particular, it is not necessary to obtain an extremely accurate approximation as would normally be the case. We can argue this heuristically by noting that the residual carried to the coarser grid, and hence the defect correction, depends (linearly) on the error in the approximate solution. Thus, we would expect (correctly) that the defect correction should, to some extent, automatically compensate for errors in the fine-grid approximation of u_h (the larger the error, the larger the defect correction). But we must keep in mind one of our basic assumptions that permits use of a defect correction computed on a coarser grid: $e_{2h}^{(n+1)}$ is presumed to be smooth. To guarantee this, it must be true that $r_h^{(n+1)}$ (or, at least $r_{2h}^{(n+1)}$) is smooth, and this implies that enough iterations must be performed on the fine grid to eliminate the high-wavenumber contributions to the error. These correspond to lack of smoothness in $u_h^{(n+1)^*}$ and, hence, also in $r_h^{(n+1)^*}$. At the same time, we should not perform significantly more iterations than are required to accomplish this. In particular, sufficiently few iterations should be used to avoid reaching the asymptotic convergence rate of the relaxation method being employed (recall Fig. 2.4).

These ideas imply that the iteration method employed on the fine grid need not be rapidly convergent in the sense that its spectral radius is very much less than unity, but rather that the method should possess strong smoothing properties with respect to high wavenumbers. It is interesting to note that this is precisely the behavior of the typical relaxation schemes such as Jacobi and Gauss–Seidel (but not optimal SOR) iterations. All of these efficiently remove high-wavenumber errors but are very inefficient at reducing the low-wavenumber errors. (Recall that the spectral radius of the Jacobi iteration matrix is set by the lowest possible wavenumbers.)

In connection with these notions we can observe that the underlying reason for the effectiveness of MG procedures is that they do not attempt to eliminate low-wavenumber error during the (expensive) fine-grid calculations. Rather, they carry out low-wavenumber error reduction on coarse grids where the low wavenumbers become "high" wavenumbers with respect to that particular grid, and are thus more easily removed.

Restriction: fine-to-coarse grid transfer

The next main step (assuming convergence has not occurred) in this basic algorithm is restriction of the now (presumably) smooth residual from the fine-grid calculation to the coarse grid. To discuss methods for doing this, it is useful to consider some details of the h and $2h$ grids as shown in Fig. 2.16. In this figure it is clear that $\mathcal{S}_{2h} \subset \mathcal{S}_h$, so the restriction operator $\mathcal{R}_h^{2h}$ may be chosen in an especially simple manner, *i.e.*, *injection.* This simply entails defining $r_{2h}^{(n+1)}$ so as to equal $r_h^{(n+1)}$ at those grid points contained in $\mathcal{S}_{2h} \cap \mathcal{S}_h$.

On the other hand, restriction is more often carried out in the following way. Suppose we first use the above injection to identify the (i, j) fine-grid point with the (k, l) coarse-grid point. Then, instead of setting $r_{k,l}^{(n+1)} = r_{i,j}^{(n+1)}$, as would be done for injection, we use

$$r_{k,l} = \frac{1}{16}\left[r_{i-1,j-1} + 2r_{i-1,j} + r_{i-1,j+1} + 2r_{i,j-1} + 4r_{i,j} + 2r_{i,j+1} + r_{i+1,j-1} + 2r_{i+1,j} + r_{i+1,j+1}\right] , \qquad (2.142)$$

where we have suppressed iteration counter notation.

It is not hard to check that this constitutes a simple filtering procedure, so in addition to restriction, *per se*, it provides further smoothing. This is an important consideration because it implies that fewer smoothing iterations may be sufficient on the fine grid if this restriction operator is used. On the other hand, however, it is also important to recognize that additional $\mathcal{O}(h^2)$ errors are introduced by this filter, and these will be carried, ultimately, to $e_h^{(n+1)}$, from which they must be removed during fine-grid smoothing.

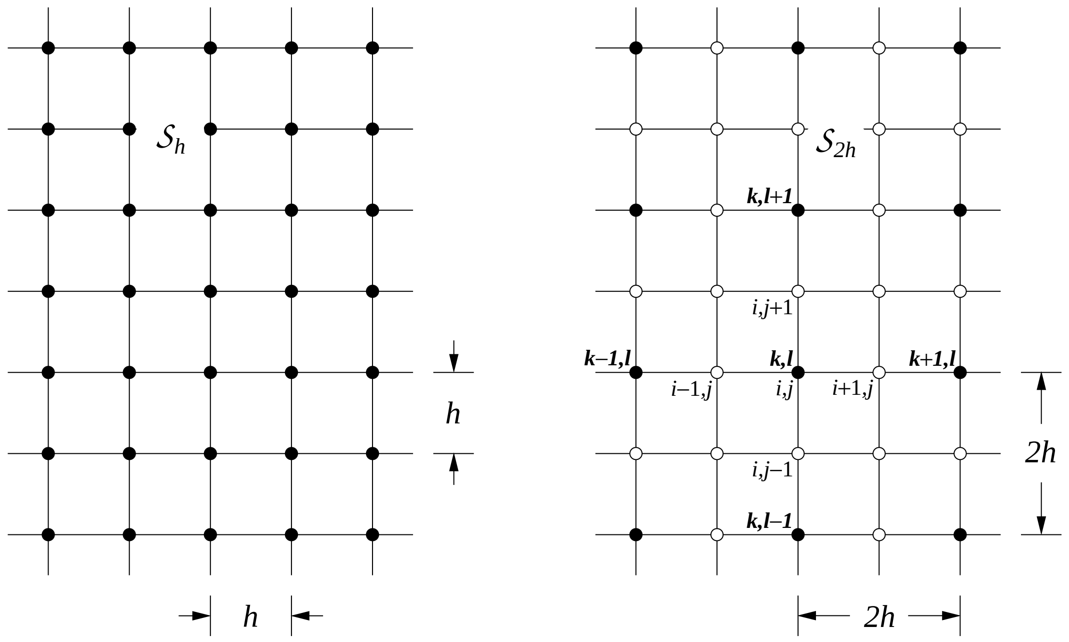

Figure 2.16: Comparison of h and $2h$ grids for multigrid implementations.

Solution of the defect equation

We are now prepared to solve the defect equation on the coarse grid. Typically, one might use the same solution procedure as employed for the fine-grid smoothing, but there is no *a priori* reason for doing this, and in fact, there are some reasons for not doing so. In particular, the algorithm employed for fine-grid smoothing need not be rapidly convergent—as we have already emphasized, whereas it is important that the procedure used for solving the defect equation be quite efficient since an exact solution to this equation is required. For example, Jacobi (or even damped-Jacobi) iterations are often recommended for fine-grid smoothing; but optimal SLOR, or possibly a conjugate-gradient method, would in most cases be more effective for solving the defect equation in a two-grid algorithm.

Prolongation: coarse-to-fine grid transfer

Just as there are many possibilities for restriction operators, there are also many ways in which prolongation (interpolation) may be done (and as we will note later, these are closely related to restriction). Clearly, the details of prolongation will depend on the specific type of h-$2h$ gridding used. In particular, Fig. 2.16 is not the only possibility; the interested reader may wish to consult Hackbusch [40] for more details on this and other aspects of multigrid methods in general. For the grid configuration depicted above, linear interpolation described in the following algorithm is often used.

Algorithm 2.5 (Prolongation: coarse-to-fine grid transfer of defect correction) *After "exact" solution of defect equation, perform the following steps to interpolate results to fine grid.*

1. *Interpolate along horizontal lines to obtain*

$$e_{i+1,j}^{(n+1)} = \frac{1}{2}\left(e_{k,l}^{(n+1)} + e_{k+1,l}^{(n+1)}\right)$$

2. *Interpolate along vertical lines to obtain*

$$e_{i,j+1}^{(n+1)} = \frac{1}{2}\left(e_{k,l}^{(n+1)} + e_{k,l+1}^{(n+1)}\right)$$

3. *Obtain results for interior points not appearing on the $2h$ grid lines by averaging the four nearest corner neighbors:*
$$e_{i-1,j-1}^{(n+1)} = \frac{1}{4}\left(e_{k-1,l-1}^{(n+1)} + e_{k-1,l}^{(n+1)} + e_{k,l-1}^{(n+1)} + e_{k,l}^{(n+1)}\right)$$

It is important to recognize that these formulas hold <u>only</u> for uniform grid spacings such as depicted in Fig. 2.16. That is, in this case simple arithmetic averaging is equivalent to linear interpolation. For general non-uniform grids, the factors 1/2 and 1/4 must be replaced with appropriate interpolation coefficients.

We can now complete the current multigrid iteration by calculating
$$u_h^{(n+1)} = u_h^{(n+1)^*} + e_h^{(n+1)}\,. \tag{2.143}$$
But it is important to note at this point that new high-wavenumber errors will have been introduced by the interpolation process, and these must be removed. This is done with a final smoothing on the fine grid, to be performed before convergence is tested. Recall that in the h-$2h$ algorithm, we tested convergence after an initial fine-grid smoothing operation, but in fact in this case the final smoothing discussed here and the initial one of the algorithm are actually the same if more than one overall iteration is to be performed.

Fixed-point form of 2-grid multigrid procedures

It is interesting to note at this point that the above algorithm can be cast in the standard form of a linear fixed-point iteration. This can be done via the following steps. First, recall that
$$r_h^{(n+1)} = b_h - A_h u_h^{(n+1)^*}\,,$$
and thus
$$r_{2h}^{(n+1)} = \mathcal{R}_h^{2h}\left(b_h - A_h u_h^{(n+1)^*}\right)\,.$$
Now the formal solution to the defect equation is
$$\begin{aligned} e_{2h}^{(n+1)} &= A_{2h}^{-1} r_{2h}^{(n+1)} \\ &= A_{2h}^{-1}\mathcal{R}_h^{2h}\left(b - A_h u_h^{(n+1)^*}\right)\,, \end{aligned}$$
so it follows that
$$e_h^{(n+1)} = \mathcal{P}_{2h}^h A_{2h}^{-1}\mathcal{R}_h^{2h}\left(b_h - A_h u_h^{(n+1)^*}\right)\,.$$
Thus, the updated solution is
$$\begin{aligned} u_h^{(n+1)} &= u_h^{(n+1)^*} + e_h^{(n+1)} \\ &= u_h^{(n+1)^*} + \mathcal{P}_{2h}^h A_{2h}^{-1}\mathcal{R}_h^{2h}\left(b_h - A_h u_h^{(n+1)^*}\right) \\ &= \left(I - \mathcal{P}_{2h}^h A_{2h}^{-1}\mathcal{R}_h^{2h} A_h\right) u_h^{(n+1)^*} + \mathcal{P}_{2h}^h A_{2h}^{-1}\mathcal{R}_h^{2h} b_h\,. \end{aligned}$$

At this point recall that $u_h^{(n+1)^*}$ results from some number ν of smoothing operations which we will assume correspond to a linear fixed-point iteration written as
$$u_h^{(m+1)} = S u_h^{(m)} + k\,,$$
where S is a smoothing operator, *e.g.*, Gauss–Seidel or Jacobi iteration, and $m = 0, 1, \ldots, \nu$, with $m = 0$ corresponding to $u_h^{(n)}$. It is easily shown that
$$\begin{aligned} u_h^{(n+1)^*} &= S^\nu u_h^{(n)} + \left(I + S + \cdots + S^{\nu-1}\right) k \\ &= S^\nu u_h^{(n)} + \left(I + \sum_{m=1}^{\nu-1} S^m\right) k\,. \end{aligned}$$

Next substitute these results into the above expression for $u_h^{(n+1)}$ to obtain

$$u_h^{(n+1)} = \left(I - \mathcal{P}_{2h}^h A_{2h}^{-1} \mathcal{R}_h^{2h} A_h\right) \left[S^\nu u_h^{(n)} + \left(I + \sum_{m=1}^{\nu-1} S^m\right) k\right] + \mathcal{P}_{2h}^h A_{2h}^{-1} \mathcal{R}_h^{2h} b_h \,.$$

Now associated with S there is a splitting matrix, say Q, such that

$$k = Q^{-1} b_h \,,$$

so the above becomes

$$\begin{aligned} u_h^{(n+1)} &= \left(I - \mathcal{P}_{2h}^h A_{2h}^{-1} \mathcal{R}_h^{2h} A_h\right) \left[S^\nu u_h^{(n)} + \left(I + \sum_{m=1}^{\nu-1} S^m\right) Q^{-1} b_h\right] + \mathcal{P}_{2h}^h A_{2h}^{-1} \mathcal{R}_h^{2h} b_h \\ &= \left(I - \mathcal{P}_{2h}^h A_{2h}^{-1} \mathcal{R}_h^{2h} A_h\right) S^\nu u_h^{(n)} \\ &\quad + \left[\left(I - \mathcal{P}_{2h}^h A_{2h}^{-1} \mathcal{R}_h^{2h} A_h\right) \left(I + \sum_{m=1}^{\nu-1} S^m\right) Q^{-1} + \mathcal{P}_{2h}^h A_{2h}^{-1} \mathcal{R}_h^{2h}\right] b_h \,. \end{aligned} \tag{2.144}$$

Clearly, this is in the usual linear fixed-point form with

$$\begin{aligned} G &\equiv \left(I - \mathcal{P}_{2h}^h A_{2h}^{-1} \mathcal{R}_h^{2h} A_h\right) S^\nu \,, \\ k &\equiv \left[\left(I - \mathcal{P}_{2h}^h A_{2h}^{-1} \mathcal{R}_h^{2h} A_h\right) \left(I + \sum_{m=1}^{\nu-1} S^m\right) Q^{-1} + \mathcal{P}_{2h}^h A_{2h}^{-1} \mathcal{R}_h^{2h}\right] b_h \,. \end{aligned}$$

Finally, we remark that analogous results can be obtained for the ℓ-grid multigrid methods and the full multigrid procedure of subsequent sections. Moreover, we will later see that very similar constructions can be carried out in the context of domain-decomposition methods discussed later, providing a link between these two very important classes of modern elliptic equation solvers.

2.7.3 ℓ-grid multigrid methods

We next consider the more general ℓ-grid procedures. As should be clear from the discussions we provide here, these are usually necessary if the fine grid contains a relatively large number of points, say $\sim \mathcal{O}(10^4)$ or more. To make our descriptions and figures simpler, we introduce the following symbolism. We let $\bullet$ denote smoothing, $\square$ represent an exact solution, and $\longrightarrow$ indicates direction of transfer of information. Use of these symbols is demonstrated in Fig. 2.17.

Part (a) of this figure depicts use of the symbols in representing the already familiar 2-grid form of multigrid, while part (b) shows a 3-grid method for which $\ell = 3$. This particular version of multigrid is known as a *V-cycle* procedure, for obvious reasons. Calculations begin at the upper left with fine-grid smoothing; results are transferred to the coarse grid indicated by the square, where an exact solution (in the sense described earlier) is computed, and these results are interpolated back to the fine grid by proceeding up the right-hand side of the diagram, where additional smoothing iterations are performed as needed. In the case of $\ell = 3$, an intermediate grid is employed. Thus, the first restriction is to this intermediate grid where mid-range wavenumber errors are removed. These results are then restricted to the coarse grid for removal of low-wavenumber errors. This, again, requires an exact solution, and once this is completed prolongation back to the fine grid is begun; the first step is interpolation to the intermediate grid were a few (usually not more than three or four) smoothing iterations are performed. Then prolongation to the finest grid is performed, followed by additional smoothing.

It is not hard to imagine the construction of much more elaborate multigrid cycles than those depicted in Fig. 2.17, and there is an additional parameter to help identify these. In particular, the two V-cycles

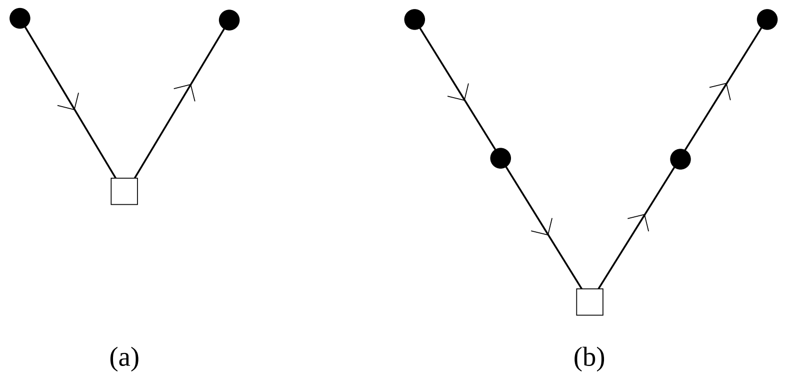

Figure 2.17: Multigrid V-cycles; (a) $\ell = 2$, and (b) $\ell = 3$.

shown above each contain a single V, and they correspond to a value of the parameter $\gamma = 1$. Figure 2.18 provides an indication of how cycle structure changes with γ for fixed ℓ $(= 3)$ in this case. What can be seen from this figure is that γ indicates the number of V-cycles present in the overall multigrid cycle. Furthermore, we can see from Fig. 2.18 (b) that when $\gamma = 2$, the cycle appears as a W, and as might be expected, this is termed a *W-cycle*. One can also assign values of γ to W-cycles in the same way as we have done for V-cycles: $\gamma = 2$ would imply two W-cycles. Much numerical experimentation has been done with various cycles, but there seems to be little in the way of clear-cut evidence that any particular one should be preferred. We shall not pursue this further herein, but additional details are available, for example, in [40].

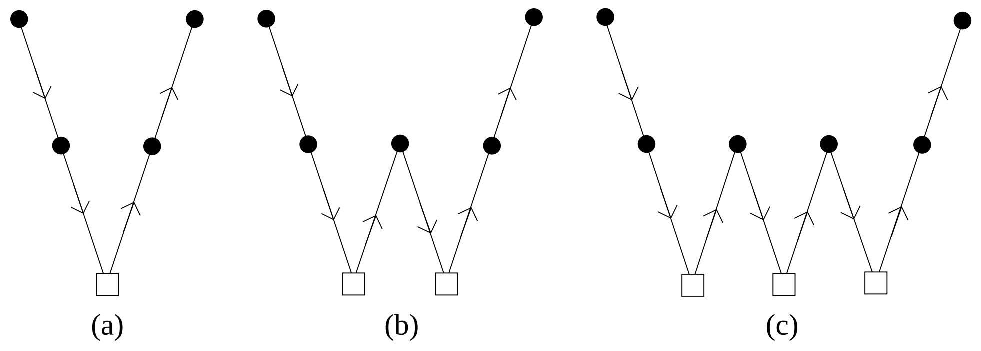

Figure 2.18: Multigrid V-cycles with $\ell = 3$ and different values of γ; (a) $\gamma = 1$, (b) $\gamma = 2$ and (c) $\gamma = 3$.

2.7.4 The full multigrid method

The final topic we will present with regard to multigrid methods is the so-called *full multigrid method* (FMG) in which all of the ideas discussed at the outset are incorporated in a single algorithm. The reader should recall at this point that of the three ideas listed as making up a multigrid procedure, only nested iteration has been left untreated. In the present section we will describe the use of this in conjunction with fine-grid smoothing and coarse-grid corrections for constructing the FMG procedure.

We begin by recalling that *nested iteration* is simply the process in which easily computed results from a coarse grid are interpolated to a finer grid and used as an initial guess for iterations on that grid. At some point (probably before a completely converged solution is obtained) iterations are terminated, and results are interpolated to the next finer grid until the finest grid is reached. There are several important aspects of this procedure to note. First, it is a very natural one, and it has been used alone on an intuitive basis for many years. Second, it is not very different from the right-hand side of a V-cycle discussed above.

But in this regard, there is at least one important distinction. In the V-cycle it is errors (defects) that are being interpolated from one grid to the next finer one, but in the case of nested iteration it is the solution itself. On the other hand, since the errors are to be used to improve the solution on the finest grid, it is not clear that the overall treatment or strategy should differ much between the two cases. The final item of note (and this is also true of prolongation of the coarse-grid corrrections) is that convergence results have been proven. In particular, in the present case it has been shown that a properly implemented nested iteration will result in an initial guess on the finest grid that is no more in error from the true solution than a multiple of the truncation error of the discretization (see [40]).

There are many different alternative FMG implementations, and we will consider a very specific one here. We remark, however, that a common trait among all of these is that they are claimed to exhibit $\mathcal{O}(N)$ scaling for total arithmetic, or equivalently, a number of (fine-grid equivalent) iterations independent of the grid spacing. In Fig. 2.19 we present a schematic of the procedure we will investigate. We begin

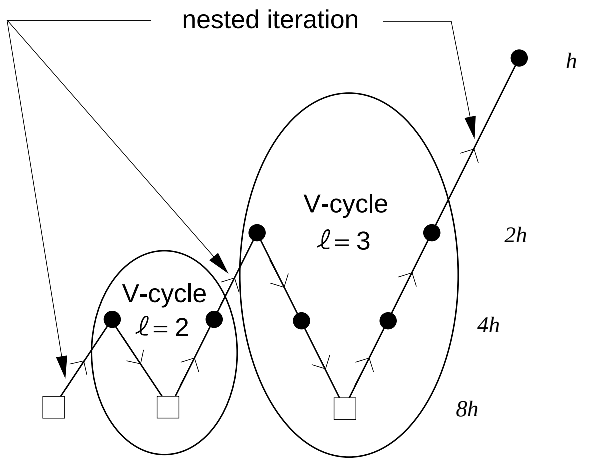

Figure 2.19: Four-Level, V-cycle full multigrid schematic.

by observing that this algorithm can be viewed as a combination of two V-cycles with nested iteration on both sides; this is not especially important except for the fact that the overall performance of FMG does depend on the structure of the cycles employed, and it is known that W-cycles exhibit somewhat better convergence properties than do V-cycles (see [40]). On the other hand, the computational work required in a W-cycle exceeds that of a V-cycle by a significant amount, so we will analyze the simple V-cycles here. It is also important to note that the choice of a four-level scheme is simply to provide a non-trivial example, and as we will indicate in the sequel, larger numbers of levels may be necessary in some cases.

We now describe the manner in which an algorithm corresponding to this schematic would be executed, and we follow this with an analysis of the required total arithmetic. Figure 2.19 indicates that calculations begin with an exact solution on the coarsest ($8h$) grid (recall our definition of exact). This solution is then prolongated to the $4h$ grid via interpolation. In this particular case, the process corresponds to nested iteration because it is the solution, itself, that is prolongated, and not the error. Then a small number of iterations is performed on the $4h$ grid to accomplish smoothing, and the residual is then restricted back to the $8h$ grid for further removal of low-wavenumber errors. Because this is the coarsest grid, the exact solution must be found for the defect equation. The next step is to prolongate this error back to the $4h$ grid where it is used to update the solution. At this point a few smoothing iterations (usually only one) may be applied before the nested iteration step is invoked to transfer the solution from the $4h$ grid to the $2h$ grid. Once this transfer is made, smoothing iterations are again applied before restriction of the residual back to the $4h$ grid. Then, some additional smoothing may be done before restriction back to the $8h$ grid. Next, the exact error is computed one more time, and prolongated back to the $4h$ grid where it is combined with the solution obtained by injection from the $2h$ grid. Nested iteration of the solution is now applied successively between the $4h$ and $2h$ grids, and then after smoothing on the $2h$ grid, one more time between the $2h$ and h grids.

As noted above, the solution on the fine h grid thus obtained can be shown to be within an $\mathcal{O}(1)$ constant of the converged solution on the fine grid. Thus, it should be expected that a few additional smoothing iterations should be sufficient to complete the solution, obtained from only a single pass through the FMG algorithm. In practice, however, this is not always the case. If the desired tolerance has not been met, it is common practice to employ several additional V-cycles. We will, in what follows, assume that a single pass through the FMG procedure is sufficient; modifications needed to account for other situations are not difficult, and they, in general, will not strongly influence the conclusions we will draw.

It will be convenient to refer to the (2-D) h grid as the N-point grid, the $2h$ grid as the $N/4$-point grid, *etc.* We will first estimate the number of arithmetic operations required for exact solves on the coarsest ($N/64$-point) grid. We see from Fig. 2.19 that these must be performed three times during the course of executing the complete algorithm. We will assume that a standard, optimal relaxation scheme is being used for these solves, but clearly other alternatives are available (*e.g.*, possibly conjugate gradient with ILU preconditioning). It follows that, up to an $\mathcal{O}(1)$ constant, the total arithmetic is $3A_{ops/pt}(N/64)^{3/2}$, where $A_{ops/pt}$ is the number of arithmetic operations per grid point, which in the present case is $\sim \mathcal{O}(10^1)$. We are assuming that there is negligible difference between the solves for the actual solution and those for the error, as should be the case.

On the first pass through the $N/16$-point grid smoothing iterations will be required, and it is rather common to use only a small number—typically, about three. This will be assumed here. The second pass through this grid is part of a nested iteration, and as such very little smoothing is utilized. Seldom is more than a single iteration employed, and often no iterations are done. Here, we will assume one iteration in each such case. The next operation on the $N/16$-point grid corresponds to calculating the error from a defect equation; again, only about three iterations are usually employed for this. The final calculation on the $N/16$-point grid involves combining the exact error computed on the $N/64$-point grid with an injected solution from the $N/4$-point grid. There is considerably less arithmetic required for this than for an actual iteration, so we will ignore it. One iteration will be performed prior to interpolation to the $N/4$-point grid. This leads to the equivalent of eight iterations on the $N/16$-point grid.

We next consider work required on the $N/4$-point grid. This is first encountered in the fine-grid smoothing mode after prolongation (in the form of nested iteration) from the $N/16$-point grid at the beginning of the $\ell = 3$ V-cycle. We will assume application of the usual three iterations in this context. The next time this grid is used is in the nested iteration leading to the finest grid. As before, we will use only a single iteration for this. Thus, the equivalent of four iterations are performed on this grid. Following this, the solution is interpolated to the finest (N-point) grid where three additional smoothing iterations are performed, completing the FMG process.

The total arithmetic expended in iterations is obtained by summing these contributions. We have

$$\begin{aligned} A_{ops,iter} &= A_{ops/pt}\left[\underbrace{3\left(\frac{N}{64}\right)^{3/2}}_{8h\text{-grid}} + \underbrace{8\left(\frac{N}{16}\right)}_{4h\text{-grid}} + \underbrace{4\left(\frac{N}{4}\right)}_{2h\text{-grid}} + \underbrace{3\,(N)}_{h\text{-grid}}\right] \\ &\le A_{ops/pt}\left(0.006N^{3/2} + 5N\right) . \end{aligned}$$

There are several consequences of this equation worth noting. First, over a range of N that is somewhat moderate, the $5N$ term will dominate the total arithmetic arising from iterations. But as N is continually increased, the $N^{3/2}$ term will become increasingly important. Thus, for a fixed algorithm, *i.e.*, fixed number of levels and fixed relation between the grid-point sets (factor of two increase in grid spacing in the present case), total arithmetic cannot scale with N, as is widely claimed. Second, there are ways to manage the gridding to maintain dominance of the $\mathcal{O}(N)$ term. The simplest is to add levels as the number N increases so that, *e.g.*, $N/64$ is replaced by, say $N/256$, at some point; but there are other alternatives. Finally, in general, the total arithmetic is quite reasonable, and we will see that this is still the case after accounting for the work involved in restrictions and prolongations, as we will do next.

To complete our picture of FMG we now consider the work required to perform the necessary restrictions and prolongations. We will cast this in the context of the typical operators given in Eq. (2.142) and Algorithm 1.5. In particular, it can be seen from (2.142) that the total arithmetic required for restriction is approximately 50% greater than that needed per point for a typical relaxation iteration. At the same time, the linear interpolation formulas of the prolongation algorithm show that this step requires somewhat less than 50% of an iteration per grid point. Furthermore, in the FMG scheme of Fig. 2.19 the number of prolongations exceeds that of restrictions, so counting each of these as one iteration will provide an upper bound for this auxiliary arithmetic.

From the figure it is easily seen that there are two restrictions to the $N/64$-point grid, and thus an equivalent of $2(N/64)$ units of work. There is one additional restriction to the $N/16$-point grid with a corresponding $N/16$ units of work. There are three prolongations of the $N/64$-point grid results, so this adds $3(N/16)$ work units, and there are two prolongations from the $N/16$-point grid to the $N/4$-point grid, giving $2(N/4)$ units of work. Finally, the prolongation from the $N/4$-point grid to the N-point grid gives an additional N units. Thus, the total arithmetic for restriction and prolongation is

$$\begin{aligned}
A_{ops,r/p} &= \underbrace{\overbrace{2\left(\frac{N}{64}\right)}^{8h} + \overbrace{\left(\frac{N}{16}\right)}^{4h}}_{restriction} + \underbrace{\overbrace{3\left(\frac{N}{16}\right)}^{4h} + \overbrace{2\left(\frac{N}{4}\right)}^{2h} + \overbrace{N}^{h}}_{prolongation} \\
&< 2N\,,
\end{aligned}$$

(times the arithmetic operations per point, $A_{ops/pt}$) which implies that for the particular FMG form we are considering, all work associated with restrictions and prolongations accounts for less than two iterations on the finest grid. Thus, the total arithmetic to complete one execution of the FMG algorithm depicted in Fig. 2.19 is

$$A_{total} < A_{ops/pt}\left(0.006N^{3/2} + 7N\right) . \tag{2.145}$$

It is also of interest to compare this total arithmetic with that which would be required by a basic (optimal) relaxation method such as SOR on a grid of N points. Clearly, this would be $\sim A_{ops/pt}N^{3/2}$, so it is easily seen that the speed up obtained from the multigrid formulation is quite considerable even if several FMG cycles are required.

2.7.5 Some concluding remarks

In this section we have provided a basic introduction to multigrid methods, but this has been, admittedly, rather cursory. There are many good references available from which the interested reader can obtain much additional information (see [37], [38], [40] among many others). Here, we will provide a few clarifying remarks, some of which can be found in the literature, and some of which may not. We will first comment on MG convergence rate, which has been widely studied. In particular, as noted in [40], the convergence rate of 2-grid multigrid is, in fact, theoretically the same as for full multigrid; *viz.*, independent of discretization step size. But in practice this is not usually observed, and the 2-grid MG rate is sometimes quoted as $\mathcal{O}(N \log N)$. One can readily see from our analysis of total arithmetic that if only two grids are employed, unless there is an extreme difference in grid spacing between the fine and coarse grids, the coarse-grid (required) exact solutions will ultimately dominate the arithmetic, so the convergence rate will simply be the rate associated with these solves, and this will not lead to $\mathcal{O}(N)$ arithmetic—and in most cases, it is not even as minimal as $\mathcal{O}(N \log N)$.

We next need to provide a comment concerning the relationship between restriction and prolongation. We hinted earlier that these are related: in a somewhat loose sense they are inverses of one another. There are analyses showing that for these two operations to work together properly it is necessary that their matrix representations be transposes of one another. Hence, one might first select an interpolation scheme for prolongation, and then construct its transposed matrix to determine the restriction operator. But in practice this is often not done. For example, if injection is used for restriction, no interpolation scheme employed for prolongation will have a matrix that is the transpose of the injection matrix. This is one of the main arguments against employing injection. In addition is the fact that typical restriction operators such as that given in Eq. (2.142) are actually low-pass filters. Hence, they provide additional smoothing of the fine-grid residual, permitting use of fewer iterations on the fine grid, as already noted.

Finally, we must comment on the overall effectiveness of removing low-wavenumber error on course grids where it is best removed. In principle this seems like a perfectly reasonable notion. But consider the situation (often encountered, *e.g.* in CFD) wherein solution resolution on the finest grid is at best sufficient, and possibly not even that. We perform some smoothing iterations to estimate this solution, and then we restrict to a yet coarser grid to remove low-wavenumber errors. Suppose we use the typical restriction operator of Eq. (2.142) to guarantee that the residuals transferred to the coarse grid are smooth. Then on this coarse grid we solve the defect equation. Since results on the fine grid are already barely resolved, the defect produced on the coarse grid will not be properly resolved; it will be aliased, and the associated error will be interpolated back to the fine grid to be used as a "correction" to the solution. It is rather clear that such an approach has little chance for success and, indeed, success of multigrid in problems where production of aliased solutions is likely has not been altogether satisfactory—$\mathcal{O}(N)$ arithmetic is essentially never observed. As a consequence, we must be somewhat careful in our assessment of the performance capabilities of multigrid methods, and correspondingly in our choice of problems to which we attempt to apply them.

2.8 Domain-Decomposition Methods

The basic ideas underlying *domain decomposition* were first proposed by Schwarz [41] in 1870 in the context of constructing analytical solutions to elliptic PDEs on irregularly-shaped domains. Schwarz was able to prove convergence of an iterative procedure in which the domain is divided into two (or more) regularly-shaped, overlapping subdomains on which exact solutions could be constructed, but on which some of the boundary conditions required by one subdomain must be determined from the solution in the other subdomain.

This approach was described by Mitchell and Griffiths [16], and earlier by Mitchell [42], to aid the implementation of ADI methods on L-shaped domains, as depicted in Fig. 2.20. It is easily seen from this

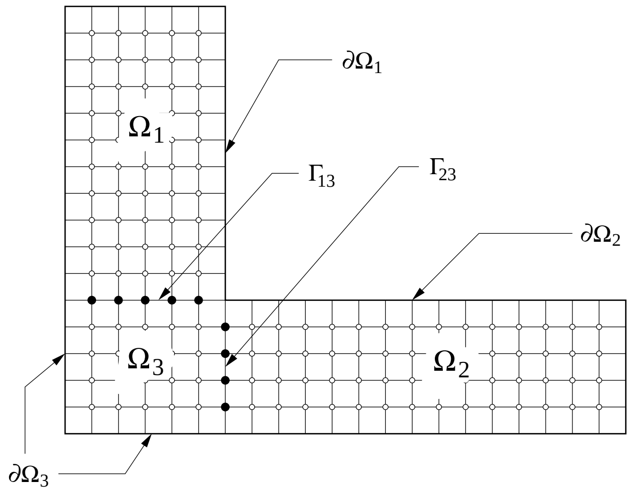

Figure 2.20: L-shaped grid depicting basic domain-decomposition approach.

figure that ADI-like procedures would be somewhat difficult to construct for the domain, as a whole, but they are readily implemented on either $\Omega_1 \cup \Omega_3$ or $\Omega_2 \cup \Omega_3$. On the other hand, a segment of the boundary (corresponding to the sets of filled grid points of the figure) is not specified for either of these subdomains. However, if initially we guess the boundary values, say on Γ_{13}, and solve the PDE in $\Omega_2 \cup \Omega_3$, then we can use the computed result to provide the necessary boundary values on Γ_{23} for the subdomain $\Omega_1 \cup \Omega_3$. Then, the PDE can be solved in this region, in turn yielding a better estimate for boundary conditions on Γ_{13}. This process is continued iteratively until a desired level of convergence for the solution on the whole domain is achieved.

We can formally summarize this for Poisson–Dirichlet problems as follows. Consider the problem

$$-\Delta u = f(x,y)\,, \qquad (x,y) \in \Omega \ (\equiv \Omega_1 \cup \Omega_2 \cup \Omega_3)\,,$$
$$u(x,y) = g(x,y)\,, \qquad (x,y) \in \partial\Omega\,.$$

Following the heuristic description given above, we construct the following two-step process for completing one full iteration. First solve

$$\begin{aligned} -\Delta u^{(n+1/2)} &= f(x,y)\,, \qquad (x,y) \in \Omega_1 \cup \Omega_3 \\ u(x,y) &= g(x,y)\,, \qquad (x,y) \in \partial\Omega_1 \cup \partial\Omega_3\,, \\ u(x,y) &= u^{(n)}(x,y)\,, \qquad (x,y) \in \Gamma_{23}\,. \end{aligned}$$

Then solve

$$\begin{aligned} -\Delta u^{(n+1)} &= f(x,y)\,, \qquad (x,y) \in \Omega_2 \cup \Omega_3 \\ u(x,y) &= g(x,y)\,, \qquad (x,y) \in \partial\Omega_2 \cup \partial\Omega_3\,, \\ u(x,y) &= u^{(n+1/2)}(x,y)\,, \qquad (x,y) \in \Gamma_{13}\,. \end{aligned}$$

The motivation for employing such an approach is now rather different from that of simply trying to lessen the effort of applying ADI as was the case in the late 1960s. In particular, we will see that with some modification, this basic idea can be used with great success to both provide a straightforward means to parallelization on modern supercomputers and at the same time reduce the total required arithmetic for solving large sparse systems arising from discrete approximation of elliptic PDEs to little more than $\mathcal{O}(N)$. We will not attempt an in-depth presentation in these lectures; the reader is referred to Smith *et al.* [43] for such a treatment. Our goal here is to provide sufficient introduction to permit further study from such references. Thus, the remainder of this section will be divided into the following subsections, each one of which pertains to a very fundamental part of the theory of domain decomposition. We will first provide a more formal treatment of the original method, as proposed by Schwarz [41]. In modern terminology this is referred to as the *alternating Schwarz method*, and it will be clear from our discussions that it is not easily parallelized. Second, we will consider some details of another rather old idea, use of the Schur complement, and the problems encountered in attempting to use it in a modern computational environment. The next two sections will consist of expositions of the modern versions of the alternating Schwarz procedure: multiplicative and additive Schwarz methods. We will discuss the advantages and disadvantages of these with respect to one another in the context of required arithmetic and parallel implementations.

Finally, we will briefly introduce the most recent development in domain decomposition and multigrid methods, namely, multi-level domain decomposition. This approach is quite possibly the closest that one will be able to come in solving a wide range of elliptic problems in only $\mathcal{O}(N)$ arithmetic operations. It is worthwhile to recognize that these two methods, multigrid and domain decomposition, are in a sense complementary, and one would expect them to work well together. In particular, recall that the spectral radius of typical relaxation methods is usually similar in form to that of Jacobi iterations, given in Eq. (2.39) for Laplace–Dirichlet problems as the Taylor expansion

$$\rho(B) = 1 - \frac{1}{4}\left[\left(\frac{\pi}{a}\right)^2 + \left(\frac{\pi}{b}\right)^2\right] h^2 + \mathcal{O}(h^4)\,,$$

where a and b are lengths of the sides of a rectangular domain whose lower left-hand corner is at the origin. It is important to note that there are two distinct ways in which this quantity can be made small. First, we can make h large; this is the approach taken in multigrid methods when carrying out coarse-grid corrections, and it can be very effective. On the other hand, we might instead make a and/or b small. This is equivalent to what is done in domain decomposition, and it is also very effective. But a further advantage is that it does not introduce the under-resolution problem inherent in coarse-grid corrections of multigrid methods. Thus, for many practical problems we would expect that domain decomposition might be preferred. But we will see that the best approach seems to be to directly exploit complementarity of the two methods, and use them together. It is important to recognize that, just as is true for multigrid methods, domain-decomposition algorithms must employ more fundamental solution techniques such as SOR or conjugate gradient, *etc.*; they do not, as a class of methods, provide any new basic linear equation-solving approaches.

2.8.1 The alternating Schwarz procedure

In this section we introduce a more formal representation of the simple algorithm discussed above to provide the background for understanding domain decomposition in general. The problem considered by Schwarz [41] was solution of Dirichlet problems for the Poisson equation on the "keyhole-shaped" domain shown in Fig. 2.21. That is, we solve

$$-\Delta u = f(x,y)\,, \qquad (x,y) \in \Omega_1 \cup \Omega_2$$

with

$$u = g(x,y)\,, \qquad (x,y) \in \partial\Omega_1 \backslash \Gamma_1 \cup \partial\Omega_2 \backslash \Gamma_2\,.$$

The corresponding numerical procedure is analogous to that described for the preceding problem, but

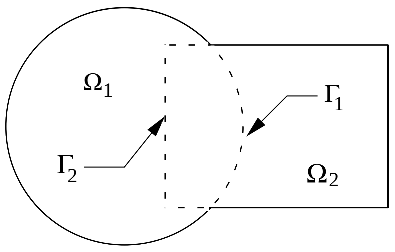

Figure 2.21: Keyhole-shaped domain $\Omega_1 \cup \Omega_2$ considered by Schwarz [41].

with the additional complication of requiring interpolation between the two different grids (one of which will now probably be in polar coordinates) to obtain grid function values on Γ_1 after solving the equation in Ω_2, and similarly on Γ_2 after solving in Ω_1. Thus, the alternating Schwarz method for the analytical PDE problem can now be written as a two-step procedure:

1. Solve

   $$-\Delta u_1^{(n+1/2)} = f_1 \qquad \text{in } \Omega_1$$

   with

   $$u_1^{(n+1/2)} = g_1 \qquad \text{on } \partial\Omega_1 \backslash \Gamma_1\,,$$

   and

   $$u_1^{(n+1/2)} = \mathcal{I}_{\Omega_2}^{\Gamma_1} u_2^{(n)} \qquad \text{on } \Gamma_1\,.$$

2. Solve

   $$-\Delta u_2^{(n+1)} = f_2 \qquad \text{in } \Omega_2$$

with

$$u_2^{(n+1)} = g_2 \qquad \text{on } \partial\Omega_2\backslash\Gamma_2 \,,$$

and

$$u_2^{(n+1)} = \mathcal{I}_{\Omega_1}^{\Gamma_2} u_1^{(n+1/2)} \qquad \text{on } \Gamma_2 \,.$$

In these expressions, $\mathcal{I}_{\Omega_i}^{\Gamma_j}$ is an interpolation operator from the Ω_i subdomain to the Γ_j "internal" boundary that it contains.

If we let A_{Ω_i}, $i = 1,2$, denote the matrix of coefficients arising from discretizing the Laplacian on the subdomain Ω_i, and decompose the solution vector as $u = (u_{\Omega_1}, u_{\partial\Omega_1\backslash\Gamma_1}, u_{\Gamma_1}, u_{\Omega_2}, u_{\partial\Omega_2\backslash\Gamma_2}, u_{\Gamma_2})^T$, then we can express a discrete version of the above procedure in the form of the following pseudo-language algorithm.

Algorithm 2.6 (Alternating Schwarz domain decomposition) *Suppose n iterations have been completed. To advance calculations to the $n+1$ level, carry out the following steps.*

1. *Solve*

$$A_{\Omega_1} u_{\Omega_1}^{(n+1/2)} = f_1 \qquad \textit{in } \Omega_1$$

with

$$u_{\partial\Omega_1\backslash\Gamma_1}^{(n+1/2)} = g_1 \qquad \textit{on } \partial\Omega_1\backslash\Gamma_1 \,,$$

and

$$u_{\Gamma_1}^{(n+1/2)} = \mathcal{I}_{\Omega_2}^{\Gamma_1} u_{\Omega_2}^{(n)} \qquad \textit{on } \Gamma_1 \,.$$

2. *Solve*

$$A_{\Omega_2} u_{\Omega_2}^{(n+1)} = f_2 \qquad \textit{in } \Omega_2$$

with

$$u_{\partial\Omega_2\backslash\Gamma_2}^{(n+1)} = g_2 \qquad \textit{on } \partial\Omega_2\backslash\Gamma_2 \,,$$

and

$$u_{\Gamma_2}^{(n+1)} = \mathcal{I}_{\Omega_1}^{\Gamma_2} u_{\Omega_1}^{(n+1/2)} \qquad \textit{on } \Gamma_2 \,.$$

3. *Test convergence, and return to 1. if necessary.*

The two separate steps of this algorithm can be combined in a single matrix representation that explicitly includes the equations for the interior boundary values.

$$\begin{bmatrix} A_{\Omega_1} & A_{\partial\Omega_1\backslash\Gamma_1} & A_{\Gamma_1} & 0 & \cdots & 0 \\ 0 & I & 0 & 0 & \ddots & \vdots \\ \vdots & \ddots & I & -\mathcal{I}_{\Omega_2}^{\Gamma_1} & 0 & 0 \\ \vdots & & \ddots & A_{\Omega_2} & A_{\partial\Omega_1\backslash\Gamma_2} & A_{\Gamma_2} \\ 0 & & & \ddots & I & 0 \\ -\mathcal{I}_{\Omega_1}^{\Gamma_2} & 0 & \cdots & \cdots & 0 & I \end{bmatrix} \begin{bmatrix} u_{\Omega_1} \\ u_{\partial\Omega_1\backslash\Gamma_1} \\ u_{\Gamma_1} \\ u_{\Omega_2} \\ u_{\partial\Omega_2\backslash\Gamma_2} \\ u_{\Gamma_2} \end{bmatrix} = \begin{bmatrix} f_1 \\ g_1 \\ 0 \\ f_2 \\ g_2 \\ 0 \end{bmatrix} . \tag{2.146}$$

We have suppressed formal iteration indexing here for brevity, but we now reinsert this notation as we expand Eq. (2.146) to obtain the following systems.

$$\begin{aligned} A_{\Omega_1} u_{\Omega_1}^{(n+1/2)} &= f_1 - A_{\partial\Omega_1\backslash\Gamma_1} u_{\partial\Omega_1\backslash\Gamma_1}^{(n+1/2)} - A_{\Gamma_1} u_{\Gamma_1}^{(n)} \\ &= f_1 - A_{\partial\Omega_1\backslash\Gamma_1} g_1 - A_{\Gamma_1} \mathcal{I}_{\Omega_2}^{\Gamma_1} u_{\Omega_2}^{(n)} \,, \end{aligned}$$

and similarly,

$$A_{\Omega_2} u_{\Omega_2}^{(n+1)} = f_2 - A_{\partial\Omega_2\backslash\Gamma_1} g_2 - A_{\Gamma_2} \mathcal{I}_{\Omega_1}^{\Gamma_2} u_{\Omega_1}^{(n+1/2)} .$$

We can write these as

$$\begin{aligned} u_{\Omega_1}^{(n+1/2)} &= A_{\Omega_1}^{-1} \left(f_1 - A_{\partial\Omega_1\backslash\Gamma_1} g_1 - A_{\Gamma_1} \mathcal{I}_{\Omega_2}^{\Gamma_1} u_{\Omega_2}^{(n)} \right) \\ u_{\Omega_2}^{(n+1)} &= A_{\Omega_2}^{-1} \left(f_2 - A_{\partial\Omega_2\backslash\Gamma_2} g_2 - A_{\Gamma_2} \mathcal{I}_{\Omega_1}^{\Gamma_2} u_{\Omega_1}^{(n+1/2)} \right) . \end{aligned}$$

Then addition and subtraction of the appropriate $u_{\Omega_i}^{(n)}$, $i = 1, 2$, on the right-hand side of each leads to

$$u_{\Omega_1}^{(n+1/2)} = u_{\Omega_1}^{(n)} + A_{\Omega_1}^{-1} \left(f_1 - A_{\Omega_1} u_{\Omega_1}^{(n)} - A_{\partial\Omega_1\backslash\Gamma_1} g_1 - A_{\Gamma_1} \mathcal{I}_{\Omega_2}^{\Gamma_1} u_{\Omega_2}^{(n)} \right) , \tag{2.147a}$$

$$u_{\Omega_2}^{(n+1)} = u_{\Omega_2}^{(n)} + A_{\Omega_2}^{-1} \left(f_2 - A_{\Omega_2} u_{\Omega_2}^{(n)} - A_{\partial\Omega_2\backslash\Gamma_2} g_2 - A_{\Gamma_2} \mathcal{I}_{\Omega_1}^{\Gamma_2} u_{\Omega_1}^{(n+1/2)} \right) . \tag{2.147b}$$

We now observe that since the last two terms on the right-hand side of each of these equations are known, we can define

$$\tilde{f}_i \equiv f_i - A_{\partial\Omega_i\backslash\Gamma_j} g_i - A_{\Gamma_i} \mathcal{I}_{\Omega_j}^{\Gamma_i} u_{\Omega_j}^{(n)} , \qquad i = 1, 2, \quad j = 2, 1 ,$$

by replacing $u_{\Omega_1}^{(n+1/2)}$ with $u_{\Omega_1}^{(n)}$ in the second equation. This leads to

$$u_{\Omega_1}^{(n+1/2)} = u_{\Omega_1}^{(n)} + A_{\Omega_1}^{-1} \left(\tilde{f}_1 - A_{\Omega_1} u_{\Omega_1}^{(n)} \right) , \tag{2.148a}$$

$$u_{\Omega_2}^{(n+1)} = u_{\Omega_2}^{(n)} + A_{\Omega_2}^{-1} \left(\tilde{f}_2 - A_{\Omega_2} u_{\Omega_2}^{(n)} \right) , \tag{2.148b}$$

which has the formal structure of a preconditioned Richardson iteration; at the same time, it can be seen that Eqs. (2.147) are equivalent to a block Gauss–Seidel procedure—but only with respect to the internal boundary, whereas Eqs. (2.148) correspond to usual block Jacobi iteration. Moreover, it is clear that residuals appear in both of these systems, and in this sense they are closely related to the fine-grid updates by defect equation results in multigrid methods.

An important observation to make at this point is that the grid point values in the overlap region of Fig. 2.21 must be computed twice for each complete iteration (as was the case with the simpler example provided earlier in connection with ADI implementations) in order to generate all necessary internal boundary data. Obviously, this can represent a significant amount of arithmetic in some cases, and beyond this is the fact that unless the simple Richardson formulation of Eqs. (2.148) is used, it is not possible to parallelize the algorithm (although the chosen solution procedure(s) within the algorithm might be parallelized)—a situation similar to that of multigrid methods.

This leads us to seek modifications that will make the overall algorithm more efficient. It is clear from Fig. 2.21 that if we did not require overlap between the two subdomains Ω_1 and Ω_2 beyond the interior boundary Γ_1, much less calculation would be required. Moreover, if we could in some manner obtain the solution values on Γ_1 independently, then parallelization of the remaining algorithm would be easy. In the next subsection we treat a method that in principle allows us to accomplish both of these goals, but which is itself not directly related to Schwarz methods. In a later subsection we will see modifications to Algorithm 2.6 that will achieve similar results.

2.8.2 The Schur complement

In this subsection we will introduce an exact procedure for analytically obtaining selected portions of a solution vector in terms of only the (known) matrix coefficients. One might view this as a selective form of direct elimination performed on blocks of solution vector elements. The advantage of this is that it, in principle, permits us to solve for the interior boundary components. Then after obtaining these we are able to calculate the remaining portions of the solution vector in parallel.

To make these notions transparent, we will work with a simple rectangular domain that has been divided into two subdomains, as depicted in Fig. 2.22. We label the single interior boundary as Γ_{12} and decompose the solution vector as $u = (u_1, u_2, u_3)^T$, with u_3 being the subvector containing the grid function values on Γ_{12}. We have also shown a typical finite-difference five-point mesh stencil in the figure to indicate the manner in which coupling between the subdomains will occur and thus, why the results obtained as the Schur complement are not trivial.

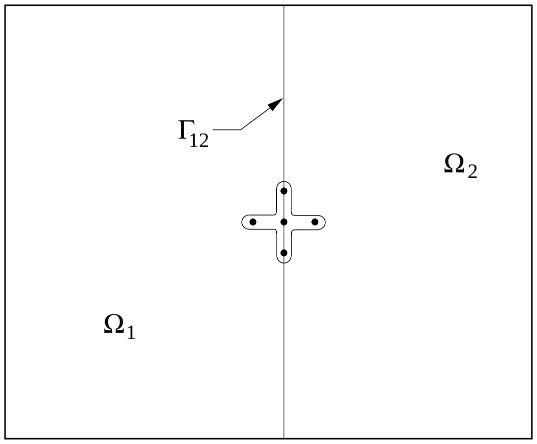

Figure 2.22: Simple two-subdomain problem to demonstrate Schur complement.

It is now possible to cast the matrix of coefficients of the discrete equations into a form analogous to that used earlier to describe the alternating Schwarz method. Namely, we have

$$\begin{bmatrix} A_{11} & 0 & A_{13} \\ 0 & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix} . \tag{2.149}$$

From this we see that if u_3 were known we could simultaneously solve for u_1 and u_2 in parallel since, formally

$$u_1 = A_{11}^{-1}\left[f_1 - A_{13}u_3\right], \qquad \text{and} \qquad u_2 = A_{22}^{-1}\left[f_2 - A_{23}u_3\right].$$

From Eq. (2.149) we also have

$$u_3 = A_{33}^{-1}\left[f_3 - A_{31}u_1 - A_{32}u_2\right],$$

and substitution of the above formulas into this yields

$$u_3 = A_{33}^{-1}\left[f_3 - A_{31}A_{11}^{-1}f_1 - A_{32}A_{22}^{-1}f_2\right] + A_{33}^{-1}\left(A_{31}A_{11}^{-1}A_{13} + A_{32}A_{22}^{-1}A_{23}\right)u_3\,,$$

or

$$\underbrace{\left[A_{33} - \left(A_{31}A_{11}^{-1}A_{13} + A_{32}A_{22}^{-1}A_{23}\right)\right]}_{\text{Schur complement of } A} u_3 = f_3 - A_{31}A_{11}^{-1}f_1 - A_{32}A_{22}^{-1}f_2\,. \tag{2.150}$$

Thus, we can determine solution values u_3 on the internal boundary Γ_{12} by inverting the left-hand side matrix, the *Schur complement* of the original matrix. Moreover, the vector u_3 obtained from this can be substituted back into the preceding expressions for u_1 and u_2 to completely remove dependence of the internal boundary. This is, of course, not unexpected since the "boundary" is an artificial one.

It is also of interest to compare these equations for u_1 and u_2 with the equations for the alternating Schwarz method treated earlier. We can write the present equations as

$$u_1 = u_1 + A_{11}^{-1}\left[f_1 - A_{11}u_1 - A_{13}u_3\right],$$

and

$$u_2 = u_2 + A_{22}^{-1}\Big[f_2 - A_{22}u_2 - A_{23}u_3\Big] .$$

After substituting the expression for u_3, adding appropriate iteration counters will lead to the alternating Schwarz iteration.

There are several observations to be made at this point. First, there are two distinct advantages to this domain-decomposition approach. The first, obvious, one is its "embarassing" parallelism, provided u_3 can actually be determined. The second, somewhat more subtle, thing to note is that both of the two systems arising from the decomposition are much smaller (in terms of the number of grid points) than the original system and hence, can be solved much more rapidly.

But there are also disadvantages that restrict applicability of this approach. Not only is the Schur complement not sparse, but it formally requires inversion of A_{11} and A_{22} for its construction, as does evaluation of the right-hand side of (2.150). While there is significant potential for parallelism, the total required arithmetic would be large. Because of this, it is usual to replace these inverses with approximations that are sparse and allow very inexpensive matrix multiplications. This implies that u_3 will no longer be exact, and that the overall process will require iteration. Much effort has gone into developing preconditioners to speed these iterations, but it must be emphasized that this has been done mainly in the context of rather simple model problems. Finally, we note that even for these model problems, use of the Schur complement becomes much more complicated if more subdomains are defined in order to utilize larger numbers of parallel processors; this is especially true if the original domain is divided both horizontally and vertically as shown in Fig. 2.23. In particular, we note that there are now two new types of boundary points requiring

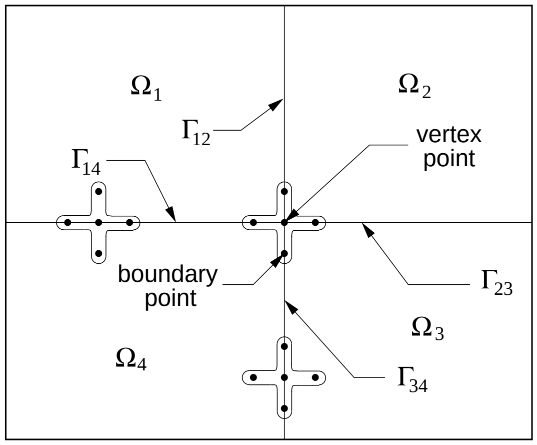

Figure 2.23: Four-subdomain problem demonstrating types of points required for Schur complement construction.

separate treatment, in addition to the usual points of the interior boundaries, Γ_{ij}. Namely, first are points having the property that each occurs in the calculations of two distinct boundaries; we have termed these simply "boundary points." Second, in the case displayed in Fig. 2.23 there is a single point associated with all four interior boundaries; these are called "vertex points." Each of these types requires special treatment.

2.8.3 Multiplicative and additive Schwarz methods

Now that we have seen a formal procedure for expressing internal boundary values directly via the Schur complement, we will now return to the Schwarz-like methods to find what are essentially approximations to the Schur complement. It is clear from the preceding that this will be necessary for computational efficiency. What is at question is whether good approximations can be found that do not destroy parallelism. As we have noted earlier, there are two main classes of Schwarz-like procedures by means of which the prominent overlap of the alternating Schwarz method can be reduced. These are multiplicative and additive Schwarz techniques; we will describe each of these in what follows.

Multiplicative Schwarz method

Consider a domain such as the one depicted in Fig. 2.22 used to study the Schur complement, but now permit some overlap, as shown in Fig. 2.24, part (a). We will assume that discretization has resulted in

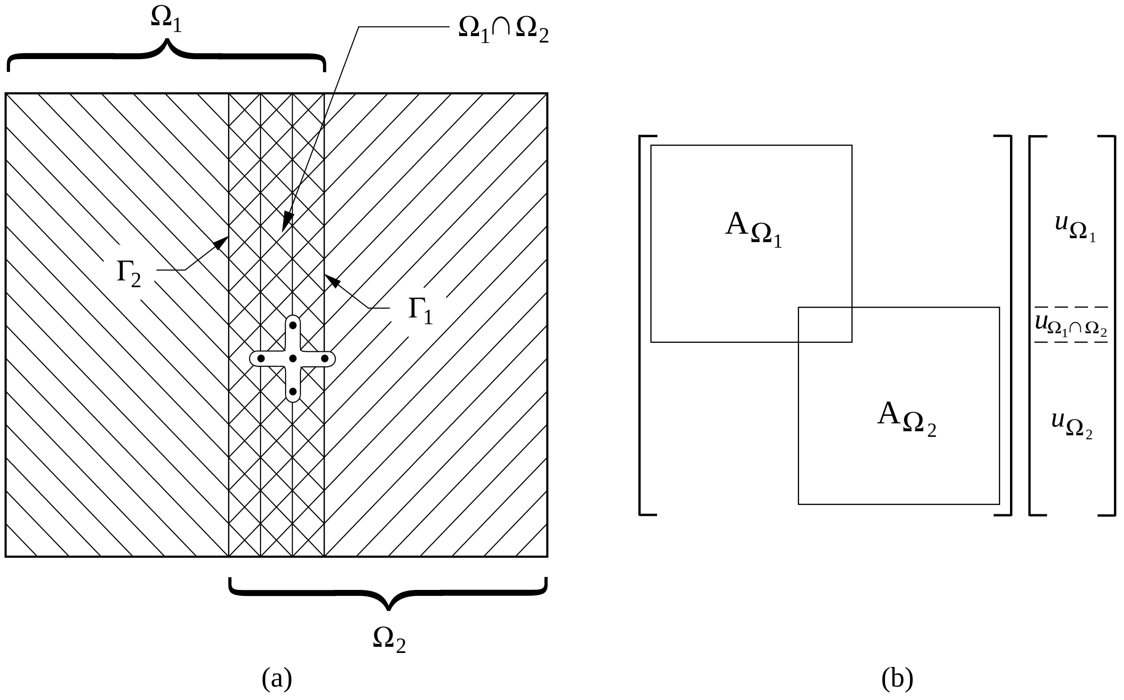

Figure 2.24: Domain decomposition with two overlapping subdomains; (a) domain geometry, (b) matrix structure.

matched grids so that no interpolation operators will be needed to transfer information between subdomains. Clearly, this simplification can be dropped when it is necessary to do so.

If we absorb boundary data into the f_is as done previously, but now suppress the tilde notation, we can obtain equations similar in form to those given earlier in the context of alternating Schwarz methods:

$$u_{\Omega_1}^{(n+1/2)} = u_{\Omega_1}^{(n)} + A_{\Omega_1}^{-1}\left(f_1 - A_{\Omega_1} u_{\Omega_1}^{(n)} - A_{\Gamma_1} u_{\Gamma_1}^{(n)}\right) ,$$

$$u_{\Omega_2}^{(n+1)} = u_{\Omega_2}^{(n)} + A_{\Omega_2}^{-1}\left(f_2 - A_{\Omega_2} u_{\Omega_2}^{(n)} - A_{\Gamma_2} u_{\Gamma_2}^{(n+1/2)}\right) .$$

It is clear that the overlap region is again being computed twice with each complete iteration, and that this formulation is equivalent to block Gauss–Seidel methods with respect to internal boundaries. We also observe that dependence on these boundaries can be formally removed by expressing these equations as

$$u_{\Omega_1}^{(n+1/2)} = u_{\Omega_1}^{(n)} + A_{\Omega_1}^{-1}\left(f_1 - A_{\Omega_1} u_{\Omega_1}^{(n)} - A_{\Omega\backslash\Omega_1} u_{\Omega\backslash\Omega_1}^{(n)}\right) , \tag{2.151a}$$

$$u_{\Omega_2}^{(n+1)} = u_{\Omega_2}^{(n)} + A_{\Omega_2}^{-1}\left(f_2 - A_{\Omega_2} u_{\Omega_2}^{(n)} - A_{\Omega\backslash\Omega_2} u_{\Omega\backslash\Omega_2}^{(n+1/2)}\right) . \tag{2.151b}$$

It should be emphasized, however, that interior boundary dependence is not truly removed, but rather buried in the notation; *viz.*, the solution on these boundaries is contained in $u_{\Omega\backslash\Omega_1}^{(n)}$ and $u_{\Omega\backslash\Omega_2}^{(n+1/2)}$.

To gain further insight into the workings of the multiplicative Schwarz methods (and among other things, see whence the terminology arose) it is useful to construct the fixed-point form of this technique. We begin by introducing the formal representation

$$u^{(n+1/2)} = u^{(n)} + \begin{pmatrix} A_{\Omega_1}^{-1} & 0 \\ 0 & 0 \end{pmatrix} \left(f - Au^{(n)}\right) , \tag{2.152a}$$

$$u^{(n+1)} = u^{(n+1/2)} + \begin{pmatrix} 0 & 0 \\ 0 & A_{\Omega_2}^{-1} \end{pmatrix} \left(f - Au^{(n+1/2)}\right) . \tag{2.152b}$$

It is important to recognize that, despite the notation, during each half step only that portion of the solution vector is updated as is indicated by the portion of the original matrix used in that step (see Fig. 2.24. It should also be noted that the form, while still basically a Gauss–Seidel iteration, is somewhat different than that used previously; in particular, the current form is, in some sense, closer to what we usually call Gauss–Seidel iteration than was the previous block form.

We will now introduce some definitions and notation to aid in constructing the fixed-point form of the overall domain-decomposition procedure. We first define restrictions (similar to injections in the multigrid sense) via

$$u_{\Omega_1} \equiv \mathcal{R}_1 u = (I, 0) \begin{pmatrix} u_{\Omega_1} \\ u_{\Omega\backslash\Omega_1} \end{pmatrix} ,$$

and

$$u_{\Omega_2} \equiv \mathcal{R}_2 u = (0, I) \begin{pmatrix} u_{\Omega\backslash\Omega_2} \\ u_{\Omega_2} \end{pmatrix} .$$

From this it is easily seen that

$$A_{\Omega_i} = \mathcal{R}_i A \mathcal{R}_i^T , \qquad i = 1, 2 , \tag{2.153}$$

since the identity matrix appearing in each $\mathcal{R}_i$ is just the right size to pick out the corresponding u_{Ω_i}. It is also important to note that the $\mathcal{R}_i$s can be defined simply in terms of an index set—no matrix $\mathcal{R}_i$ is stored, and certainly we would not actually perform a multiplication to construct u_{Ω_i}.

Now define

$$B_i = \mathcal{R}_i^T \left(\mathcal{R}_i A \mathcal{R}_i^T\right)^{-1} \mathcal{R}_i , \qquad i = 1, 2 . \tag{2.154}$$

These matrices correspond to the following sequence of steps: *i*) restrict the residual appearing in Eqs. (2.152) to one subdomain; *ii*) solve the problem on that subdomain to generate a correction, and then *iii*) prolongate the correction back to the whole domain. (The similarities to multigrid procedures are quite pronounced.) Again, we note that in practical calculations these matrices are never actually formed; they merely provide a conceptual representation for analysis of the method.

We can now use these definitions in Eqs. (2.152) for the multiplicative Schwarz method to obtain

$$\begin{aligned} u^{(n+1/2)} &= u^{(n)} + \mathcal{R}_1^T \left(\mathcal{R}_1 A \mathcal{R}_1^T\right)^{-1} \mathcal{R}_1 \left(f - Au^{(n)}\right) , \\ u^{(n+1)} &= u^{(n+1/2)} + \mathcal{R}_2^T \left(\mathcal{R}_2 A \mathcal{R}_2^T\right)^{-1} \mathcal{R}_2 \left(f - Au^{(n+1/2)}\right) , \end{aligned}$$

or

$$u^{(n+1/2)} = u^{(n)} + B_1 \left(f - Au^{(n)}\right) , \tag{2.155a}$$

$$u^{(n+1)} = u^{(n+1/2)} + B_2 \left(f - Au^{(n+1/2)}\right) . \tag{2.155b}$$

Combining these two steps in the usual way yields

$$\begin{aligned} u^{(n+1)} &= u^{(n)} + B_1 \left(f - Au^{(n)}\right) + B_2 \left[f - A\left(u^{(n)} + B_1\left(f - Au^{(n)}\right)\right)\right] \\ &= u^{(n)} + \left[B_1 + B_2 - B_2 A B_1\right] \left(f - Au^{(n)}\right) . \end{aligned}$$

This can be easily rearranged to obtain the usual form of a linear fixed-point iteration,

$$u^{(n+1)} = Gu^{(n)} + k\,,$$

where now

$$G \equiv I - (B_1 + B_2 - B_2AB_1)\,A\,, \tag{2.156}$$

and

$$k \equiv (B_1 + B_2 - B_2AB_1)\,f\,. \tag{2.157}$$

It is of interest to observe that this formula for the iteration matrix G can be factored to obtain

$$G = (I - B_1A)\,(I - B_2A)\ ,$$

and it is from this product form that the terminology *multiplicative Schwarz iteration* arises. It is clear from Eqs. (2.155) that the first step must be completed (*modulo* a possible "wavefront" type algorithm) before the second step is started. Hence, parallelization can only be done within each step, and not across the steps. On the other hand, compared with the alternating Schwarz procedure, recalculation of the overlap region can be avoided. Moreover, multiplicative Schwarz methods, because they correspond to block Gauss–Seidel iteration, are more rapidly convergent than are the alternating Schwarz approaches. These items are reflected in the following pseudo-language algorithm.

Algorithm 2.7 (Two-subdomain multiplicative Schwarz method) *Suppose n iterations have been completed for a 2-subdomain multiplicative Schwarz procedure. To advance to the $n+1$ iteration level, carry out the following steps.*

1. *Use Eq.* (2.155a)*, corresponding to subdomain 1, to advance the solution to iteration level $n+1/2$:*

$$u^{(n+1/2)} = u^{(n)} + B_1\big(f - Au^{(n)}\big)\,.$$

2. *Evaluate Eq.* (2.155b)*,*

$$u^{(n+1)} = u^{(n+1/2)} + B_2\big(f - Au^{(n+1/2)}\big)\,,$$

   to advance the solution to iteration level $n+1$ on subdomain 2.

3. *Compute residual for entire domain, and test convergence.*

4. *If convergence has not been achieved, go to 1.*

Additive Schwarz method

As we have emphasized from the start of our discussion of multiplicative Schwarz methods, they are basically block Gauss–Seidel iterations; as such they are not as easily parallelized as would be block Jacobi iterations. In particular, the matrix B_2AB_1A appearing in (2.156) contains information from the entire matrix A, and hence also from the complete solution vector. But the multiplicative nature of the overall iteration matrix G "hides" this difficulty at the expense of requiring calculation of an intermediate result and a consequent reduction in parallelizability. Despite the fact that Gauss–Seidel methods can be expected to be more rapidly convergent, it is possible that improved parallelization would more than make up for this, and this has motivated study of the so-called *additive Schwarz* procedures in which the matrix B_2AB_1A is ignored. The fixed-point form of these methods can be obtained as follows.

First recall Eq. (2.152b) and replace all entries of $u^{(n+1/2)}$ on the right-hand side with $u^{(n)}$. Clearly, this is then a block Jacobi iteration in the same sense that (2.152) is Gauss–Seidel iteration. We can now simply write these two steps as a single equation to obtain

$$u^{(n+1)} = u^{(n)} + \left[\begin{pmatrix} A_{\Omega_1}^{-1} & 0 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & A_{\Omega_2}^{-1} \end{pmatrix}\right]\Big(f - Au^{(n)}\Big)\,,$$

or upon using Eqs. (2.153) and (2.154),

$$u^{(n+1)} = u^{(n)} + (B_1 + B_2)\left(f - Au^{(n)}\right) . \tag{2.158}$$

This is easily rearranged to obtain the usual fixed-point form with

$$G \equiv I - (B_1 + B_2)\,A\,, \tag{2.159}$$

and

$$k = (B_1 + B_2)\,f \tag{2.160}$$

follow.

It is clear that the formula for G cannot be factored as it could in the multiplicative Schwarz case, so the present form is termed the *additive Schwarz* procedure. Because it is a block Jacobi method, it can be expected to require approximately double the number of iterations needed for multiplicative Schwarz techniques. But now parallelization can be implemented across subdomains (as well as possibly within them), so if there are more than two subdomains an additive Schwarz method presents an advantage. Finally, this approach, like multiplicative Schwarz methods, does not require recalculation of overlap regions and is more rapidly convergent than is the alternating Schwarz technique.

It is easy to see that Eq. (2.158) can be expressed as separate systems of equations on the two different subdomains, *viz.*,

$$u^{(n+1)} = u^{(n)} + B_1\left(f - Au^{(n)}\right) , \tag{2.161a}$$

$$u^{(n+1)} = u^{(n)} + B_2\left(f - Au^{(n)}\right) . \tag{2.161b}$$

This suggests the following pseudo-language algorithm.

Algorithm 2.8 (Two-subdomain additive Schwarz method) *Suppose n iterations of a 2-subdomain additive Schwarz method have been completed. The $n+1^{st}$ iteration is computed as follows.*

```
1. Evaluate Eq. (2.161a) to update results in subdomain 1.
2. Repeat 1. for subdomain 2 using Eq. (2.161b).
3. Test convergence in the entire domain.
4. If iterations have not converged, return to step 1.
```

Comparison of Schwarz methods

From the standpoint of parallelization the advantage of additive Schwarz over all of the other techniques we have considered is quite clear. Application of the two matrices B_1 and B_2 (note that this is simply yet another case of preconditioning) can easily be performed in parallel. On the other hand, the analogy between Gauss–Seidel (multiplicative Schwarz) and Jacobi (additive Schwarz) iterations suggests that the former should converge twice as fast as the latter. Indeed, this is approximately borne out in the numerical experiments presented by Smith *et al.* [43]. These were performed with the following model problem solved with two subdomains similar to what is shown in Fig. 2.22.

$$-\Delta u = xe^y \qquad \text{in} \quad \Omega \equiv (0,2)\times(0,1)\,,$$

with Dirichlet conditions

$$u = -xe^y \qquad \text{on} \ \ \partial\Omega\,.$$

The results reported in [43] are repeated here in the following table.

Table 1. Comparison of required iterations for alternating, multiplicative and additive Schwarz methods.

# of grid points	overlap	alternating	multiplicative	additive
11×6	1	7	2	4
	2	5	2	3
	3	4	2	3
21×11	1	13	3	5
	2	7	2	4
	3	5	2	4
31×16	1	19	3	5
	2	10	3	5
	3	7	2	4

It is easily seen from the table that the expected relationship between multiplicative and additive Schwarz convergence rates is observed; moreover neither is very sensitive to either the amount of overlap or the number of points in the grid. On the other hand, it is clear that the alternating Schwarz method is fairly sensitive to both of these—especially, the former. But it is important to be aware of the context of these results. The computations were performed with an error tolerance of only 10^{-2}, which not only is quite large, but it was applied only to the error-reduction factor; *i.e.*, when the initial error had been reduced to 1% of its original amount, the iterations were declared converged. In fact, almost any method will perform fairly well with such a coarse tolerance because the number of iterations required to achieve this is often sufficiently small that only high-wavenumber errors have been removed, and the method has not yet reached its asymptotic behavior. Moreover, the grids in these cases contain so few points that the results can in no way be considered asymptotic. Thus, the contents of Table 1 must be viewed with considerable caution. Indeed, it is by now widely accepted that the basic domain-decomposition methods discussed to this point are not extremely effective for finely-gridded problems.

Schwarz methods in the context of many subdomains

We now briefly describe how the preceding results can be extended to the case of p subdomains. We demonstrate this specifically for the case of multiplicative Schwarz methods; the additive case is actually easier, as the reader should by now recognize, and we leave its construction for an exercise.

We suppose the solution domain has been subdivided into p subdomains and write the equations constituting the multiplicative Schwarz procedure as

$$\begin{aligned} u^{(n+1/p)} &= u^{(n)} + B_1\left(f - Au^{(n)}\right) \\ u^{(n+2/p)} &= u^{(n+1/p)} + B_2\left(f - Au^{(n+1/p)}\right) \\ &\;\;\vdots \\ u^{(n+1)} &= u^{(n+\frac{p-1}{p})} + B_p\left(f - Au^{(n+\frac{p-1}{p})}\right)\,, \end{aligned} \tag{2.162}$$

where, as in Eq. (2.154), $B_i = \mathcal{R}_i^T\left(\mathcal{R}_i A \mathcal{R}_i^T\right)\mathcal{R}_i$.

It is worth noting here that details of the subdomain geometric structures are not important in this representation because the $\mathcal{R}_i$s are constructed to account for these. It is also useful to consider the effects of such a decomposition on the performance of any typical iterative solver employed on each of the p

subdomains. Suppose $\Omega = \bigcup_{i=1}^{p} \Omega_i$ has been discretized with uniform grid spacing h and that each of the p subdomains is of size H with $h < H < \min(a,b)$, where Ω is the rectangle $(0,a) \times (0,b)$. Now recall the form of the spectral radius for Jacobi iterations given earlier in this section, and also as Eq. (2.39); namely,

$$\rho(B) = 1 - \frac{1}{4}\left[\left(\frac{\pi}{a}\right)^2 + \left(\frac{\pi}{b}\right)^2\right] h^2 .$$

We note that $\rho(\mathcal{L})$ for Gauss–Seidel iterations is simply the square of this quantity, and that $\rho(\mathcal{L}_{\omega_b})$ for optimal SOR is also closely related—but with h^2 replaced by h. We see that substituting H for a and b in this expression results in a significant decrease in the size of $\rho(B)$ if H is sufficiently small, and thus an increase in performance of the iteration scheme. On the other hand, solutions from these individual subdomains must be iteratively combined to construct the complete solution.

We can qualitatively characterize convergence of this iterative process, and hence that of the Schwarz procedures, as follows (see [43] and references therein for more details): let h and H be as above, and let δ denote the width of overlap between subdomains. Then

i) *the number of required iterations grows as* $1/H$;

ii) *if* δ *is kept proportional to* H, *the number of iterations is bounded independently of* h *and* H/h;

iii) *the number of iterations needed for multiplicative Schwarz is approximately one half of that required by additive Schwarz*;

iv) *convergence is poor if* $\delta = 0$ *(no overlap) but improves rapidly with increasing* δ.

We should note that point *i)* above might seem to contradict the conclusions reached in our preceding discussion of effects of spectral radius for solvers on the subdomains. But, in fact, it arises due to a different effect, alluded to above. Namely, as H decreases the number of subdomains, and hence the number of subdomain boundaries, must increase; this makes the problem of piecing together the subdomain solutions more difficult. We will consider this from a somewhat different viewpoint below.

Point *ii)* provides a means by which domain decomposition converges at a rate independent of grid spacing, and it works in a manner rather analogous to adding levels in a FMG method as grid spacing is refined. Point *iii)* is especially important, for it implies that on multiprocessors the simpler additive Schwarz methods will generally be at least as effective as are the multiplicative Schwarz procedures. Indeed, the latter cannot be parallelized without "coloring" the subdomains. In the absence of coloring, a two-subdomain decomposition can be calculated in roughly equal times by the two approaches; moreover, if $p > 2$ the additive Schwarz methods will be more efficient provided there are more than two processors available. Finally, point *iv)* should also be noted. Clearly, some overlap is extremely important, but it has been found that generally no more than two or three grid lines are needed for significant improvements in convergence rates, and beyond this further increases in overlap are generally not useful.

2.8.4 Multilevel domain-decomposition methods

We conclude this introduction to domain-decomposition methods with a brief discussion of so-called *multilevel domain decomposition* which actually corresponds to a combination of multigrid and domain-decomposition methods, as will be evident from what follows. The motivation for this approach is the need to remove low-wavenumber errors, as is done in multigrid methods. But the interpretation of this problem is now somewhat different. In particular, we have already seen that domain decompositions reduce the spectral radius of the iteration matrices used in each subdomain in a way that is complementary to what multigrid methods accomplish. On the other hand, because computations are never done on a coarse grid in the usual single-level domain decompositions, boundary information from the actual problem geometry propagates to the interior of the domain somewhat slowly. This can be improved with the use of a coarse grid in conjunction with the fine grid on which the accurate calculations are performed.

We begin by noting that the preconditioning matrices B_i introduced earlier have analogues in multigrid procedures. To see this, recall that the multigrid coarse-grid correction is carried out via the following steps: *i*) restriction of the fine-grid residual, *ii*) coarse-grid solution of the defect equation, and *iii*) prolongation of the coarse-grid error back to the fine grid. We also noted that typical restriction operators are transposes of a corresponding prolongation. Thus,

$$B_C = \mathcal{R}^T A_C^{-1} \mathcal{R}$$

(where $A_C = \mathcal{R} A \mathcal{R}^T$) provides a preconditioner embodying all the components of a coarse-grid correction. Clearly, this is of the same general form as the domain-decomposition preconditioner discussed in the previous section if we appropriately interpret A_C^{-1} and at the same time recognize that the details of restriction (and prolongation) are rather different in the multigrid case.

With these ideas in mind, we can define a two-level method as

$$u_F^{(n+1/2)} = u_F^{(n)} + B_C \left(f - A_F u_F^{(n)} \right) , \tag{2.163a}$$

$$u_F^{(n+1)} = u_F^{(n+1/2)} + B_F \left(f - A_F u_F^{(n+1/2)} \right) . \tag{2.163b}$$

In these expressions the subscript F corresponds to "fine grid," and C represents "coarse grid." Our two-grid multigrid method discussed in the preceding section was, in fact, of precisely this form, but there the preconditioning matrices were defined so as to apply over the entire domain, rather than over subdomains as in the present case. Here we leave open the possibility of constructing either, or both, of B_C and B_F (but especially the latter) separately on individual subdomains within each of the above steps. Furthermore, we note that the formalism of an additive Schwarz procedure can be employed to parallelize the above two-level method.

We present Fig. 2.25 to help clarify some of these ideas. In this figure the grid points indicate the coarse

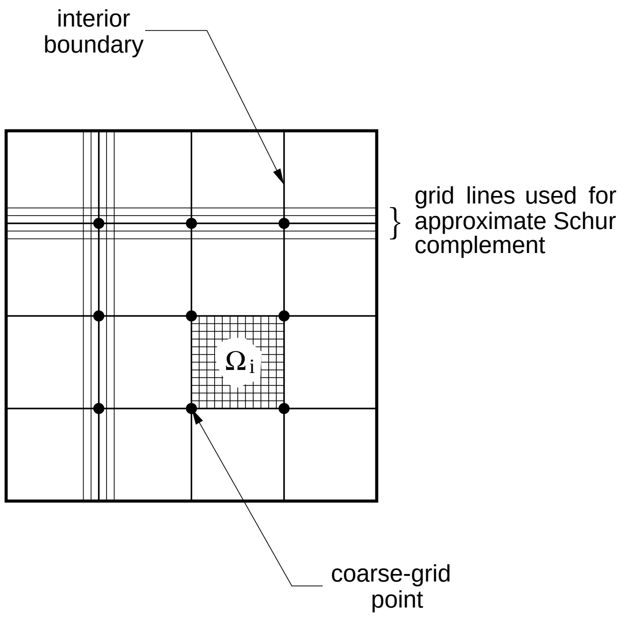

Figure 2.25: Schematic depicting two-level domain decomposition and approximate Schur complement.

grid on which B_C will act, and the fine grid is shown in detail only within the subdomain Ω_i although it is employed throughout.

Now suppose an estimate of $u_F^{(n)}$ has been calculated on the fine grid by some means, say by either multiplicative or additive Schwarz iterations. We then apply Eq. (2.163a) to obtain a coarse-grid correction, thus reducing low-wavenumber errors or, more specifically, more quickly moving boundary condition information into the interior of the domain. These results are then interpolated back to the fine grid to produce $u_F^{(n+1/2)}$. At this point we apply the second step of the algorithm. The preconditioner B_F will be of multiplicative or additive Schwarz type, and particularly in the latter case would be applied to all Ω_is in parallel. Only a few global iterations of this sort are needed, independent of the fine-grid discretization step size.

Schueller and McDonough [44] have recently proposed a variant of these multilevel domain decompositions that includes two additional aspects. One is analogous to a global fine-grid smoothing, but is intended mainly to aid in globally coupling the subdomain solutions, and the second is an approximate Schur complement calculation that provides much more accurate boundary data for the interior boundaries. Otherwise, the structure of the algorithm is quite similar to what we have already discussed.

Just as in a typical multigrid method, the fine-grid smoothing is performed as the first step of the current iteration. Following this, the approximate Schur complements are constructed for all interior boundaries by applying SLOR on each such boundary and a few of the nearest-neighbor grid lines, as indicated in Fig. 2.25. Clearly, all such solves can be performed in parallel in each separate direction (there is a parallelization issue with vertex points if all lines in both directions are done simultaneously). Once these boundary values have been computed, all subdomains can be computed in parallel if the additive Schwarz procedure is employed.

These two added steps require minimal arithmetic per pass through the complete algorithm. The fine-grid smoothing is done for only a few iterations, and calculation of the approximate Shur complements requires less than one fine-grid smoothing operation for all boundaries combined. Moreover, the latter provides additional overlap even though the domain decomposition, *per se*, has only one common line of data for every pair of adjacent subdomains.

Preliminary results [44] for a model Poisson–Dirichlet problem have been very encouraging, indicating that total arithmetic scales as $\mathcal{O}(N)$ even for extremely large problems (up to $> 2.5\times 10^6$ grid points). But in addition, the overall parallelizability of the method is such as to yield run times that are independent of problem size provided there are sufficient processors available to permit linear increase in number of processors with the number of subdomains.

2.9 Summary

In this chapter we have presented a wide range of algorithms for solving elliptic partial differential equations. We began with the basic theory of linear fixed-point iteration, and applied this to analysis of SOR by relating SOR to Jacobi iteration, an easily analyzed case. We considered a number of modifications to basic point SOR, including line SOR which is often implemented as an ADI procedure. We also thoroughly analyzed ADI, the result of which showed that SLOR is usually more efficient.

We continued with study of incomplete LU decompositions which now are used mainly as preconditioners, and seldom as stand-alone algorithms as had once been proposed. In addition to preconditioners we also briefly treated acceleration methods such as the conjugate-gradient algorithm. This method in various forms, as well as other Krylov space-based techniques such as GMRES, have also been widely proposed as stand-alone solvers, but they are better employed as acceleration methods applied with other approaches.

Our treatment of multigrid methods was fairly basic and abbreviated. Nevertheless, sufficient detail was presented to permit construction of basic computational algorithms. The same is true of our presentation of domain-decomposition methods. The most important point to be made for either of these approaches is that they are the only techniques that can come close to solving the sparse linear systems that arise in discretizations of elliptic PDEs in only $\mathcal{O}(N)$ arithmetic operations. Indeed, combination of these two methods provides an algorithm that not only requires only $\mathcal{O}(N)$ total arithmetic but is also highly parallelizable, and thus is optimal.

Chapter 3

Time-Splitting Methods for Evolution Equations

In this chapter we will treat the numerical solution of time-dependent partial differential equations, mainly of parabolic type (but often employing techniques that apply equally well to hyperbolic equations). As was the case in Chap. 2, we will consider problems consisting of only a single, linear equation, and posed on rectangular domains of two and three spatial dimensions.

We will concentrate our efforts on so-called *time-splitting* methods for solving such problems, in the context of finite-difference (or finite-volume) discretizations. This is by far the most efficient combination for general (having not necessarily smooth solutions) problems. Time-splitting procedures are often termed "method of fractional steps," especially in the Russian literature, and they are frequently referred to simply as "ADI methods." In the present lectures we will identify three distinct classes of time-splitting techniques based, not necessarily on their final form (which tends to almost always be nearly the same), but rather on the manner in which they will have been constructed. These are:

i) alternating direction implicit (ADI),

ii) locally one-dimensional (LOD), and

iii) general Douglas–Gunn (D–G).

A section of this chapter will be devoted to treatment of each of these.

We will consider as a basic model problem for all methods, the 2-D transient heat equation on a rectangle:

$$u_t = \Delta u + f(x,y,t) \qquad (x,y) \in \Omega \equiv (0,a) \times (0,b)\,, \quad t \in (0,t_f]\,, \tag{3.1a}$$

with initial data

$$u(x,y,0) = u_0(x,y)\,, \tag{3.1b}$$

and boundary conditions

$$u(x,y,t) = g(x,y,t), \qquad (x,y) \in \partial\Omega\,. \tag{3.1c}$$

This problem is particularly simple, and exact solutions are readily constructed. It thus provides a good reference point by means of which performance of the various classes of methods can be compared.

We emphasize at the outset that the reason for the efficiency of time-splitting is that multi-dimensional problems are decomposed into sequences of 1-D problems via this approach. As we will see below, this results in only $\mathcal{O}(N)$ arithmetic operations per time step, and this cannot be easily matched with any iterative method, as is clear from the preceding chapter. Explicit methods are the only alternatives to exhibit this feature, but for multi-dimensional problems these require (usually) unacceptably small time steps. These 1-D problems are, of course, easily solved (and easily parallelized, at least in principle). But the requirement for accomplishing time splitting is that the system matrix corresponding to the complete multi-dimensional system be sparse and banded, and with very regular band structure. It is this requirement that leads to difficulty for finite-element methods, and even for finite-difference methods constructed on unstructured grids.

3.1 Alternating Direction Implicit Methods

We have already discussed ADI methods for solving elliptic problems in Chap. 2 of these lectures, but the treatment there (as we had noted at that time) was rather different from the time-dependent analysis we will now study. We note that there are three main steps in constructing an ADI method in this context:

i) discretization of the partial differential equation(s),

ii) factorization of the discrete equation(s), and

iii) splitting of the factored equation(s).

It is important to recognize that error will be introduced with each one of these steps, and in order to assess the overall accuracy of any given ADI scheme it will be necessary to analyze each such error contribution. We will carry this out for the model problem in the context of the Peaceman–Rachford method [22], and then provide a simple analysis of stability for this approach. We will then briefly consider a second widely-used ADI scheme, and then conclude the section with some implementation details.

3.1.1 Peaceman–Rachford ADI

In this section we will derive the Peaceman–Rachford alternating direction implicit scheme for 2-D parabolic equations, and in the course of doing this analyze error contributions from each of the individual steps. We emphasize that this method can be applied only to 2-D problems, but we present it here because it is very well known and it provides a good tool for the beginning study of ADI construction processes in general.

Accuracy

The *discretization* we employ here is the widely-used Crank–Nicolson method (see, *e.g.*, Mitchell and Griffiths [16]) which in 2D takes the form

$$\left[I - \frac{k}{2}\left(D_{0,x}^2 + D_{0,y}^2\right)\right] u_{\ell,m}^{n+1} = \left[I + \frac{k}{2}\left(D_{0,x}^2 + D_{0,y}^2\right)\right] u_{\ell,m}^{n} + \frac{k}{2}\left(f_{\ell,m}^{n+1} + f_{\ell,m}^{n}\right) . \tag{3.2}$$

This difference equation holds for all $\ell = 2, 3, \ldots, N_x - 1$ and $m = 2, 3, \ldots, N_y - 1$; the boundary point values corresponding to the indices $\ell = 1, N_x$ and $m = 1, N_y$ are prescribed as indicated in Eq. (3.1c). Also note that in these lectures the notation $D_{0,\cdot}$ will be used to indicate a second-order accurate centered-difference approximation, and k denotes the time step value, *i.e.*, Δt ($\equiv t^{n+1} - t^n$). This approximation is well-known to be second order in both space and time, as may be easily checked via a formal truncation error analysis, which we leave to the reader.

To simplify notation, we set $A \equiv D_{0,x}^2$ and $B \equiv D_{0,y}^2$ and write

$$\left[I - \frac{k}{2}(A+B)\right] u^{n+1} = \left[I + \frac{k}{2}(A+B)\right] u^{n} + \frac{k}{2}(f^{n+1} + f^n) , \tag{3.3}$$

where we have now suppressed spatial grid-point indices in favor of the more compact matrix-vector representations. We remark that structure of the matrix on the left-hand side of the above equation is that of a discrete Laplacian. Hence, solution of the system is expensive—and it must be done many times in the case of time-dependent problems. We have seen in the preceding chapter that rather complicated algorithms are needed to obtain solutions in only $\mathcal{O}(N)$ arithmetic; moreover, the actual total amount of arithmetic still is fairly large. This provides the motivation for techniques we consider in this chapter.

Our task now is to *factor* the matrices on both sides of Eq. (3.3). This can be done by observing that

$$\left(I - \frac{k}{2}A\right)\left(I - \frac{k}{2}B\right) = I - \frac{k}{2}(A+B) + \frac{k^2}{4}AB\,.$$

Thus, the factorization on the left-hand side agrees with the matrix on the left-hand side of (3.3) up to terms of $\mathcal{O}(k^2)$. A similar result holds for the right-hand-side matrix of (3.3), and substitution of these back into Eq. (3.3) yields

$$\left[\left(I - \frac{k}{2}A\right)\left(I - \frac{k}{2}B\right) - \frac{k^2}{4}AB\right]u^{n+1} = \left[\left(I + \frac{k}{2}A\right)\left(I + \frac{k}{2}B\right) - \frac{k^2}{4}AB\right]u^n + \frac{k}{2}\left(f^{n+1} + f^n\right)\,,$$

or

$$\left(I - \frac{k}{2}A\right)\left(I - \frac{k}{2}B\right)u^{n+1} = \left(I + \frac{k}{2}A\right)\left(I + \frac{k}{2}B\right)u^n + \frac{k}{2}\left(f^{n+1} + f^n\right) + \frac{k^2}{4}AB\left(u^{n+1} - u^n\right)\,. \qquad (3.4)$$

Now recall that the original unsplit Crank–Nicolson scheme has $\mathcal{O}(k^3)$ local temporal error, so at first it appears that factorization has resulted in a reduction of order of accuracy. But if $u \in C^1$, the mean value theorem implies that

$$u^{n+1} - u^n = u_t(\xi)\left(t^{n+1} - t^n\right) \quad \text{for some} \quad \xi \in \left(t^n, t^{n+1}\right)\,,$$

from which it follows that $u^{n+1} - u^n \sim \mathcal{O}(k)$. Hence, no formal accuracy has been lost in the factorization given by (3.4). But we must also note that we have not yet gained any simplification of the solution procedure.

To accomplish this simplification we must now carry out the *splitting* step. There are numerous ways by which this can be done; here we will employ the Peaceman–Rachford (P–R) method [22], and write (3.4) as the system of equations

$$\left(I - \frac{k}{2}A\right)u^{n+1^*} = \left(I + \frac{k}{2}B\right)u^n + \frac{k}{2}f^n\,, \qquad (3.5a)$$

$$\left(I - \frac{k}{2}B\right)u^{n+1} = \left(I + \frac{k}{2}A\right)u^{n+1^*} + \frac{k}{2}f^{n+1}\,, \qquad (3.5b)$$

where u^{n+1^*} corresponds to an intermediate time t^*, $t^n \le t^* \le t^{n+1}$. In many presentations of the P–R method it is assumed that $t^* = t^{n+1/2}$; this is not always correct in general, despite the fact that it is necessary for maintaining consistency of the individual split equations. It is important to note at this time that both Eqs. (3.5a) and (3.5b) correspond to tridiagonal systems, and thus can be solved very efficiently. Hence, we have produced the desired simplification. But we must yet determine the error caused by splitting.

To analyze the formal accuracy of P–R ADI we will use Eqs. (3.5) in an attempt to recover the factored form of the Crank–Nicolson scheme (because we already know the accuracy of this method), and note the difference between these two results. We begin by solving Eq. (3.5a) for u^{n+1^*} and substituting the result into Eq. (3.5b) to obtain

$$\begin{aligned}\left(I - \frac{k}{2}B\right)u^{n+1} &= \left(I + \frac{k}{2}A\right)\left[\left(I - \frac{k}{2}A\right)^{-1}\left(I + \frac{k}{2}B\right)u^n + \frac{k}{2}\left(I - \frac{k}{2}A\right)^{-1}f^n\right] + \frac{k}{2}f^{n+1} \\ &= \left(I + \frac{k}{2}A\right)\left(I - \frac{k}{2}A\right)^{-1}\left[\left(I + \frac{k}{2}B\right)u^n + \frac{k}{2}f^n\right] + \frac{k}{2}f^{n+1}\,. \qquad (3.6)\end{aligned}$$

At this point we should recognize that if we are to recover anything that resembles (3.4) we need to multiply (3.6) by $\left(I - \frac{k}{2}A\right)$; this leads to

$$\left(I - \frac{k}{2}A\right)\left(I - \frac{k}{2}B\right)u^{n+1} = \left(I - \frac{k}{2}A\right)\left(I + \frac{k}{2}A\right)\left(I - \frac{k}{2}A\right)^{-1}\left[\left(I + \frac{k}{2}B\right)u^n + \frac{k}{2}f^n\right] + \frac{k}{2}\left(I - \frac{k}{2}A\right)f^{n+1} . \tag{3.7}$$

We now observe that $I - \frac{k}{2}A$ and $I + \frac{k}{2}A$ commute (provided A is independent of time), so (3.7) becomes

$$\begin{aligned}\left(I - \frac{k}{2}A\right)\left(I - \frac{k}{2}B\right)u^{n+1} &= \left(I + \frac{k}{2}A\right)\left(I + \frac{k}{2}B\right)u^n + \frac{k}{2}\left(I + \frac{k}{2}A\right)f^n + \frac{k}{2}\left(I - \frac{k}{2}A\right)f^{n+1} \\ &= \left(I + \frac{k}{2}A\right)\left(I + \frac{k}{2}B\right)u^n + \frac{k}{2}\left(f^{n+1} + f^n\right) - \frac{k^2}{4}A\left(f^{n+1} - f^n\right) .\end{aligned} \tag{3.8}$$

From this we see that if A is independent of time, and $f \in C^1(t_0, t_f)$, so that

$$f^{n+1} - f^n = f_t(\xi)\left(t^{n+1} - t^n\right) \sim \mathcal{O}(k) ,$$

then (3.8) is equivalent to (3.4) up to terms of $\mathcal{O}(k^3)$.

Since this is the order of the local truncation error of the unsplit Crank–Nicolson scheme, we have shown that the formal accuracy of P–R ADI is equivalent to that of Crank–Nicolson for the 2-D heat equation with C^1 source term, despite the much higher solution efficiency gained from the time splitting.

Stability

At this time we briefly analyze stability of time-split schemes, but mainly in the context of the P–R method just presented. It is useful to note that all of the time-split methods we will consider herein are implicit (although explicit time-splitting is also widely applied), so stability (at least for linear problems) is typically not a problem. An important property of any time-split method is that individual steps need be neither consistent nor stable in order for the overall scheme to exhibit these properties. In some situations this can be a significant advantage.

For the P–R method being considered here, each of the steps is basically a 1-D Crank–Nicolson scheme, which for linear heat equations is well known to be unconditionally stable with respect to a von Neumann stability analysis (see, *e.g.*, [16]). This implies that the amplification factor for each individual step is less than unity, and we then simply need to show that the amplification factor for the complete method is the product of the amplification factors of the individual steps.

Consider Eq. (3.5) with $A = B$. We note that this occurs for the heat equation if the same spatial step size is used in both directions. In terms of a *matrix stability analysis* it is required that the spectral radius of

$$C \equiv \left(I - \frac{k}{2}A\right)^{-1}\left(I + \frac{k}{2}B\right)$$

be less than or equal to unity; *i.e.*, $\rho(C) \le 1$, for the first split step to be stable. Similarly, we require $\rho(D) \le 1$, with

$$D \equiv \left(I - \frac{k}{2}B\right)^{-1}\left(I + \frac{k}{2}A\right) ,$$

for the second step. Now recall that for any matrix M, $\rho(M) \le \|M\|$ for all matrix norms $\|\cdot\|$. Also recall that if we define

$$z^n \equiv v^n - u^n ,$$

where v^n is the exact solution to the difference equations, and u^n is the computed one, we have

$$z^{n+1} = \left(I - \frac{k}{2}B\right)^{-1}\left(I + \frac{k}{2}A\right)\left(I - \frac{k}{2}A\right)^{-1}\left(I + \frac{k}{2}B\right)z^n \,. \tag{3.9}$$

Thus,

$$\|z^{n+1}\| = \|DCz^n\| \le \|D\|\|C\|\|z^n\| \,.$$

If we now take $\|\cdot\|$ to be the spectral norm for the matrices (and the 2-norm for vectors) so that $\|\cdot\| = \rho(\cdot)$, which is valid provided C and D are diagonalizable with real eigenvalues, we have

$$\|z^{n+1}\| \le \rho(C)\cdot\rho(D)\|z^n\| \,.$$

But since C and D are equivalent to matrices arising in the 1-D Crank–Nicolson method, we have $\rho(C) \le 1$ and $\rho(D) \le 1$ for all values of k and h. Hence,

$$\|z^{n+1}\| \le \|z^n\| \,,$$

and we have demonstrated (unconditional) stability for the P–R method.

It should be clear from this development that since all that really is needed is $\rho(C)\cdot\rho(D) \le 1$, it is possible for one or the other (but not both) individual split steps to be unstable while still maintaining overall stability of the complete method.

3.1.2 Douglas–Rachford ADI

In this section we will briefly treat the Douglas–Rachford [45] ADI scheme. This method is based on a backward-Euler time integration, and so is only first-order accurate in time. But it is unconditionally stable just as was the P–R method, and unlike that method, the Douglas–Rachford (D–R) scheme is easily generalized to three space dimensions. Furthermore, it is worth noting that in some situations, maintaining high-order accuracy is difficult for boundary condition implementations, so use of a first-order method may sometimes be appropriate in any case.

The discretization of the heat equation using backward-Euler integration in time and centered differencing in space results in

$$\left[I - k\left(D_{0,x}^2 + D_{0,y}^2\right)\right]u_{\ell,m}^{n+1} = u_{\ell,m}^n + kf_{\ell,m}^{n+1} \,, \tag{3.10}$$

for all $\ell, m = 2, 3, \dots, N_x - 1, N_y - 1$, or in our abbreviated notation,

$$\left[I - k\,(A + B)\right]u^{n+1} = u^n + kf^{n+1} \,.$$

The factored form of this is

$$(I - kA)(I - kB)u^{n+1} = u^n + f^{n+1} \,, \tag{3.11}$$

and the Douglas–Rachford splitting results in

$$(I - kA)u^{n+1^*} = (I + kB)u^n + kf^{n+1} \tag{3.12a}$$

$$(I - kB)u^{n+1} = u^{n+1^*} - kBu^n \,. \tag{3.12b}$$

We leave derivation of the factorization and splitting errors for this method as an exercise for the interested reader.

As we noted above with regard to the Peaceman–Rachford splitting, there are many possible ways in which this might be carried out once the equation has been factored. But clearly, one should always try to produce a splitting which when re-composed will be as close as possible to the factored but unsplit formulation—or, better yet, as close as possible to the original unfactored discrete form. It is readily seen by formally solving (3.12a) for u^{n+1^*} and substituting this into (3.12b), followed by multiplication by $(I - kA)$ that this has been accomplished by the Douglas–Rachford splitting. It will be seen in a later section that this splitting is the same as that employed in the Douglas–Gunn procedures.

We will not present details of a stability analysis for this method. Since it is based on backward-Euler integration in time, it is expected to be unconditionally stable for linear problems; and an analysis similar to that used above for the P–R method will confirm this for the split scheme. (We encourage the reader to carry out this analysis.) Moreover, we note that based on the preceding analysis, we can conclude that stability of the split scheme is usually at least as good as for the unsplit one. In fact, explicit schemes are sometimes split specifically to exploit this property of time splitting.

We will close this section by presenting the equations for the D–R procedure applied to a 3-D heat equation in the homogeneous case:

$$\left(I - kD_{0,x}^2\right) u_{j,\ell,m}^{n+1^*} = \left[I + k\left(D_{0,y}^2 + D_{0,z}^2\right)\right] u_{j,\ell,m}^n\,, \tag{3.13a}$$

$$\left(I - kD_{0,y}^2\right) u_{j,\ell,m}^{n+1^{**}} = u_{j,\ell,m}^{n+1^*} - kD_{0,y}^2 u_{j,\ell,m}^n\,, \tag{3.13b}$$

$$\left(I - kD_{0,z}^2\right) u_{j,\ell,m}^{n+1} = u_{j,\ell,m}^{n+1^{**}} - kD_{0,z}^2 u_{j,\ell,m}^n\,, \tag{3.13c}$$

for all $j, \ell, m = 2, 3, \ldots, N_x - 1, N_y - 1, N_z - 1$. We leave as an exercise for the interested reader derivation of corresponding formulas in the nonhomogeneous case, as well as analysis of the splitting error for this scheme.

3.1.3 Implementation of ADI schemes

In this section we will discuss some of the details that must be understood to successfully implement ADI methods. In fact, most of what we will present here carries over to all time-splitting techniques. There are two main items that are of particular importance; these are: *i*) specific details of carrying out the line-by-line solves and *ii*) treatment of boundary conditions.

ADI line-by-line solves

One of the advantages of ADI methods from their earliest use was the ability to decompose the global solution domain into a sequence of lines, thus permitting solution of the complete PDE problem as a collection of solutions to problems that were each no larger than ODE two-point boundary-value problems on the same line of grid points. In the modern era of extremely large CPUs and memories, this is a much less important factor. On the other hand, in instances of small cache sizes, and/or in the context of parallelization, this feature can still present an advantage over other approaches in addition to gains in efficiency in numerical linear algebra, as we have already indicated. In this subsection we will describe the details of implementing these line-by-line solves, and also briefly describe an alternative.

It is of interest to consider the form of the discrete equation on any arbitrary line of the grid to gain an understanding of some subtle aspects of correct implementation. We will consider the P–R scheme here, although similar requirements arise for most other methods, an exception being the LOD techniques described in the next section. We choose an arbitrary line from the first split step, corresponding to Eq. (3.5a), and express this as

$$-ru_{\ell-1,m}^{n+1^*} + (1 + 2r)u_{\ell,m}^{n+1^*} - ru_{\ell+1,m}^{n+1^*} = ru_{\ell,m-1}^n + (1 - 2r)u_{\ell,m}^n + ru_{\ell,m+1}^n + \frac{k}{2}f_{\ell,m}^n\,, \tag{3.14}$$

where $r = k/2h^2$. An equation of this form holds for each $m = 2, 3, \ldots, N_y{-}1$ and, with some modification for boundary conditions, also for $m = 1$ and $m = N_y$.

Since the right-hand side is known, the problem is completely analogous to that of solving an ODE boundary-value problem along the m^{th} grid line. But the fact that everything on the right-hand side is to be evaluated with time-level n information presents a slight difficulty in terms of storing newly-computed results. The problem is indicated graphically in Fig. 3.1, and we describe it here. The important thing to observe is that results on grid line $m{-}1$ at time level n are still needed for the $n{+}1$ time-level calculation at

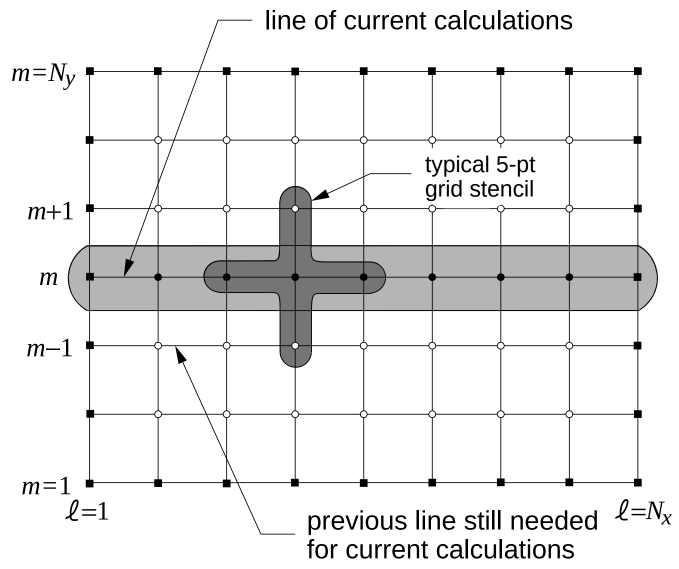
line of current calculations
$m=N_y$
typical 5-pt grid stencil
$m+1$
m
$m-1$
$m=1$
$\ell=1$
previous line still needed for current calculations
$\ell=N_x$

Figure 3.1: Implementation of line-by-line solves for time splitting of time-dependent problems.

grid line m. But these would typically have been overwritten by $n+1$ time-level results to reduce storage requirements. It should be clear from Eq. (3.14) that it is necessary to employ the correct time level of data. There are two main implementation approaches to dealing with this (other than always storing two complete time levels of data—which might be acceptable on modern computers, and for a single PDE). The first is to utilize a temporary array to store the time-level n results on line $m-1$ before they are overwritten with the new time-level $n+1$ calculations. Then these are used in Eq. (3.14) at the appropriate place. If the number of points per grid line is quite large, as can easily be the case in 2-D problems, this is nearly always the preferred approach. But we observe that the first approach, retaining the n time level results in storage, is required for the D–R scheme given in Eq. (3.13) in any case.

The second possibility is rather different, and depending on specific machine architectures and problem sizes may, or may not, be of use. It involves recognizing that the theoretical time-split formulas in matrix form do not of themselves imply line-by-line solves. These were introduced from the beginning to reduce required computer memory. Thus, for moderate-size problems on modern supercomputers, it may be advisable to simultaneously solve all equations corresponding to a given direction. This still involves a tridiagonal matrix, and thus the same solution procedure that would be used for the line-by-line solves, but now the solution vector contains entries for every grid point of the entire grid. The advantages are that in this manner of calculating one need not be concerned whether a previous line has already been updated, and at the same time the matrix sizes are typically large enough to keep CPUs busy. On the other hand, it is now necessary to keep two complete copies of the solution vector. Moreover, if the CPU being used has a small cache, many calls to memory will be required, thus degrading computational efficiency.

Boundary conditions for ADI methods

Implementation of boundary conditions is often viewed as creating difficulties for ADI methods, and time-split methods in general. Some of the more subtle problems are associated with an inability to set a precise time for evaluation of boundary conditions used during the intermediate split step(s). This is discussed in some detail in [16] and [42], and considerable effort was expended on this perceived difficulty during the 1970s and 80s. We will not repeat these treatments here largely due to the fact that, as will be evident in later sections, some forms of time-split methods handle this in a completely natural and implementationally-easy manner.

A more serious problem is treatment of Neumann boundary conditions. It is well known that the analytical Dirichlet–Neumann problem is well posed provided the boundary of the solution domain contains at least one point at which a Dirichlet condition is imposed. On the other hand, for a numerical problem such as depicted in Fig. 3.2 being solved via a time-splitting procedure, the line solves being done in the direction having Neumann conditions on both ends are formally ill posed; that is, the matrices arising from a typical numerical implementation will be singular.

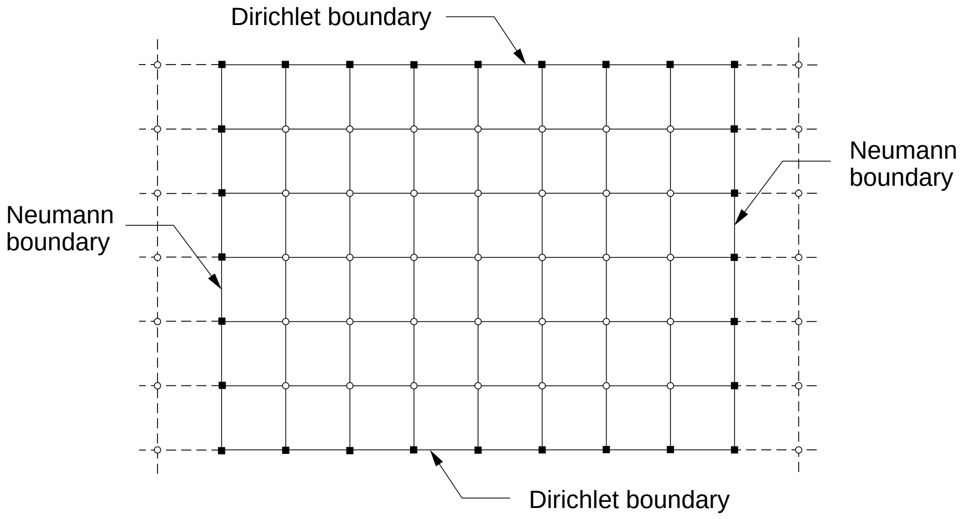

Figure 3.2: Numerical Dirichlet–Neumann problem; points on dashed lines are "image" points needed for implementation of centered discretizations.

It should first be pointed out that despite the formal singularity encountered, there is often sufficient rounding and truncation error in practice to prevent singularity, so the solution procedure proceeds without undue difficulty. In the event that this is not the case, there are steps that can be taken to avoid the singularity. It must be recognized that the system matrix as a whole is not singular; the singularities in the individual line solves arise specifically from the splitting procedure. So one remedy would be to avoid the time splitting approach. But this would cause significant increase in required arithmetic to obtain a problem solution, as we have already emphasized.

Another approach is to solve first in the Dirichlet condition direction; *i.e.*, in that direction for which there is a Dirichlet condition on at least one end. This basically establishes the value of the constant that is arbitrary in the context of Neumann boundary conditions. While this is easily implemented in the context of a problem such as displayed in the figure, it is less straightforward when, *e.g.*, only a single Dirichlet point is present in the entire domain. In such cases, an alternative that still retains the efficiency of time splitting is to perform all line solves in a given direction for the entire grid, as discussed in the preceding subsection. Aside from the implementational disadvantages noted there, this is probably the best approach because it is completely general and permits any assortment of Dirichlet and Neumann boundary points. In particular, the numerical problem during each split step closely mimics the analytical problem and thus retains well posedness.

3.2 Locally One-Dimensional Methods

Locally one-dimensional (LOD) methods were first introduced by numerical analysts of the former Soviet Union and are treated in detail by Yanenko [46] with the often-used terminology "method of fractional steps." A fairly complete, but condensed, treatment is provided by Mitchell and Griffiths [16], and we will to some extent follow that herein. Construction of LOD methods differs from what we have just studied for ADI techniques in that the differential equations are split first (usually, but not always, directionally), and the result is then discretized by standard methods. Hence, there are two main steps required for analysis and implementation of these techniques:

i) splitting of the <u>differential</u> equation,

ii) discretization of resulting 1-D equations.

This eliminates the factorization step altogether, so it is a simpler procedure that is more readily analyzed. Moreover, it is typically also somewhat easier to implement because each split step is often completely one dimensional. (Recall that in the P–R ADI procedure, the individual solves are 1D, but the right-hand side of the equations contains information from the opposite direction.) On the other hand, because the differential operators have been split, it is difficult to maintain the same order of temporal accuracy with LOD methods as achieved by the unsplit scheme unless the original procedure is only first-order accurate in time.

To demonstrate the LOD approach, we will again consider Eq. (3.1a):

$$u_t = \Delta u + f(x,y,t)\,.$$

As noted above, we begin by splitting the differential equation. In LOD methods this is done along spatial directions, and in the present case it results in

$$\frac{1}{2}u_t = u_{xx} + \frac{1}{2}f(x,y,t)\,, \tag{3.15a}$$

$$\frac{1}{2}u_t = u_{yy} + \frac{1}{2}f(x,y,t)\,. \tag{3.15b}$$

Observe that the sum of these two equations is the original PDE, and this immediately suggests the approach to be applied in extensions to 3D.

We can now apply the Crank–Nicolson method (or any other desired technique) to each equation separately. It is standard in 2D to perform each such discretization over a one-half time step; thus the discrete formulas are

$$\left(I - \frac{k}{2}D^2_{0,x}\right)u^{n+1/2}_{\ell,m} = \left(I + \frac{k}{2}D^2_{0,x}\right)u^n_{\ell,m} + \frac{k}{4}\left(f^{n+1/2}_{\ell,m} + f^n_{\ell,m}\right)\,, \tag{3.16a}$$

$$\left(I - \frac{k}{2}D^2_{0,y}\right)u^{n+1}_{\ell,m} = \left(I + \frac{k}{2}D^2_{0,y}\right)u^{n+1/2}_{\ell,m} + \frac{k}{4}\left(f^{n+1}_{\ell,m} + f^{n+1/2}_{\ell,m}\right)\,, \tag{3.16b}$$

for $\ell, m = 2, 3, \ldots, N_x{-}1, N_y{-}1$. It is important to remark here that neither (3.16a) nor (3.16b) is consistent with the original PDE. Nevertheless, as we will demonstrate in what follows, when the equations are used as indicated, the result is consistent, and first-order accurate in time.

As usual, we write these in the simplified matrix-vector notation

$$\left(I - \frac{k}{2}A\right)u^{n+1/2} = \left(I + \frac{k}{2}A\right)u^n + \frac{k}{4}\left(f^{n+1/2} + f^n\right)\,, \tag{3.17a}$$

$$\left(I - \frac{k}{2}B\right)u^{n+1} = \left(I + \frac{k}{2}B\right)u^{n+1/2} + \frac{k}{4}\left(f^{n+1} + f^{n+1/2}\right)\,, \tag{3.17b}$$

and then attempt to recover the unsplit discrete equations, which in the present case would be the factored Crank–Nicolson method applied to the 2-D heat equation, namely Eq. (3.2). We begin by solving Eq. (3.17a) for $u^{n+1/2}$ and substituting this result into Eq. (3.17b). After multiplication by $\left(I - \frac{k}{2}A\right)$, this becomes

$$\left(I - \frac{k}{2}A\right)\left(I - \frac{k}{2}B\right)u^{n+1} = \left(I - \frac{k}{2}A\right)\left(I + \frac{k}{2}B\right)\left(I - \frac{k}{2}A\right)^{-1}\left[\left(I + \frac{k}{2}A\right)u^n + \frac{k}{4}\left(f^{n+1/2} + f^n\right)\right]$$

$$+\frac{k}{4}\left(I-\frac{k}{2}A\right)\left(f^{n+1}+f^{n+1/2}\right). \tag{3.18}$$

It is easy to see that if A and B commute we can write this as

$$\begin{aligned}\left(I-\frac{k}{2}A\right)\left(I-\frac{k}{2}B\right)u^{n+1} &= \left(I+\frac{k}{2}A\right)\left(I+\frac{k}{2}B\right)u^n+\frac{k}{4}\left(I+\frac{k}{2}B\right)\left(f^{n+1/2}+f^n\right)\\ &\qquad +\frac{k}{4}\left(I-\frac{k}{2}A\right)\left(f^{n+1}+f^{n+1/2}\right)\\ &= \left(I+\frac{k}{2}A\right)\left(I+\frac{k}{2}B\right)u^n+\frac{k}{4}\left(f^{n+1}+2f^{n+1/2}+f^n\right)\\ &\qquad +\frac{k^2}{8}\left[B\left(f^{n+1/2}+f^n\right)-A\left(f^{n+1}+f^{n+1/2}\right)\right]\\ &= \left(I+\frac{k}{2}A\right)\left(I+\frac{k}{2}B\right)u^n+\frac{k}{2}\left(f^{n+1}+f^n\right)\\ &\qquad +\frac{k^2}{8}\left[B\left(f^{n+1/2}+f^n\right)-A\left(f^{n+1}+f^{n+1/2}\right)\right]+\mathcal{O}(k^3).\end{aligned}$$

Note that the $\mathcal{O}(k^3)$ term arises from what is essentially an $\mathcal{O}(k^2)$-accurate filtering of $f^{n+1/2}$ in the second step, above, and viewing this as arising from midpoint integration of the result. We leave proof of this as an exercise for the reader. In particular, this is within $\mathcal{O}(k^3)$ of the trapezoidal integration shown in the last step, and which is needed to coincide with the (factored) Crank–Nicolson procedure of the unsplit scheme.

It is clear that even if A and B commute, unless $f \equiv 0$, the error due to splitting will be $\mathcal{O}(k^2)$ locally. We have not considered the equivalent factorization error—which from the form of the equations is obviously still present even though no formal factorization was employed—because if $u \in C^1$ this is an $\mathcal{O}(k^3)$ error just as was true for P–R ADI, and is thus higher order.

It is worthwhile to consider the case $f \equiv 0$, but for $AB \neq BA$ to examine the nature of the splitting error, and from this propose a way to remove it. In Eq. (3.18) we have

$$\left(I-\frac{k}{2}A\right)\left(I+\frac{k}{2}B\right)=I-\frac{k}{2}(A-B)-\frac{k^2}{4}AB,$$

and

$$\left(I+\frac{k}{2}B\right)\left(I-\frac{k}{2}A\right)=I-\frac{k}{2}(A-B)-\frac{k^2}{4}BA;$$

hence, subtracting the second of these from the first yields

$$\left(I-\frac{k}{2}A\right)\left(I+\frac{k}{2}B\right)=\left(I+\frac{k}{2}B\right)\left(I-\frac{k}{2}A\right)-\frac{k^2}{4}(AB-BA).$$

Thus, in the homogeneous case (3.18) can be written as

$$\left(I-\frac{k}{2}A\right)\left(I-\frac{k}{2}B\right)u^{n+1}=\left(I+\frac{k}{2}B\right)\left(I+\frac{k}{2}A\right)u^n-\frac{k^2}{4}(AB-BA)\left(I-\frac{k}{2}A\right)^{-1}\left(I+\frac{k}{2}A\right)u^n.$$

We also have, similar to relations found earlier,

$$\left(I+\frac{k}{2}B\right)\left(I+\frac{k}{2}A\right)=\left(I+\frac{k}{2}A\right)\left(I+\frac{k}{2}B\right)-\frac{k^2}{4}(AB-BA),$$

so the above becomes

$$\begin{aligned}\left(I-\frac{k}{2}A\right)\left(I-\frac{k}{2}B\right)u^{n+1} &= \left(I+\frac{k}{2}A\right)\left(I+\frac{k}{2}B\right)u^{n} \\ &\quad -\frac{k^2}{4}(AB-BA)\left[I+\left(I-\frac{k}{2}A\right)^{-1}\left(I+\frac{k}{2}A\right)\right]u^n, \end{aligned} \tag{3.19}$$

which clearly demonstrates the $\mathcal{O}(k^2)$ error in the absence of commutativity of A and B.

Finally, we note that

$$\left(I-\frac{k}{2}A\right)^{-1} = \left(I+\frac{k}{2}A\right)+\mathcal{O}(k^2)\,,$$

as is easily checked; thus, the $\mathcal{O}(k^2)$ term can be expressed as

$$\frac{k^2}{4}(AB-BA)\left[I+\left(I+\frac{k}{2}A\right)^2\right]u^n+\mathcal{O}(k^3)\,,$$

or simply

$$\frac{k^2}{2}(AB-BA)u^n+\mathcal{O}(k^3)\,.$$

We next observe that at least in 2D there is a straightforward way to recover full second-order accuracy with LOD methods. It is simply to perform each time step twice, switching the direction that is done first between the two calculations, and then averaging the results. In the homogeneous case it is easily checked that this results in cancellation of the *commutator* $AB-BA$. Moreover, on modern parallel computers the run times are essentially the same as for doing the time step once since the calculations are completely independent and can be done in parallel. In principle, such a treatment can be extended to nonhomogeneous problems, and to three space dimensions; but in either of these cases it is not so direct, and especially in the latter becomes rather inefficient. We leave further details of this sort to the reader, and to references such as [16].

There is one final observation we should make regarding LOD procedures. Unless first-order temporal accuracy is acceptable, they probably should not be used in the spatial time-splitting mode considered here. But these methods provide simple models of general *operator-splitting* methods that are in very wide use in many contexts, especially in CFD, and the analysis tools we have presented here apply in these cases as well. Indeed, one of the main approaches to solving the incompressible Navier–Stokes equations is a class of techniques known as *projection methods.* Although these differ considerably from one particular method to the next, they are all based on the notion of operator splitting implemented (conceptually) as follows.

We can write the momentum equations of incompressible flow as

$$\boldsymbol{U}_t+\boldsymbol{U}\cdot\nabla\boldsymbol{U} = -\nabla P+\frac{1}{Re}\Delta\boldsymbol{U}\,,$$

where $\boldsymbol{U}\equiv(u,v,w)^T$ is the velocity vector, and $P\equiv p/\rho$ is "kinematic" pressure; p is the usual hydrodynamic pressure, and ρ is (constant) density. Pressure-velocity coupling is a rather difficult aspect of solving these equations together with satisfaction of the divergence-free constraint $\nabla\cdot\boldsymbol{U}=0$. But this can be accomplished quite efficiently through an operator-splitting procedure analogous to what we have studied in this section. Namely, we first solve the momentum equations without the pressure gradient term, written as

$$\boldsymbol{U}_t+\boldsymbol{U}\cdot\nabla\boldsymbol{U} = \frac{1}{Re}\Delta\boldsymbol{U}\,.$$

This is relatively easy to accomplish because this system contains only mild nonlinearities (see Chap. 4 of these notes for efficient ways to treat these) and is otherwise just a weakly-coupled parabolic system. After this has been done, the equation

$$\boldsymbol{U}_t = -\nabla P$$

is solved. This is not usually carried out in the direct manner implied here, and we will not elaborate on the details. The main point is that an operator splitting has been employed to significantly simplify an otherwise fairly difficult problem. In CFD this is often termed a "fractional-step" method.

3.3 General Douglas–Gunn Procedures

We begin presentation of the Douglas–Gunn [47] time splitting methods in the context of the straightforward two-level scheme that is widely used for parabolic PDEs. We will provide a quite detailed treatment of this method, including thorough analysis of truncation error and a brief indication of problems posed by the presence of mixed derivatives in the differential operator. We will then consider the more general multi-level Douglas–Gunn schemes that are applicable to second-order hyperbolic problems and in the development of more elaborate techniques for parabolic equations. We close the section with some discussion concerning details of implementation, including differences from that of ADI and LOD, and especially boundary condition treatment.

One aspect of the D–G methods that will be apparent for both two-level and multi-level versions is the basic approach to their derivation. One can think of the steps required to construct a D–G scheme as being exactly the reverse of those employed for LOD; namely,

i) discretize the PDE(s), and

ii) split the resulting difference equation(s).

There is no formal factorization as found in the ADI methods, but as we have already seen for LOD (for which this is also true), there nevertheless can be a factorization error.

3.3.1 D–G methods for two-level difference equations

We begin by noting that the problems to which the D–G methods can be applied are quite general and may be expressed as

$$\frac{\partial u}{\partial t} - Lu = f(\boldsymbol{x}, t) \qquad \text{with} \qquad \boldsymbol{x} \in \Omega \subset \mathbb{R}^d , \quad t \in (t_0, t_f] . \tag{3.20}$$

Initial conditions

$$u(\boldsymbol{x}, t_0) = u_0(\boldsymbol{x}) ,$$

and boundary conditions

$$B_{\partial\Omega} u = g(\boldsymbol{x}, t) \qquad \boldsymbol{x} \in \partial\Omega ,$$

are required to formulate a well-posed initial boundary-value problem. In (3.20) L is a general second-order linear elliptic operator (but could be a first-order spatial operator), and $B_{\partial\Omega}$ is any combination of (linear) boundary operators leading to a well-posed problem. It will be formally assumed that $u(\boldsymbol{x}, t) \in C^2(\Omega) \times C^1((t_0, t_f])$, but in many situations it is sufficient to instead have $u(\boldsymbol{x}, t) \in H^2(\Omega) \times L^1((t_0, t_f])$.

Any *two-level* difference approximation of Eq. (3.20) can be cast in the form

$$\left(I + \mathcal{A}^{n+1}\right) u^{n+1} + \mathcal{B}^n u^n = s^n , \tag{3.21}$$

where it is assumed that all discrete boundary data are contained in the matrices $\mathcal{A}$ and $\mathcal{B}$, as well as in s^n, as appropriate. We also note that the matrices $\mathcal{A}$ and $\mathcal{B}$ may be time dependent, as implied by the notation; in particular, the matrix $\mathcal{A}$ will always represent advanced time level difference equation coefficients and correpondingly, $\mathcal{B}$ will denote previous time-level information. Hence, with this understood, we will often suppress the superscripts to simplify notation. Finally, we remark that the grid function s^n is taken to be known for all time, and in many cases (*e.g.*, the Crank–Nicolson method) it will actually contain information from the advanced time level.

Construction of the basic D–G method

In the treatment of Eq. (3.21) to be provided here we will closely follow the paper by Douglas and Gunn [47], but with occasional changes in notation. As noted earlier, the next step after obtaining the above discretization is splitting. To accomplish this we begin by noting that the matrix $\mathcal{A}$ can be written as a sum of matrices,

$$\mathcal{A} = \sum_{i=1}^{q} \mathcal{A}_i \, .$$

Typically, but not always, $q = d$, and each $\mathcal{A}_i$ is a matrix associated with discretization in a specific spatial direction. (It is worthwhile to recall from Chap. 2 that the steady-state form of P–R ADI was formulated in an analogous fashion.) Also, let $v^{(1)}, v^{(2)}, \ldots, v^{(q-1)}$ denote intermediate estimates of u^{n+1}, with $v^{(q)} = u^{n+1}$.

The Douglas–Gunn procedure for determining each of the $v^{(i)}$s, and hence u^{n+1}, is the following. For each $i = 1, 2, \ldots, q$ solve the system of equations

$$\left(I + \mathcal{A}_i^{n+1}\right) v^{(i)} + \sum_{j=1}^{i-1} \mathcal{A}_j^{n+1} v^{(j)} + \sum_{j=i+1}^{q} \mathcal{A}_j^{n+1} u^n + \mathcal{B}^n u^n = s^n \, , \tag{3.22}$$

where each $\mathcal{A}_i^{n+1}$ is a $N{\times}N$ matrix that is usually compactly banded and tridiagonal if second-order centered spatial discretizations are employed. We have here included the time-level superscripts for emphasis.

Now observe that the first summation in (3.22) contains intermediate solution vectors that all have been calculated during previous substeps of the current time step, while the second summation contains portions of the split matrix from beyond the current substep—but operating on results from the preceding (and thus, known) time step. (Note that this resembles the structure of Gauss–Seidel iteration; *viz.*, use new information in succeeding substeps as soon as it is generated. Indeed, this was done in all of the time-splitting schemes discussed earlier.)

It should be clear from (3.22) that there is a considerable amount of arithmetic needed to evaluate the q equations of this form unless extra storage is used to avoid recalculations. But this can be reduced significantly by the following rearrangement. If we write (3.22) for the $i-1^{th}$ substep we obtain (with time indexing of matrices suppressed)

$$\left(I + \mathcal{A}_{i-1}\right) v^{(i-1)} + \sum_{j=1}^{i-2} \mathcal{A}_j v^{(j)} + \sum_{j=i}^{q} \mathcal{A}_j u^n + \mathcal{B} u^n = s^n \, .$$

Then subtracting this from Eq. (3.22) yields

$$\left(I + \mathcal{A}_i\right) v^{(i)} - \left(I + \mathcal{A}_{i-1}\right) v^{(i-1)} + \sum_{j=1}^{i-1} \mathcal{A}_j v^{(j)} - \sum_{j=1}^{i-2} \mathcal{A}_j v^{(j)} + \sum_{j=i+1}^{q} \mathcal{A}_j u^n - \sum_{j=i}^{q} \mathcal{A}_j u^n = 0 \, ,$$

or

$$\left(I + \mathcal{A}_i\right) v^{(i)} - v^{(i-1)} - \mathcal{A}_i u^n = 0 \, .$$

It is clear from this that from the standpoint of computational efficiency the $v^{(i)}$s should be calculated from the following formulas:

$$\left(I + \mathcal{A}_1\right) v^{(1)} + \sum_{j=2}^{q} \mathcal{A}_j u^n + \mathcal{B} u^n = s^n \, , \tag{3.23a}$$

$$\left(I + \mathcal{A}_i\right) v^{(i)} - v^{(i-1)} - \mathcal{A}_i u^n = 0 \qquad i = 2, 3, \ldots, q \, , \tag{3.23b}$$

with $u^{n+1} = v^{(q)}$. We observe that these equations are of precisely the same form (up to notation) as those given earlier for Douglas–Rachford ADI. Thus, we now see how such a time splitting can be produced systematically from first principles. Moreover, note that (3.23a) is the entire discrete equation while the remaining equation(s) (3.23b) is (are) 1D as in LOD methods treated earlier.

In the following subsections we will demonstrate several important features of this procedure in the context of the 2-D heat equation considered earlier for both ADI and LOD methods. We will first show that each of the individual split steps of D–G schemes are consistent with the PDE; we then show that the order of accuracy of the original unsplit discretization is preserved after splitting. Finally, we will provide results concerning stability and state a general theorem associated with all of these features.

Consistency of individual split steps

It is worthwhile at this point to consider a specific example of applying the preceding formulas. For sake of comparison we again consider the 2-D transient heat equation with Dirichlet boundary conditions already treated via both ADI and LOD methods. Thus, in the general notation of Eq. (3.20) $L = \Delta$, $B_{\partial\Omega} = I$, and we again have Eqs. (3.1). As before, we will apply a Crank–Nicolson discretization leading to Eq. (3.2), repeated here as:

$$\left[I - \frac{k}{2}\left(D_{0,x}^2 + D_{0,y}^2\right)\right] u_{\ell,m}^{n+1} = \left[I + \frac{k}{2}\left(D_{0,x}^2 + D_{0,y}^2\right)\right] u_{\ell,m}^{n} + \frac{k}{2}\left(f_{\ell,m}^{n+1} + f_{\ell,m}^{n}\right) . \tag{3.24}$$

In the notation of the general two-level discrete scheme, Eq. (3.21), we see that

$$\begin{aligned}
\mathcal{A} &\equiv -\frac{k}{2}\left(D_{0,x}^2 + D_{0,y}^2\right) , \\
\mathcal{B} &\equiv -\left[I + \frac{k}{2}\left(D_{0,x}^2 + D_{0,y}^2\right)\right] , \\
s^n &\equiv \frac{k}{2}\left(f^{n+1} + f^n\right) .
\end{aligned}$$

Furthermore, it is clear that we can set

$$\mathcal{A} = \mathcal{A}_1 + \mathcal{A}_2 = -\frac{k}{2}A_1 - \frac{k}{2}A_2$$

with $A_1 \equiv D_{0,x}^2$ and $A_2 \equiv D_{0,y}^2$. In this simple case, $q = 2$, and we can write Eqs. (3.22) as

$$\left(I - \frac{k}{2}A_1\right) v^{(1)} = \left[I + \frac{k}{2}\left(A_1 + A_2\right)\right] u^n + \frac{k}{2}A_2 u^n + \frac{k}{2}\left(f^{n+1} + f^n\right) , \tag{3.25a}$$

$$\left(I - \frac{k}{2}A_2\right) v^{(2)} = \left[I + \frac{k}{2}\left(A_1 + A_2\right)\right] u^n + \frac{k}{2}A_1 v^{(1)} + \frac{k}{2}\left(f^{n+1} + f^n\right) , \tag{3.25b}$$

for the two split steps.

We will now show that <u>both</u> of these individual split steps are consistent with the PDE when the $v^{(i)}$s are viewed as approximations to u^{n+1}. Note that this is in sharp contrast to the situation for LOD methods treated earlier. It should be clear that if $v^{(1)}$ is a consistent approximation to u^{n+1}, then the second split step is consistent since in this case it is analogous to the <u>unsplit</u> Crank–Nicolson scheme. Hence, we need only demonstrate consistency of Eq. (3.25a).

We begin by expanding $v^{(1)}$ in a Taylor series under the assumption that it is an approximation to u^{n+1}, as needed. Thus,

$$v^{(1)} = u^n + k u_t^n + \frac{k^2}{2} u_{tt}^n + \cdots .$$

Substitution of this into (3.25a) results in

$$\left(I - \frac{k}{2}A_1\right) u^n = \left[I + \frac{k}{2}\left(A_1 + A_2\right)\right] u^n + \frac{k}{2}A_2 u^n + \frac{k}{2}\left(f^{n+1} + f^n\right)$$

$$-\left(I - \frac{k}{2}A_1\right)\left[ku_t^n + \frac{k^2}{2}u_{tt}^n + \cdots\right],$$

and rearrangement leads to

$$ku_t^n = k\left(A_1 + A_2\right)u^n + \frac{k}{2}\left(f^{n+1} + f^n\right) + \mathcal{O}(k^2).$$

Also note that

$$f^{n+1} + f^n = 2f^n + \mathcal{O}(k),$$

and from the definitions of the A_is, $A_1 + A_2 = \Delta + \mathcal{O}(h^2)$. Thus, for arbitrary n division by k leaves

$$u_t^n = \Delta u^n + f^n + \mathcal{O}(k + h^2).$$

Hence, we have shown that the first split step of the Douglas–Gunn formalism is consistent with the original PDE, and alone it is first-order accurate in time. Moreover, we have already argued that consistency of the first split step implies consistency of the second in this case. We leave formal demonstration of this to the reader. A major consequence of this feature of the D–G schemes is that it removes the question of when in time to evaluate boundary conditions for intermediate split-step time levels. In particular, we have demonstrated that the intermediate results are consistent approximations to the $n + 1^{th}$ time level results, so boundary conditions should be implemented accordingly.

Accuracy of the complete method

We next show that the two consistent steps of the D–G procedure, when combined, lead to recovery of full second-order accuracy of the original discretization. We begin by writing the equivalent form given in an earlier subsection, namely Eqs. (3.23), rearranged as

$$\left(I - \frac{k}{2}A_1\right)v^{(1)} = \left[I + \frac{k}{2}\left(A_1 + 2A_2\right)\right]u^n + \frac{k}{2}\left(f^{n+1} + f^n\right), \tag{3.26a}$$

$$\left(I - \frac{k}{2}A_2\right)u^{n+1} = v^{(1)} - \frac{k}{2}A_2u^n. \tag{3.26b}$$

As we have done previously, we solve the first of these for $v^{(1)}$,

$$v^{(1)} = \left(I - \frac{k}{2}A_1\right)^{-1}\left[I + \frac{k}{2}\left(A_1 + 2A_2\right)\right]u^n + \frac{k}{2}\left(I - \frac{k}{2}A_1\right)^{-1}\left(f^{n+1} + f^n\right),$$

and substitute this result into the second equation. This yields

$$\left(I - \frac{k}{2}A_2\right)u^{n+1} = \left(I - \frac{k}{2}A_1\right)^{-1}\left\{\left[I + \frac{k}{2}\left(A_1 + 2A_2\right)\right]u^n + \frac{k}{2}\left(f^{n+1} + f^n\right)\right\} - \frac{k}{2}A_2u^n.$$

Now multiply this result by $\left(I - \frac{k}{2}A_1\right)$ to obtain the required form on the left-hand side:

$$\left(I - \frac{k}{2}A_1\right)\left(I - \frac{k}{2}A_2\right)u^{n+1} = \left[I + \frac{k}{2}\left(A_1 + 2A_2\right)\right]u^n + \frac{k}{2}\left(f^{n+1} + f^n\right) - \frac{k}{2}\left(I - \frac{k}{2}A_1\right)A_2u^n,$$

or

$$\left(I - \frac{k}{2}A_1\right)\left(I - \frac{k}{2}A_2\right)u^{n+1} = \left[I + \frac{k}{2}\left(A_1 + A_2\right)\right]u^n + \frac{k}{2}\left(f^{n+1} + f^n\right) + \frac{k^2}{4}A_1A_2u^n.$$

Now recall that

$$\left(I - \frac{k}{2}A_1\right)\left(I - \frac{k}{2}A_2\right) = \left[I - \frac{k}{2}\left(A_1 + A_2\right)\right] + \frac{k^2}{4}A_1A_2,$$

which leads to the desired result

$$\left[I - \frac{k}{2}\left(A_1 + A_2\right)\right] u^{n+1} = \left[I + \frac{k}{2}\left(A_1 + A_2\right)\right] u^n + \frac{k}{2}\left(f^{n+1} + f^n\right) - \frac{k^2}{4} A_1 A_2 (u^{n+1} - u^n)\,. \tag{3.27}$$

We see from this that if $u \in C^1$ we have recovered the <u>unsplit</u> Crank–Nicolson scheme to within $\mathcal{O}(k^3)$ local accuracy, and this has been done for a general nonhomogeneous problem without any requirement of commutativity of the matrices A_1 and A_2. We also note that this is still true for problems with time-dependent coefficients (a major problem for many time-split methods), the proof of which is left as an exercise for the reader. In this regard, one should observe that the term $2A_2$ of Eq. (3.26a) would be replaced with $A_2^n + A_2^{n+1}$ if temporal accuracy is to be maintained for this case. This is a very important issue during implementation.

Stability

We now briefly consider stability of the Douglas–Gunn time-splitting procedure. To do this it is convenient to introduce the so-called *δ-form* of the method, as done in [47]. We also remark that this form is now widely used in computational algorithms and is the most efficient of the forms we have considered.

We start by observing that Eq. (3.23b) can be written as

$$(I + \mathcal{A}_i)\left(v^{(i)} - u^n\right) = v^{(i-1)} - u^n$$

by subtracting u^n from both sides. Now define

$$\delta v^{(i)} \equiv v^{(i)} - u^n$$

so that

$$(I + \mathcal{A}_i)\,\delta v^{(i)} = \delta v^{(i-1)} \qquad \forall \quad i = 2, \ldots, q\,. \tag{3.28}$$

Similarly, by subtracting $(I + \mathcal{A}_1)\,u^n$ from both sides of (3.23a), we obtain

$$(I + \mathcal{A}_1)\,\delta v^{(1)} = s^n - (I + \mathcal{A})u^n - \mathcal{B}u^n\,. \tag{3.29}$$

It is of interest to observe that the right-hand side of this equation is the same as the original, unsplit discrete equation, but evaluated only at time level n. This implies that if a steady solution exists, this approach will accurately find it. But beyond this is the fact that correct representation up to $\mathcal{O}(k^2)$ of temporal behavior is contained in the left-hand side of (3.29), thus indicating consistency of the first split step.

We will use Eq. (3.29) to show that the split scheme, viewed in complete generality (in contrast to the specific example problem treated above), is equivalent to the original unsplit difference equations plus a small perturbation. Then, for linear problems, if the unsplit scheme is stable so must be the split method.

From (3.28) we work backwards starting with $i = q$ to obtain

$$\begin{aligned}
\delta v^{(q-1)} &= (I + \mathcal{A}_q)\delta v^{(q)} \\
\delta v^{(q-2)} &= (I + \mathcal{A}_{q-1})\delta v^{(q-1)} = (I + \mathcal{A}_{q-1})(I + \mathcal{A}_q)\delta v^{(q)} \\
&\ \vdots \\
\delta v^{(1)} &= (I + \mathcal{A}_2)\delta v^{(2)} = \cdots = \left[\prod_{j=2}^{q}(I + \mathcal{A}_j)\right]\delta v^{(q)}\,,
\end{aligned}$$

or

$$v^{(1)} - u^n = \left[\prod_{j=2}^{q}(I + \mathcal{A}_j)\right](u^{n+1} - u^n)\,.$$

Substitution of this into Eq. (3.29) yields

$$\left[\prod_{j=1}^{q}(I+\mathcal{A}_j)\right](u^{n+1}-u^n)+(I+\mathcal{A})u^n+\mathcal{B}u^n=s^n\,. \tag{3.30}$$

Now let σ be the *multi-index* $(i_1, i_2, \ldots, i_m)$ with $i_1 < i_2 < \cdots < i_m$, with $2 \le m \le q$, and $|\sigma| = i_1 + \cdots + i_m$. Then expanding the product in Eq. (3.30) leads to

$$(I+\mathcal{A})u^{n+1}+\mathcal{B}u^n+\sum_{2\le|\sigma|\le q}\mathcal{A}_\sigma(u^{n+1}-u^n)=s^n\,, \tag{3.31}$$

where

$$\mathcal{A}_\sigma=\prod_{\ell=1}^{m}\mathcal{A}_{i_\ell}\,.$$

We can observe from this that, first, Eq. (3.31) is the perturbation of the original difference equation, Eq. (3.21), mentioned above, and it is equivalent to the splitting error analyzed earlier for other methods. Second, if $u \in C^1$ the factor $u^{n+1} - u^n$ is $\mathcal{O}(k)$ and third, from the definition of the matrices $\mathcal{A}_i$ and $\mathcal{A}_\sigma$, the $\mathcal{A}_\sigma$s must be no more than $\mathcal{O}(k^2)$. Hence, the perturbation is, in general, of the same order as the splitting error demonstrated for the model heat equation problem treated in detail above; so for a linear problem this would not grow in time if the unsplit scheme is stable.

In Douglas and Gunn [47] various results concerning consistency, accuracy and stability are proven as separate theorems. We collect that information in the following theorem. The foregoing analyses we have presented here have not been done with sufficient rigor to constitute a proof of the theorem. On the other hand, they at least suggest a direction to be taken in constructing such a proof.

Theorem 3.1 (Douglas & Gunn Time Splitting) *Suppose that a consistent and stable unsplit approximation has been provided for Eq.* (3.20)*, and the corresponding split scheme has been rearranged to produce the perturbed unsplit form Eq.* (3.31)*. Then the following hold:*

i) Eq. (3.31) *is consistent in the norm* $\|\cdot\|$ *provided*

$$\lim_{h,k\to 0}\ \max_{t_0\le nk\le t_f}\left\|k^{-1}\sum_{2\le|\sigma|\le q}\mathcal{A}_\sigma(u^{n+1}-u^n)\right\|=0\,.$$

ii) Eq. (3.31) *is stable provided the operators* $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_q$ *and* $\mathcal{B}$ *have the following properties:*

(a) $\mathcal{A}_i$ *is nonnegative definite* $\forall\ i = 1, \ldots, q$*;*

(b) $\mathcal{B}$ *is Hermitian;*

(c) $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_q$, $\mathcal{B}$ *commute.*

iii) Under the above hypotheses, and in addition assuming that $u \in C^\infty(\Omega)$*, the formal global accuracy of the split scheme is the same as that of the original unsplit scheme.*

There are several things to note regarding this theorem. First, part *i)* of the theorem is obvious in light of our above discussions. In part *ii)*, property *(a)* will hold for essentially any typical discretization and splitting; nevertheless, it is possible to construct splittings of $\mathcal{A}$ for which this could be violated—and this should be avoided. The Hermitian property imposed on the matrix $\mathcal{B}$ will often not hold. Next, commutativity required by *ii)*, *(c)*—which, in general, will not hold—is not necessary for maintaining the order of accuracy implied by the theorem statement, at least in the context of Crank–Nicolson discretizations. The same may also be true for stability; however, it should be noted that this is somewhat less easily tested because the specific types of problems that lead to noncommutativity can also lead to instabilities of the unsplit Crank–Nicolson method—namely, time-dependent coefficients in the differential operators. Finally, with regard to accuracy, the requirement of C^∞ solutions was a standard one at the time the original analyses of Douglas and Gunn were performed. It is recognized in modern numerical analysis that much less than this is needed. In particular, it is usually sufficient to have $u \in H^m(\Omega)$ where m is the order of the derivative in the leading term of the truncation error expansion; *i.e.*, this leading term is in $L^2(\Omega)$ and is therefore bounded almost everywhere, and has finite energy, in Ω.

There is one additional item associated with this two-level splitting procedure that we will consider briefly. It is treatment of problems containing mixed derivatives. We refer the reader to Mitchell and Griffiths [16] for a thorough analysis of this problem. Here, we first point out that it is a problem that seems to have no good solution in the context of time split schemes. Possibly the most successful attempts are embodied in the family of methods proposed by Beam and Warming [48] in the early 1980s. In the context of the splitting procedures just described, a possible alternative is to perform additional split steps to separately treat the mixed derivative, analogous to the operator splitting already treated in conjunction with LOD methods. In two space dimensions if one considers difference operators defined along diagonal lines of the grid as suggested in [47], it is easily checked that two such operators are needed to construct the usual centered approximation to a second mixed derivative. In this case, one can take $q = 4$ and proceed just as we have already done. Of course this entails essentially double the arithmetic per time step, and it requires much more complicated data structures for the diagonal direction solves. The situation is far more complicated in 3D; thus, this does not seem to be a very practical approach to this problem. We mention, in passing, that for nonlinear problems requiring iteration at each time step, simply lagging the mixed derivative and evaluating it with currently-known data usually works quite well. It does not generally slow convergence of the nonlinear iterations, and it typically has little effect on stability (which will already have been influenced by the nonlinearities). Moreover, this approach (iteration) is often more efficient, even for linear problems—especially in 3D, than performing calculations required by additional splits.

3.3.2 D–G methods for multi-level difference equations

Following Douglas and Gunn [47] we define a *(M+2)-level* difference equation approximating a time-dependent partial differential equation by

$$(I + \mathcal{A})u^{n+1} + \sum_{m=0}^{M} \mathcal{B}_m u^{n-m} = s^n\,, \qquad M \le n\,, \tag{3.32}$$

where $u^0, u^1, \ldots, u^n$ are assumed to have been specified. Clearly, in general, such methods have the same disadvantages as do multi-step schemes for ODEs: namely, they are not generally self starting, and they require considerably more storage, especially in the PDE case. But in addition, stability is more difficult to analyze, and subtle errors can occur due to short periods of instability followed by return of stable behavior before a numerical solution actually diverges to infinity. On the other hand, an advantage in the PDE case is that such methods can be used for both parabolic and higher-order hyperbolic problems (*e.g.*, the second-order wave equation).

Basic construction

We again assume $\mathcal{A}$ to be decomposable as $\sum_i \mathcal{A}_i$. Also, we let φ_m, $m = 0, 1, \ldots, M$, be constants such that

$$\sum_{m=0}^{M} \varphi_m = 1\,,$$

and define

$$u^{n+1^*} \equiv \sum_{m=0}^{M} \varphi_m u^{n-m}\,.$$

Thus, if the φ_ms are chosen properly, u^{n+1^*} provides a prediction, via extrapolation, of u^{n+1}.

We can now form the split version of Eq. (3.32) in a way that is analogous to the splitting of two-level methods given in the preceding section:

$$(I + \mathcal{A}_i)v^{(i)} + \sum_{j=1}^{i-1} \mathcal{A}_j v^{(j)} + \sum_{j=i+1}^{q} \mathcal{A}_j u^{n+1^*} + \sum_{m=0}^{M} \mathcal{B}_m u^{n-m} = s^n \,, \qquad i = 1, 2, \ldots, q\,. \tag{3.33}$$

As was true for the two-level schemes treated earlier, this equation can be simplified and made more computationally efficient, analogous to Eqs. (3.23); in fact, even a δ-form can be constructed. We leave investigation of this to the interested reader and consider the following example while maintaining the form given in (3.33).

Example problem: the 3-D wave equation

We, as yet, have not treated either 3-D problems, or hyperbolic problems. We will use this multi-level scheme to remedy this omission. We consider the initial boundary-value problem for the nonhomogeneous 3-D wave equation expressed as

$$u_{tt} - (u_{xx} + u_{yy} + u_{zz}) = f(x, y, z, t) \qquad \text{on } \Omega \times (t_0, t_f]\,, \tag{3.34}$$

with $\Omega \subset \mathbb{R}^3$assumed to be a rectangle. Initial data are given as

$$\begin{aligned} u(x, y, z, t_0) &= u_0(x, y, z) \\ u_t(x, y, z, t_0) &= u_{t,0}(x, y, z)\,, \end{aligned}$$

and boundary conditions are prescribed as

$$u(x, y, z, t) = g(x, y, z, t) \qquad \text{on } \partial\Omega\,.$$

If we were to formulate (3.34) as a first-order system we could directly apply the Crank–Nicolson procedure to obtain a two-level scheme and proceed as we have done in the preceding section. Here, we will employ a somewhat *ad hoc* procedure not very different from what is often done for the 1-D wave equation. Namely, we will use centered-difference discretizations in both space and time, but we will average the spatial parts over all three time levels used in the temporal discretization in order to maintain full second-order accuracy. Thus, we can express (3.34) in discrete form as

$$u^{n-1} - 2u^n + u^{n+1} = \frac{k^2}{3}\left[\left(D^2_{0,x} + D^2_{0,y} + D^2_{0,z}\right)\left(u^{n-1} + u^n + u^{n+1}\right) + \left(f^{n-1} + f^n + f^{n+1}\right)\right]\,, \tag{3.35}$$

where, as often done, we have suppressed grid-point indexing to simplify notation. If we now make our usual identifications between difference operators and their corresponding matrices of coefficients, *viz.*, $D^2_{0,x} \equiv A_1$, $D^2_{0,y} \equiv A_2$ and $D^2_{0,z} \equiv A_3$, we can write the above as

$$\begin{aligned} &\left[I - \frac{k^2}{3}(A_1 + A_2 + A_3)\right] u^{n+1} - \left[2I + \frac{k^2}{3}(A_1 + A_2 + A_3)\right] u^n \\ &\qquad + \left[I - \frac{k^2}{3}(A_1 + A_2 + A_3)\right] u^{n-1} = \frac{k^2}{3}\left(f^{n-1} + f^n + f^{n+1}\right)\,. \end{aligned} \tag{3.36}$$

Clearly, $M = 1$ in this case, and we can identify the matrices appearing in Eq. (3.32) as

$$\begin{aligned}
\mathcal{A} &\equiv -\frac{k^2}{3}(A_1 + A_2 + A_3)\ , \\
\mathcal{B}_0 &\equiv -\left[2I + \frac{k^2}{3}(A_1 + A_2 + A_3)\right]\ , \\
\mathcal{B}_1 &\equiv \left[I - \frac{k^2}{3}(A_1 + A_2 + A_3)\right]\ , \\
s^n &\equiv \frac{k^2}{3}\left(f^{n-1} + f^n + f^{n+1}\right)\ .
\end{aligned}$$

Thus, we can express (3.36) as

$$(I + \mathcal{A})u^{n+1} + \mathcal{B}_0 u^n + \mathcal{B}_1 u^{n-1} = s^n\,. \tag{3.37}$$

We now split the matrix $\mathcal{A}$ in a manner analogous to what was done in the two-level case: $\mathcal{A} = \mathcal{A}_1 + \mathcal{A}_2 + \mathcal{A}_3$ with obvious definitions for the $\mathcal{A}_i$s. Then the directionally-split equations are

$$\begin{aligned}
(I + \mathcal{A}_1)v^{(1)} + \sum_{j=1}^{0}\mathcal{A}_j v^{(j)} + \sum_{j=2}^{3}\mathcal{A}_j u^{n+1^*} + \sum_{m=0}^{1}\mathcal{B}_m u^{n-m} &= s^n \\
(I + \mathcal{A}_2)v^{(2)} + \sum_{j=1}^{1}\mathcal{A}_j v^{(j)} + \sum_{j=3}^{3}\mathcal{A}_j u^{n+1^*} + \sum_{m=0}^{1}\mathcal{B}_m u^{n-m} &= s^n \\
(I + \mathcal{A}_3)v^{(3)} + \sum_{j=1}^{2}\mathcal{A}_j v^{(j)} + \sum_{j=4}^{3}\mathcal{A}_j u^{n+1^*} + \sum_{m=0}^{1}\mathcal{B}_m u^{n-m} &= s^n\,.
\end{aligned}$$

It is to be noted that the first sum in the first of these equations, and the second sum in the third, are vacuous. Thus, we can express these as

$$\begin{aligned}
\left(I - \frac{k^2}{3}A_1\right)v^{(1)} = s^n + \frac{k^2}{3}(A_2 + A_3)u^{n+1^*} &+ \left[2I + \frac{k^2}{3}(A_1 + A_2 + A_3)\right]u^n \\
&- \left[I - \frac{k^2}{3}(A_1 + A_2 + A_3)\right]u^{n-1}\,,
\end{aligned} \tag{3.38a}$$

$$\begin{aligned}
\left(I - \frac{k^2}{3}A_2\right)v^{(2)} = s^n + \frac{k^2}{3}\left(A_1 v^{(1)} + A_3 u^{n+1^*}\right) &+ \left[2I + \frac{k^2}{3}(A_1 + A_2 + A_3)\right]u^n \\
&- \left[I - \frac{k^2}{3}(A_1 + A_2 + A_3)\right]u^{n-1}\,,
\end{aligned} \tag{3.38b}$$

$$\begin{aligned}
\left(I - \frac{k^2}{3}A_3\right)u^{n+1} = s^n + \frac{k^2}{3}\left(A_1 v^{(1)} + A_2 v^{(2)}\right) &+ \left[2I + \frac{k^2}{3}(A_1 + A_2 + A_3)\right]u^n \\
&- \left[I - \frac{k^2}{3}(A_1 + A_2 + A_3)\right]u^{n-1}\,.
\end{aligned} \tag{3.38c}$$

Before proceeding further, we should make several observations regarding Eqs. (3.38). First, the implicit part of each of these equations is tridiagonal, and is thus easily solved. Second, each step of the split scheme is consistent with the original differential equation, implying straightforward boundary condition implementation. Finally, stability properties of this method are not completely known. It is clear that a von Neumann analysis will provide only necessary conditions for stability in this case. The general theorems of [47] can be applied to infer stability, but these theorems are somewhat weak.

In order to implement the above split scheme we must determine the φ_ms to permit calculation of u^{n+1^*}, and when $n = 1$ we must find u^1. We choose the φ_ms to correspond to a simple linear extrapolation in time; *i.e.*, we set

$$u^{n+1^*} = 2u^n - u^{n-1}.$$

Thus, $\varphi_0 = 2$ and $\varphi_1 = -1$. It is worth mentioning here that linear extrapolation is formally second-order accurate, so consistency of the individual steps is not harmed by this technique. On the other hand, extrapolation can be destabilizing, and its use must be examined for each individual case.

In order to calculate u^1 we recognize that at the first calculated level of the basic difference equations we obtain u^2, so a different formulation is needed for the first time step. We first note that u^0 and u_t^0 are given as initial data. We then assume that u is sufficiently smooth at the initial time to permit expansion in a Taylor series as

$$u^1 = u^0 + k\left(\frac{\partial u}{\partial t}\right)^0 + \frac{k^2}{2}\left(\frac{\partial^2 u}{\partial t^2}\right)^0 + \mathcal{O}(k^3)\,.$$

As already noted, the first two terms in the right-hand side of this expression are known; the third term can be obtained from the differential equation. In particular,

$$\left(\frac{\partial^2 u}{\partial t^2}\right)^0 = (u_{xx} + u_{yy} + u_{zz})^0 + f\,.$$

Hence, u^1 can be determined to within $\mathcal{O}(k^3)$ since u^0 is given throughout Ω. If u^0 is prescribed in functional form, the required derivatives can be computed analytically; otherwise, they must be approximated using the same centered discretizations used throughout the remainder of the formulation. So the procedure is essentially self starting. This completes analysis of the wave equation example.

3.4 Summary

In this chapter we have presented the main implicit time-splitting methods employed for solving multi-dimensional PDEs. We have noted that these implicit procedures are, in general, far more stable (at least for the linear problems treated here) than their explicit counterparts. Moreover, even though many line solves involving compactly-banded matrices are required at each time step, the increase in time step size possible with these methods usually gives them a distinct advantage over explicit methods. The only exceptions to this involve cases for which extremely high time accuracy is required; then, it may be possible that an explicit scheme may be useful. It is also worth noting that even in this case of employing an explicit method, it may be advantageous to utilize (explicit) splitting methods analogous to those presented in this chapter. This is because stability analyses similar to those employed here also apply for splitting of explicit methods; hence, a split explicit scheme will often be more stable than the corresponding unsplit method—but in this case the δ-form is not available, so accuracy can be degraded.

We began this chapter with classical ADI methods derived from Crank–Nicolson time integration, which leads to the well-known Peaceman–Rachford procedure. We again emphasize that at one time this was widely used, but is much less so today because it possesses no natural extension to 3-D problems. This is not the case for the lower-order Douglas–Rachford method and the full second-order accurate procedure due to Douglas and Gunn. For time-accurate calculations the latter is one of the most effective procedures available, especially if it is implemented in δ-form. We continued with the so-called locally-one-dimensional schemes of Russian origin, also known as fractional-step methods. These are simpler, and less accurate, than ADI splittiings; but they are very important because they represent a special case of the general operator splittings of Yanenko [46] which are crucial for constructing solution procedures for the incompressible Navier–Stokes equations. We concluded the chapter with a thorough presentation of what we have termed "general Douglas and Gunn time splittings." We demonstrated that, at least asymptotically, these are more accurate and stable than any of the alternatives. In addition, when these are implemented in δ-form, they offer even further advantages that will be mentioned later. Furthermore, especially in this form, they are among the most efficient of all possible multi-dimensional numerical PDE methods.

Conspicuous by their absence have been discussions of techniques based on unsplit procedures. We mention here that these are now quite widely used because of the myth that elliptic solvers requiring only $\mathcal{O}(N)$ arithmetic operations are readily available. There has been recent wide use of GMRES in the time-dependent PDE context, but with little careful analysis of the arithmetic complexity. In fact, it is readily shown that this complexity far exceeds that of the D–G scheme, and that the difference between the two grows rapidly with problem size. The other motivation for employing unsplit solution procedures arises due to use of complicated discretization procedures involving finite-element methods and/or unstructured gridding. In such cases it is very difficult to construct efficient time stepping methods as we have done herein, and it is far more difficult to accomplish parallelization. Hence, the recommendation must be to try to avoid use of such approaches. What is often not recognized is that most of these were considered in various forms many years ago and were rejected then because they were not efficient. With the advent of extremely powerful computers, much of what has long been understood is now being ignored because, in some respects, it does not matter so much: for only moderate-size problems we are able to produce solutions of adequate accuracy almost independent of efficiency—or lack thereof—of the chosen method. But when one must solve very large, difficult problems requiring weeks to months of CPU time, a factor of two or four in run time can be the difference between results being useful and not. Thus, it is very important that we always use methods that are as efficient as possible, and the time-splitting methods presented in this chapter, especially when they are parallelized—which is very direct, are the most efficient of all possible methods for time-dependent problems.

Chapter 4

Various Miscellaneous Topics

In this chapter we consider several topics that will allow us to generalize our preceding analyses. In previous chapters we have restricted attention to single, linear PDES with problems posed on rectangular domains. It is only the last of these restrictions that will not be treated in the present chapter, and this will be studied in Chap. 5. Here, we will study the following topics: treatment of nonlinear problems for PDEs, analysis of systems of differential equations, mitigation of aliasing (due to under resolution) and the so-called "cell Reynolds number problem," and some advanced forms for discretizing special problems.

4.1 Nonlinear PDEs

There are numerous ways in which nonlinear problems can be treated, and this is despite the fact that the analytical theory is not nearly so well developed for nonlinear PDEs as it is for the linear case. We will begin by giving the general form of the nonlinear PDEs to be studied, and we follow this with several alternative approaches to treating equations of this form. In particular, we will briefly discuss the direct, explicit integration approach. We follow this with a short treatment of basic fixed-point iteration, or Picard iteration. Finally, we develop the Newton–Kantorovich procedure in considerable detail, analyzing both the basic form and the δ-form. We will describe how each of these approaches can be applied for either time-dependent or steady-state problems.

4.1.1 The general nonlinear problem to be considered

Here we will study fairly general nonlinear parabolic equations expressed in the form

$$u_t = Lu + N(u, u_x, u_y) + f(x, y, t) \tag{4.1}$$

on a domain $\Omega \subset \mathbb{R}^d$, $d = 2, 3$, together with appropriate boundary and initial data to comprise a well-posed problem. In Eq. (4.1), L is a linear operator (possibly null) of at most second order, and N is a general nonlinear functional of its arguments. We can write a general two-level discretization of this problem in the form

$$(I + \mathcal{A})u^{n+1} + \mathcal{B}u^n = s^{n+1}\,, \tag{4.2}$$

which is identical to Eq. (3.21) except that now s^{n+1} contains additional terms, corresponding to the nonlinear operator N, which in general must be <u>determined</u> at time level $n + 1$.

4.1.2 Explicit integration of nonlinear terms

The simplest approach to solving the numerical problem of (4.2) is to directly integrate the nonlinearity N. This might be done in the context of splitting this term as in operator splitting described in the preceding chapter, or it could be done as part of the construction of (4.2). In either case, it is not a highly-recommended approach because it can lead to instabilities in the difference equations. We will provide far more robust methods in what follows.

4.1.3 Picard iteration

The next level of complexity corresponds to basic fixed-point iteration, often termed *Picard iteration* especially in the context of solving nonlinear differential equations. We begin by observing that since s includes $N(u)$, we should employ the formal notation

$$(I+\mathcal{A})u^{n+1^{(m+1)}} + \mathcal{B}u^n = s(u^{n+1^{(m)}})\,,$$

or, in standard fixed-point form

$$u^{n+1^{(m+1)}} = (I+\mathcal{A})^{-1}\left[N(u^{n+1^{(m)}}) - \mathcal{B}u^n - s^n\right]\,, \tag{4.3}$$

where we now here explicitly denoted the time-level $n+1$ part of s as N. Clearly this is convergent only if $(I+\mathcal{A})^{-1}N(u)$ satisfies the Lipschitz condition

$$\|(I+\mathcal{A})^{-1}N(u) - (I+\mathcal{A})^{-1}N(v)\| \le K\|u-v\|\,, \tag{4.4}$$

with $K<1$ and u, v in a space of functions consistent with the operators L and N of Eq. (4.1) and the norms employed in Eq. (4.4). It is generally difficult to satisfy inequality (4.4), and as a consequence it is usually preferable to use a form of Newton's method for solving Eq. (4.2). But we remark that if N is no more than quadratic in u, a Lipschitz condition such as (4.4) is often satisfied. It turns out that this is true for the incompressible Navier–Stokes equations of fluyid dynamics, and many of the widely-used solution techniques employed in CFD represent instances of Picard iteration.

4.1.4 The Newton–Kantorovich Procedure

The specific form of Newton's method employed for solution of nonlinear PDEs varies with the investigator. Here, we will consider only the Newton–Kantorovich procedure (see, *e.g.*, Kantorovich and Akilov [49]), but we emphasize that in some particular cases it may be advantageous to employ a straightforward finite-dimensional Newton method applied to already discretized equations. The form we shall employ herein is often termed *quasilinearization*—even though the resulting equation(s) is (are) actually linear and not quasilinear; further details beyond those presented here can be found in, *e.g.*, Ames [50]. We note that the basic difference between this and a standard Newton method approach is that here we will formally apply linearization to the differential equation(s), rather than to the difference equations as would be done in a usual Newton procedure. We observe that within the context of what has already been presented in preceding chapters, this is a more natural approach because we have already treated linear equations in detail, so first linearizing the nonlinear equations of a problem immediately returns us to the familiar case already treated. The important aspect of the present lectures is construction of an iteration procedure that will guarantee convergence to the solution of the original nonlinear problem even though at each iteration we will solve only linear equations.

A basic approach

The simplest form of quasilinearization, from a conceptual standpoint, is that introduced by Bellman (see Bellman and Kalaba [51]) in which dependent variables, as well as their derivatives, are viewed as independent variables in a Fréchet–Taylor expansion of any nonlinear terms present in the problem. For example, suppose $N = N(u, u_x, u_y)$; then we linearize N as follows:

$$N(u,u_x,u_y) = N(u^{(0)},u_x^{(0)},u_y^{(0)}) + \left(\frac{\partial N}{\partial u}\right)^{(0)}\left(u-u^{(0)}\right) + \left(\frac{\partial N}{\partial u_x}\right)^{(0)}\left(u_x-u_x^{(0)}\right) + \left(\frac{\partial N}{\partial u_y}\right)^{(0)}\left(u_y-u_y^{(0)}\right) + \cdots\,, \tag{4.5}$$

where the (0) superscript denotes an initial guess for u. As will be evident from what follows, this construction is independent of the form of the temporal integration and, hence, can also be used for steady-state nonlinear elliptic problems.

At this point it is worthwhile to consider a specific temporal discretization so as to better see how the Newton–Kantorovich iteration process must interact with the time-stepping algorithm. We apply trapezoidal integration to Eq. (4.1) to obtain

$$u^{n+1} = u^n + \frac{k}{2}\left[L\left(u^{n+1}+u^n\right) + N\left(u^{n+1},\ldots\right) + N\left(u^n,\ldots\right) + f^{n+1} + f^n\right]\,. \tag{4.6}$$

We observe that since u^n is already known, $N(u^n,\ldots)$ can be directly evaluated without using the linearization given in Eq. (4.5). On the other hand, (4.5) must be used to approximate $N(u^{n+1},\ldots)$. Substitution of this into Eq. (4.6) yields

$$\begin{aligned} u^{n+1} = u^n + \frac{k}{2}\Bigg[& L\left(u^{n+1}+u^n\right) + N^{(0)} + \left(\frac{\partial N}{\partial u}\right)^{(0)}\left(u^{n+1}-u^{(0)}\right) + \left(\frac{\partial N}{\partial u_x}\right)^{(0)}\left(u_x^{n+1}-u_x^{(0)}\right) \\ & + \left(\frac{\partial N}{\partial u_y}\right)^{(0)}\left(u_y^{n+1}-u_y^{(0)}\right) + N(u^n,\ldots) + f^{n+1} + f^n\Bigg]\,. \end{aligned}$$

It should be noted that this equation is <u>linear</u> and can easily be cast in a form to which solution algorithms already studied (such as time-splitting schemes) can be applied. In particular, we have

$$\begin{aligned} \left\{I - \frac{k}{2}\left[L + \left(\frac{\partial N}{\partial u}\right)^{(0)} + \left(\frac{\partial N}{\partial u_x}\right)^{(0)}\frac{\partial}{\partial x} + \left(\frac{\partial N}{\partial u_y}\right)^{(0)}\frac{\partial}{\partial y}\right]\right\}u^{n+1} &= u^n + \frac{k}{2}\Bigg\{Lu^n + N(u^n,\ldots) \\ + N^{(0)} - \left[\left(\frac{\partial N}{\partial u}\right)^{(0)}u^{(0)} + \left(\frac{\partial N}{\partial u_x}\right)^{(0)}u_x^{(0)} + \left(\frac{\partial N}{\partial u_y}\right)^{(0)}u_y^{(0)}\right] &+ f^{n+1} + f^n\Bigg\}\,. \end{aligned} \tag{4.7}$$

We also remark that if $u^{(0)}$ is set equal to u^n, then the first terms omitted from (4.5) are $\mathcal{O}(k^2)$. As a consequence, Eq. (4.7) retains the formal (local) third-order accuracy of the original linear scheme—without iteration—under the assumption that $u \in C^1(t^n, t^{n+1})$. But stability of the time integration method can be degraded if this is done.

At this point it is worthwhile to consider some details of implementing quasilinearization. We note that Eq. (4.7) is 2-D and time dependent, but it is in the form of (4.2); thus, we expect to employ a splitting procedure to improve solution efficiency. At each time step we will generally need to perform some number M of nonlinear iterations to reduce the iteration error to below the level of the local truncation error of the time integration procedure, or below that of the spatial discretization, whichever is larger. The important thing to recognize is that iterations are performed with respect to the <u>complete</u> time step results, and not for individual split steps. Hence, the ordering of the nested loops needed for implementation is: *i*) time step, *ii*) nonlinear iterations and *iii*) split steps. That is, the time-splitting procedure is carried out at the inner-most level of the overall algorithm. Furthermore, we remark that this sequence is the same, regardless of details of methods used in the individual steps. Finally, we note that in the case of steady-state problems, the procedure is altered only with respect to the outer time-integration steps, and (possibly) the inner-most solution technique.

The δ-form

The preceding formulation is a very general and effective way to treat nonlinear PDEs, but there is considerable arithmetic involved in evaluating Eq. (4.7) at each iteration. There is an easy remedy for this, which also exhibits further favorable properties. This is often called *δ-form quasilinearization*, and we introduce this here. Again consider the general nonlinear functional $N(u, u_x, u_y)$ expanded in a Fréchet–Taylor series as done earlier in Eq. (4.5), but now expressed as

$$N(u,u_x,u_y) = N^{(0)} + \left(\frac{\partial N}{\partial u}\right)^{(0)}\delta u + \left(\frac{\partial N}{\partial u_x}\right)^{(0)}\delta u_x + \left(\frac{\partial N}{\partial u_y}\right)^{(0)}\delta u_y + \cdots\,,$$

where $\delta u \equiv u^{n+1} - u^{(0)}$, and $u^{(0)}$ is the initial guess for iterations at time level $n+1$. We comment that for typical problems this initial guess can be taken to be the result from time level n without any difficulty provided the time step k is not extremely large relative to the time scale of temporal changes exhibited by the solution.

We can write the above for a general m^{th} iteration during a time step as

$$N(u, u_x, u_y) = N^{(m)} + \left(\frac{\partial N}{\partial u}\right)^{(m)} \delta u + \left(\frac{\partial N}{\partial u_x}\right)^{(m)} (\delta u)_x + \left(\frac{\partial N}{\partial u_y}\right)^{(m)} (\delta u)_y + \cdots\,, \tag{4.8}$$

and

$$\delta u \equiv u^{n+1} - u^{n+1^{(m)}}\,. \tag{4.9}$$

We now substitute (4.8) into Eq. (4.6) to obtain

$$u^{n+1} = u^n + \frac{k}{2}\left[L(u^{n+1} + u^n) + N^{(m)} + \left(\frac{\partial N}{\partial u}\right)^{(m)} \delta u + \left(\frac{\partial N}{\partial u_x}\right)^{(m)} (\delta u)_x + \left(\frac{\partial N}{\partial u_y}\right)^{(m)} (\delta u)_y \right.$$
$$\left. + N(u^n, \ldots) + f^{n+1} + f^n \right].$$

Next, we observe that the unknown time level $n+1$ solution occurs in two different ways in this equation, namely, both as u^{n+1} and as part of δu. We want the equation to be expressed entirely in terms of δu so we use (4.9) to write $u^{n+1} = \delta u + u^{n+1^{(m)}}$ and substitute into the above. This leads to

$$\delta u + u^{n+1^{(m)}} = u^n + \frac{k}{2}\left[L(\delta u + u^{n+1^{(m)}} + u^n) + N^{(m)} \right.$$
$$\left. + \left(\frac{\partial N}{\partial u}\right)^{(m)} \delta u + \left(\frac{\partial N}{\partial u_x}\right)^{(m)} (\delta u)_x + \left(\frac{\partial N}{\partial u_y}\right)^{(m)} (\delta u)_y + N(u^n, \ldots) + f^{n+1} + f^n \right],$$

which can be solved for δu in the form

$$\left\{ I - \frac{k}{2}\left[L + \left(\frac{\partial N}{\partial u}\right)^{(m)} + \left(\frac{\partial N}{\partial u_x}\right)^{(m)} \frac{\partial}{\partial x} + \left(\frac{\partial N}{\partial u_y}\right)^{(m)} \frac{\partial}{\partial y}\right]\right\} \delta u =$$
$$u^n - u^{n+1^{(m)}} + \frac{k}{2}\left[L\left(u^{n+1^{(m)}} + u^n\right) + N^{(m)} + N(u^n, \ldots) + f^{n+1} + f^n\right]\,. \tag{4.10}$$

There are several remarks that should be made regarding Eq. (4.10). First, we should observe that the right-hand side of (4.10) is simply the residual of the original (semi-) discrete equation (4.6). Hence, as $u^{n+1^{(m)}} \to u^{n+1}$ the right-hand side of (4.10) $\to 0$. This, in turn, implies that $\delta u \to 0$—by linearity. Thus, if the Newton–Kantorovich iterations converge, they converge to a solution of the original nonlinear problem. Second, we note that the algorithm to be implemented for solving (4.10) is the same as that given earlier. Moreover, in the context of time splitting within this solution process, it should be clear from (4.10) that if convergence is tested with respect to the original unsplit formula shown on the right-hand side of this expression, then splitting errors will also be automatically iterated out of the solution.

A third important point is that the spatial discretization need not be the same on both left- and right-hand sides of Eq. (4.10). This is because as noted, for example, in [14], it is not necessary to use exact Jacobian matrices to achieve quadratic convergence with Newton methods. (It should be recognized, however, that if the approximate Jacobian matrix deviates too far from the exact one, convergence rates can deteriorate significantly.) The advantage of this in the context of solving nonlinear PDEs is that it is often desirable to use relatively high-order discretizations, but these may lead to either badly structured matrices (possibly precluding the ability to employ time-splitting techniques) or numerical instabilities—or both. With the present formulation this can be remedied by employing very robust low-order procedures to construct the approximate Jacobian matrix on the left-hand side (thus preserving compact tridiagonal structure of individual split steps) while utilizing quite sophisticated discretizations on the right-hand side (which is evaluated explicitly) to achieve good accuracy. One might view this as a form of preconditioning; namely, low-order methods are robust and yield tridiagonal matrices. Hence, they render the problem "easier to solve."

Finally, we should remark on possible growth of errors if iterations are not carried to sufficient convergence at each time step. We earlier noted that quasilinearization is formally third-order accurate in time, locally, <u>without</u> <u>iteration</u>. But, in fact, it is not usually used in this way. Iteration is essentially always necessary, as can easily be seen by considering some details of the first terms that are dropped from the Fréchet–Taylor expansion, Eq. (4.8). For example, one of these terms is the second derivative of the functional N with respect to the solution u. It follows that if

$$\left(\frac{\partial^2 N}{\partial u^2}\right)^{(m)} \gg \mathcal{O}(1)\,,$$

then δu will need to be very small in order for

$$k\left(\frac{\partial^2 N}{\partial u^2}\right)^{(m)} (\delta u)^2 \sim \mathcal{O}(k^3)$$

to effectively hold. Obviously, one remedy is to reduce the time step size k; but it is best to always perform a sufficient number of iterations to guarantee that δu is less than the minimum truncation error of the discretization method(s) being used. In particular, although it is clear from Eq. (4.10) that iteration errors (even if large) may approximately cancel between the first two terms of the right-hand side, this is generally not true for the remaining terms. Hence, iterations should be performed, with a convergence tolerance of order k^3. We remark, however, that it is typical to employ tolerances that are $\mathcal{O}(k^2)$ since for long-time solutions this is the usual order of accuracy, at best.

There are two remaining topics associated with δ-form quasilinearization. These are treatment of boundary conditions and implementation in the context of δ-form time splittings such as the Douglas and Gunn [47] method treated in Chap. 3. We will begin with the boundary condition treatment.

Treatment of nonlinear boundary conditions

Consider a general nonlinear boundary condition, say in 1D:

$$\frac{\partial u^{n+1}}{\partial x} + S(u^{n+1}) = g(x,t)\,, \tag{4.11}$$

where S is a nonlinear function of u. This is a nonlinear Robin condition, and its treatment begins with a Fréchet–Taylor expansion of S just as was done for the nonlinear function N of the differential equation. Thus, we have

$$S(u) = S^{(0)} + \left(\frac{\partial S}{\partial u}\right)^{(0)} \delta u + \cdots\,, \tag{4.12}$$

and substitution of this into (4.11) with introduction of a difference operator to discretize $\partial/\partial x$ yields

$$D_{0,x}u^{n+1} + \left(\frac{\partial S}{\partial u}\right)^{(0)} \delta u = g(x,t) - S^{(0)}\,.$$

Now recall that $u^{n+1} = u^{(0)} + \delta u$, so the above takes the form

$$D_{0,x}\delta u + \left(\frac{\partial S}{\partial u}\right)^{(0)} \delta u = g(x,t) - S^{(0)} - D_{0,x}u^{(0)}\,,$$

or for the general m^{th} iteration

$$D_{0,x}\delta u + \left(\frac{\partial S}{\partial u}\right)^{(m)} \delta u = g(x,t) - S^{(m)} - D_{0,x}u^{(m)}\,, \tag{4.13}$$

which is precisely the discrete form of a linear Robin boundary condition applied to δu. We see that the right-hand side is the original discrete boundary condition residual, so the effect of iterations on boundary conditions is the same as on the PDE itself. Clearly, this can then be combined with the discrete form of the differential equation on the boundary in the usual way (see, *e.g.*, McDonough [1]) to eliminate image-point entries of δu. Finally, we note that even more general boundary conditions can also be treated in this fashion.

Combination of δ-form quasilinearization with δ-form time splitting

There is one final aspect of δ-form quasilinearization that should be given some attention. It is its use in conjunction with δ-form Douglas and Gunn time splitting. We will carry this out only for the two-level D–G schemes; treatment of the $M+2$-level methods is left to the reader. We begin by observing that the general two-level difference equation now takes the form

$$(I+\mathcal{A})\delta u + \mathcal{B}u^n = s^{n+1}\,,$$

where $\mathcal{A}$ and s^{n+1} contain rather different information compared with the linear case. In particular, $\mathcal{A}$ now contains additional terms corresponding to the Fréchet–Taylor expansion of N, and s^n now contains the complete nonlinear discrete operator evaluated at the most recent iterate. Nevertheless, splitting can still be done in the usual way. In general, we have

$$(I+\mathcal{A}_i)(\delta u)^{(i)} + \sum_{j=1}^{i-1}\mathcal{A}_j(\delta u)^{(j)} + \sum_{j=i+1}^{q}\mathcal{A}_j(\delta u)^n + \mathcal{B}u^n = s^{n+1}\,. \tag{4.14}$$

But if iterations have been carried nearly to convergence in the preceding time step, then $(\delta u)^n \cong 0$; and it follows that

$$(I+\mathcal{A}_1)(\delta u)^{(1)} = s^{n+1} - \mathcal{B}u^n\,, \tag{4.15}$$

and in general,

$$(I+\mathcal{A}_i)(\delta u)^{(i)} + \sum_{j=1}^{i-1}\mathcal{A}_j(\delta u)^{(j)} = s^{n+1} - \mathcal{B}u^n\,. \tag{4.16}$$

Similarly,

$$(I+\mathcal{A}_{i-1})(\delta u)^{(i-1)} + \sum_{j=1}^{i-2}\mathcal{A}_j(\delta u)^{(j)} = s^{n+1} - \mathcal{B}u^n\,,$$

and subtracting this from (4.16) yields

$$(I+\mathcal{A}_i)(\delta u)^{(i)} - (I+\mathcal{A}_{i-1})(\delta u)^{(i-1)} + \mathcal{A}_{i-1}(\delta u)^{(i-1)} = 0\,.$$

Thus,

$$(I+\mathcal{A}_i)(\delta u)^{(i)} = (\delta u)^{(i-1)} \qquad \forall \quad i = 2,\ldots,q\,, \tag{4.17}$$

with

$$\delta u \equiv (\delta u)^{(q)}\,.$$

Then

$$u^{n+1^{(m+1)}} = u^{n+1^{(m)}} + \delta u\,. \tag{4.18}$$

We observe that the right-hand side of (4.15) is the residual of the complete discrete equation, implying that $\delta u^{(i)} \to 0 \;\; \forall i = 1,2,\ldots,q$, and hence that the quasilinear $\delta u \to 0$ as $u^{n+1^{(m+1)}} \to u^{n+1}$.

This completes our basic treatment of δ-form quasilinearization. In the next section we will apply this to a nontrivial example problem.

4.2 Systems of PDEs

In this section we will treat linearization of *systems* of partial differential equations. We will begin by introducing a nontrivial nonlinear system, a generalized *transport equation*, and conduct all of our analyses in the context of problems associated with its solution. These will consist of formulating the linear semi-discrete equations for rather specific forms of nonlinearities usually found in transport equations, and then considering the corresponding fully-discrete systems that follow from these.

4.2.1 Example problem—a generalized transport equation

To demonstrate application of the Newton–Kantorovich procedure applied to a nontrivial problem we consider the following system of nonlinear partial differential equations:

$$\boldsymbol{Q}_t + \boldsymbol{\nabla}\cdot\boldsymbol{F}(\boldsymbol{Q}) = \boldsymbol{\nabla}\cdot\boldsymbol{G}(\boldsymbol{Q},\boldsymbol{\nabla}\boldsymbol{Q}) + \boldsymbol{S}(\boldsymbol{Q})\,, \tag{4.19}$$

with $\boldsymbol{Q} = (Q_1(\boldsymbol{x},t),\ldots,Q_{N_v}(\boldsymbol{x},t))^T$; N_v is the number of dependent variables. It is readily seen that if we associate $\boldsymbol{F}$ with advective fluxes, $\boldsymbol{G}$ with diffusive fluxes and $\boldsymbol{S}$ with general nonlinear terms (possibly including spatial gradients) we obtain a form that can be viewed as a general transport equation of the type studied in essentially all areas of the thermal-fluid sciences.

We suppose problems associated with Eq. (4.19) are posed on a domain $\Omega \subset \mathbb{R}^d$, $d = 2,3$, and that appropriate boundary and initial conditions will be prescribed so as to constitute a well-posed problem. Furthermore, for definiteness, in these lectures we will take $d = 2$ and $N_v = 3$ to provide a setting that is nontrivial, and yet not overly burdensome. Within this framework, we might associate Q_1 with the velocity component u, Q_2 with v and Q_3 with T, the temperature. In this 2-D, three-equation setting we can formally express $\boldsymbol{F}$ and $\boldsymbol{G}$ as

$$\boldsymbol{F} = \begin{bmatrix} F_{11} & F_{12} & F_{13} \\ F_{21} & F_{22} & F_{23} \end{bmatrix}, \qquad \text{and} \qquad \boldsymbol{G} = \begin{bmatrix} G_{11} & G_{12} & G_{13} \\ G_{21} & G_{22} & G_{23} \end{bmatrix}.$$

We next observe that

$$\boldsymbol{\nabla}\cdot\boldsymbol{F} = \begin{bmatrix} \dfrac{\partial F_{11}}{\partial x} + \dfrac{\partial F_{21}}{\partial y} \\ \dfrac{\partial F_{21}}{\partial x} + \dfrac{\partial F_{22}}{\partial y} \\ \dfrac{\partial F_{13}}{\partial x} + \dfrac{\partial F_{23}}{\partial y} \end{bmatrix},$$

and each F_{ij} is generally a nonlinear function of the components of $\boldsymbol{Q}$. In typical systems involving transport equations, these would consist of, *e.g.*, $F_{11} = u^2 = Q_1^2$, $F_{12} = uv = Q_1Q_2$, $F_{13} = uT = Q_1Q_3$, *etc.*

Similarly, we have

$$\boldsymbol{\nabla}\cdot\boldsymbol{G} = \begin{bmatrix} \dfrac{\partial G_{11}}{\partial x} + \dfrac{\partial G_{21}}{\partial y} \\ \dfrac{\partial G_{21}}{\partial x} + \dfrac{\partial G_{22}}{\partial y} \\ \dfrac{\partial G_{13}}{\partial x} + \dfrac{\partial G_{23}}{\partial y} \end{bmatrix}.$$

For incompressible flows we might have, *e.g.*, $G_{11} = \nu\partial u/\partial x$ where $\nu = \nu(T)$ is kinematic viscosity and is often given as a nonlinear function of temperature (*e.g.*, via the *Sutherland formula*—see Anderson *et al.*

[52]). Thus, we can express this in our current notation as

$$G_{11} = \nu(Q_3)\frac{\partial Q_1}{\partial x}\,.$$

Analogous forms hold for G_{21}, G_{12} and G_{22} with, *e.g.*,

$$G_{13} = \kappa(Q_3)\frac{\partial Q_3}{\partial x}\,,$$

where κ is thermal diffusivity.

Finally, the nonlinear source term $\boldsymbol{S}(\boldsymbol{Q})$ can be used to represent all other physical phenomena not specifically associated with advection and diffusion—for example, chemical reaction source terms, thermal radiation, buoyancy, rotation and other possible body forces, or porous media effects. As a consequence, $\boldsymbol{S}$ might depend on derivatives of $\boldsymbol{Q}$ in addition to $\boldsymbol{Q}$ itself. We shall not provide a detailed treatment of this term because it can take on so many different forms that it would not be reasonable to attempt an all-encompassing description. With that in mind, we now express Eq. (4.19) component form as

$$Q_{1,t} + \frac{\partial F_{11}}{\partial x} + \frac{\partial F_{21}}{\partial y} = \frac{\partial G_{11}}{\partial x} + \frac{\partial G_{21}}{\partial y} + S_1(\boldsymbol{Q})\,, \tag{4.20a}$$

$$Q_{2,t} + \frac{\partial F_{12}}{\partial x} + \frac{\partial F_{22}}{\partial y} = \frac{\partial G_{12}}{\partial x} + \frac{\partial G_{22}}{\partial y} + S_2(\boldsymbol{Q})\,, \tag{4.20b}$$

$$Q_{3,t} + \frac{\partial F_{13}}{\partial x} + \frac{\partial F_{23}}{\partial y} = \frac{\partial G_{13}}{\partial x} + \frac{\partial G_{23}}{\partial y} + S_3(\boldsymbol{Q})\,. \tag{4.20c}$$

4.2.2 Quasilinearization of systems of PDEs

In this section we will provide details of applying quasilinearization (only in δ-form because this is the most efficient) only to Eq. (4.20a). The remaining two equations are treated in a completely analogous fashion, and carrying out the details is left to the reader.

We begin by again recalling that in general all of the F_{ij}s, G_{ij}s and S_js can be nonlinear functionals of the solution vector $\boldsymbol{Q}$. At the same time, we note that in the context of actual transport equations, the G_{ij}s and S_js are often linear, but we will not assume this in the sequel.

We first linearize F_{11} in the manner given in the preceding section. Namely, we construct the Fréchet–Taylor expansion of F_{11} in δ-form as

$$\begin{aligned} F_{11}(\boldsymbol{Q}) &= F_{11}\left(\boldsymbol{Q}^{(m)}\right) + \sum_{j=1}^{3}\left(\frac{\partial F_{11}}{\partial Q_j}\right)^{(m)}\delta Q_j + \cdots\,, \\ &= Q_1^{2(m)} + 2Q_1^{(m)}\delta Q_1 + \cdots\,. \end{aligned} \tag{4.21}$$

The analogous expression for F_{21} is

$$F_{21}(\boldsymbol{Q}) = Q_1^{(m)}Q_2^{(m)} + Q_2^{(m)}\delta Q_1 + Q_1^{(m)}\delta Q_2 + \cdots\,. \tag{4.22}$$

Next we consider G_{11}. We have

$$G_{11}(\boldsymbol{Q},\boldsymbol{\nabla Q}) = \nu\left(Q_3^{(m)}\right)\frac{\partial Q_1^{(m)}}{\partial x} + \nu\left(Q_3^{(m)}\right)(\delta Q_1)_x + \left(\frac{\partial \nu}{\partial Q_3}\right)^{(m)}\frac{\partial Q_1^{(m)}}{\partial x}\delta Q_3 + \cdots\,. \tag{4.23}$$

Finally, we use the somewhat generic representation of $S_j(\boldsymbol{Q})$:

$$S_j(\boldsymbol{Q}) = S_j\left(\boldsymbol{Q}^{(m)}\right) + \sum_{i=1}^{3}\left(\frac{\partial S_j}{\partial Q_i}\right)^{(m)}\delta Q_i + \cdots\,. \tag{4.24}$$

Clearly, the remaining F_{ij}s, G_{ij}s and S_js appearing in Eqs. (4.20) can be constructed in exactly the same manner.

Substitution of Eqs. (4.21) through (4.24), and analogous ones for the remaining nonlinear terms, into a semi-discrete form of (4.20a) integrated in time via trapezoidal integration leads to

$$\begin{aligned}\delta Q_1 + Q_1^{(m)} = Q_1^n + \frac{k}{2}\Bigg\{ &\frac{\partial}{\partial x}\left[\nu\left(Q_3^{(m)}\right)(\delta Q_1)_x + \nu\left(Q_3^{(m)}\right)\frac{\partial Q_1^{(m)}}{\partial x} + \left(\frac{\partial \nu}{\partial Q_3}\right)^{(m)}\frac{\partial Q_1^{(m)}}{\partial x}\delta Q_3\right] \\ &+\frac{\partial}{\partial y}\left[\nu\left(Q_3^{(m)}\right)(\delta Q_1)_y + \nu\left(Q_3^{(m)}\right)\frac{\partial Q_1^{(m)}}{\partial y} + \left(\frac{\partial \nu}{\partial Q_3}\right)^{(m)}\frac{\partial Q_1^{(m)}}{\partial y}\delta Q_3\right] \\ &-\frac{\partial}{\partial x}\left[Q_1^{2(m)} + 2Q_1^{(m)}\delta Q_1\right] - \frac{\partial}{\partial y}\left[Q_2^{(m)}\delta Q_1 + Q_1^{(m)}\delta Q_2 + Q_1^{(m)}Q_2^{(m)}\right] \\ &+S_1\left(\boldsymbol{Q}^{(m)}\right) + \sum_{i=1}^{3}\left(\frac{\partial S_1}{\partial Q_i}\right)^{(m)}\delta Q_i + \text{terms at time level } n\Bigg\}.\end{aligned}$$

We next rearrange this as

$$\begin{aligned}\Bigg\{I - \frac{k}{2}\Bigg[&\frac{\partial}{\partial x}\left(\nu\left(Q_3^{(m)}\right)\frac{\partial}{\partial x}\,\cdot\right) + \frac{\partial}{\partial y}\left(\nu\left(Q_3^{(m)}\right)\frac{\partial}{\partial y}\,\cdot\right) - 2\frac{\partial}{\partial x}\left(Q_1^{(m)}\cdot\right) - \frac{\partial}{\partial y}\left(Q_2^{(m)}\cdot\right) + \frac{\partial S_1}{\partial Q_1}\Bigg]\Bigg\}\delta Q_1 \\ = Q_1^n - Q_1^{(m)} + \frac{k}{2}\Bigg\{ &\frac{\partial}{\partial x}\left[\nu\left(Q_3^{(m)}\right)\frac{\partial Q_1^{(m)}}{\partial x} + \left(\frac{\partial \nu}{\partial Q_3}\right)^{(m)}\frac{\partial Q_1^{(m)}}{\partial x}\delta Q_3\right] \\ &+\frac{\partial}{\partial y}\left[\nu\left(Q_3^{(m)}\right)\frac{\partial Q_1^{(m)}}{\partial y} + \left(\frac{\partial \nu}{\partial Q_3}\right)^{(m)}\frac{\partial Q_1^{(m)}}{\partial y}\delta Q_3\right] - \frac{\partial}{\partial x}\left(Q_1^{2(m)}\right) \\ &-\frac{\partial}{\partial y}\left(Q_1^{(m)}\delta Q_2\right) - \frac{\partial}{\partial y}\left(Q_1^{(m)}Q_2^{(m)}\right) \\ &+S_1\left(\boldsymbol{Q}^{(m)}\right) + \sum_{i=2}^{3}\left(\frac{\partial S_1}{\partial Q_i}\right)^{(m)}\delta Q_i + \text{terms at time level } n\Bigg\}. \qquad (4.25)\end{aligned}$$

There are several observations regarding this equation. The first obvious one is that it contains <u>three</u> unknowns: δQ_1, δQ_2, δQ_3. The second is that if the original equations are in "conservation law" form, then the δ-form preserves this. This is of particular importance when treating the compressible Euler or Navier–Stokes equations. We next note that have arranged terms so that δQ_2 and δQ_3 appear only on the right-hand side, suggesting that values from a previous iteration might be used. Indeed, one alternative is to solve Eq. (4.25) and the two analogous equations for δQ_2 and δQ_3 in either block Jacobi or block Gauss–Seidel fashion, both of which might be viewed as forms of "diagonal" Newton methods. This is often the simplest approach, but as would be expected, convergence rates are no longer quadratic. On the other hand, the rates are typically somewhat better than linear—in fact, nearly *superlinear*, meaning that there exist constants $C \sim \mathcal{O}(1)$ and $1 < p < 2$ such that

$$\lim_{m\to\infty} \frac{\|\delta \boldsymbol{Q}^{(m+1)}\|}{\|\delta \boldsymbol{Q}^{(m)}\|^p} = C\,. \qquad (4.26)$$

Moreover, the convergence rate depends only very weakly on the number of discrete equations (*i.e.*, on the number of points in the discretization) in the corresponding system, in contrast to the situation for the usual linearly-convergent iteration procedures such as Picard iteration.

We now note that the three semi-discrete equations of the form (4.25) can be collected at each grid point, say (i,j), to produce the following 3×3 matrix representation:

$$\begin{bmatrix} L_{11} & L_{12} & L_{13} \\ L_{21} & L_{22} & L_{23} \\ L_{31} & L_{32} & L_{33} \end{bmatrix}_{(i,j)} \begin{bmatrix} \delta Q_1 \\ \delta Q_2 \\ \delta Q_3 \end{bmatrix}_{(i,j)} = \begin{bmatrix} rhs_1 \\ rhs_2 \\ rhs_3 \end{bmatrix}_{(i,j)} . \tag{4.27}$$

We can easily deduce from Eq. (4.25) that the matrix elements of the first row are as follows.

$$L_{11} = I - \frac{k}{2}\left[D_{0,x}\left(\nu(Q_3^{(m)})D_{0,x}\cdot\right) + D_{0,y}\left(\nu(Q_3^{(m)})D_{0,y}\cdot\right) - 2D_{0,x}\left(Q_1^{(m)}\cdot\right) - D_{0,y}\left(Q_2^{(m)}\cdot\right) + \left(\frac{\partial S_1}{\partial Q_1}\right)^{(m)}\right],$$

$$L_{12} = \frac{k}{2}\left[D_{0,y}\left(Q_1^{(m)}\cdot\right) - \left(\frac{\partial S_1}{\partial Q_2}\right)^{(m)}\right],$$

$$L_{13} = -\frac{k}{2}\left[D_{0,x}\left(\left(\frac{\partial \nu}{\partial Q_3}\right)^{(m)} D_{0,x}Q_1^{(m)}\cdot\right) + D_{0,y}\left(\left(\frac{\partial \nu}{\partial Q_3}\right)^{(m)} D_{0,y}Q_1^{(m)}\cdot\right) + \left(\frac{\partial S_1}{\partial Q_3}\right)^{(m)}\right],$$

$$\begin{aligned} rhs_1 = Q_1^n - Q_1^{(m)} + \frac{k}{2}\Bigg[& D_{0,x}\left(\nu\left(Q_3^{(m)}\right) D_{0,x}Q_1^{(m)}\right) + D_{0,y}\left(\nu\left(Q_3^{(m)}\right) D_{0,y}Q_1^{(m)}\right) \\ & -D_{0,x}\left(Q_1^{2(m)}\right) - D_{0,y}\left(Q_1^{(m)}Q_2^{(m)}\right) + S_1\left(\boldsymbol{Q}^{(m)}\right) + \text{terms at time level } n\Bigg]. \end{aligned}$$

We again observe that the remaining entries of the matrix are of forms very similar to ones shown. We leave their derivation to the ambitious reader.

The next task is to construct the matrices corresponding to the above fully-discrete equations. We first note that if this is done in the context of a time-splitting procedure, then the left-hand-side matrices will contain information from only a single direction. Hence, in the case of second-order centered differencing, as used herein, there will be only three grid-point indices associated with any given point. Thus, at each grid point, say (i,j), expansion of the, *e.g.*, x-direction difference operators of the matrix $\boldsymbol{L}$ will lead to three matrices, each 3×3 in this case, and the collection of these will produce a coupled system of the following form, say for j fixed.

$$\begin{bmatrix} [\boldsymbol{B}]_1 & [\boldsymbol{C}]_1 & \boldsymbol{0} & \cdots & \cdots & \cdots & \cdots & \boldsymbol{0} \\ [\boldsymbol{A}]_2 & [\boldsymbol{B}]_2 & [\boldsymbol{C}]_2 & \boldsymbol{0} & & & & \vdots \\ \boldsymbol{0} & \ddots & \ddots & \ddots & \ddots & & & \vdots \\ \vdots & \ddots & [\boldsymbol{A}]_i & [\boldsymbol{B}]_i & [\boldsymbol{C}]_i & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & \ddots & \ddots & \ddots & \ddots & \boldsymbol{0} \\ \vdots & & & & \ddots & [\boldsymbol{A}]_{N_x-1} & [\boldsymbol{B}]_{N_x-1} & [\boldsymbol{C}]_{N_x-1} \\ \boldsymbol{0} & \cdots & \cdots & \cdots & \cdots & \boldsymbol{0} & [\boldsymbol{A}]_{N_x} & [\boldsymbol{B}]_{N_x} \end{bmatrix} \begin{bmatrix} \delta\boldsymbol{Q}_1 \\ \delta\boldsymbol{Q}_2 \\ \vdots \\ \delta\boldsymbol{Q}_i \\ \vdots \\ \vdots \\ \delta\boldsymbol{Q}_{N_x-1} \\ \delta\boldsymbol{Q}_{N_x} \end{bmatrix} = \begin{bmatrix} \boldsymbol{RHS}_1 \\ \boldsymbol{RHS}_2 \\ \vdots \\ \boldsymbol{RHS}_i \\ \vdots \\ \vdots \\ \boldsymbol{RHS}_{N_x-1} \\ \boldsymbol{RHS}_{N_x} \end{bmatrix} . \tag{4.28}$$

Equation (4.28) holds for all $\delta\boldsymbol{Q} \equiv (\delta Q_1, \delta Q_2, \delta Q_3)^T$, and the indices shown are the x-direction grid-point indices for an arbitrary j index. Also, $\boldsymbol{RHS}_i = (RHS_1, RHS_2, RHS_3)^T$, the fully-discrete forms of the corresponding right-hand side values.

It is worthwhile to examine this equation in somewhat more detail to better understand its structure. If we consider a general i^{th} row, corresponding to discretization of the equations for $\delta\boldsymbol{Q}$ at the i^{th} grid point of the current j line, we obtain the following detailed structure.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}_i \begin{bmatrix} \delta Q_1 \\ \delta Q_2 \\ \delta Q_3 \end{bmatrix}_{i-1} + \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix}_i \begin{bmatrix} \delta Q_1 \\ \delta Q_2 \\ \delta Q_3 \end{bmatrix}_i$$

$$+ \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix}_i \begin{bmatrix} \delta Q_1 \\ \delta Q_2 \\ \delta Q_3 \end{bmatrix}_{i+1} = \begin{bmatrix} RHS_1 \\ RHS_2 \\ RHS_3 \end{bmatrix}_i . \tag{4.29}$$

It should be remarked that in general each of the 3×3 matrices in (4.29) can be dense; moreover, it is obvious that the size of these submatrices simply depends on the number of partial differential equations in the system being solved. Often, it is no more than a few; but, for example, in problems involving finite-rate chemical reactions the number can easily exceed 20. It should also be emphasized that the overall system is *block tridiagonal*. That is, the structure is tridiagonal, but the entries that would have been scalars in the case of solving a single PDE problem are now $N_v \times N_v$ matrices. Nevertheless, as we will show in the next section, the solution procedure is very similar to that in the scalar case. In particular, the main difference between this and the scalar case is that wherever division was needed in the scalar tridiagonal LU decomposition, now formal matrix inversion will be needed. Since the matrices involved are generally nonsparse, as already noted, it will be necessary to employ a Gaussian elimination routine within the sparse LU decomposition to treat such problems.

4.3 Numerical Solution of Block-Banded Algebraic Systems

In this section we will consider the details of solving block-banded systems of the form of Eq. (4.28). We will begin with a subsection containing the basic approach to employing sparse LU decomposition in this setting, and then we will carry out details for the system of Eq. (4.28). We will then briefly consider the total arithmetic operation counts for such solution procedures, and compare these with the alternatives.

4.3.1 Block-banded LU decomposition—how it is applied

We begin by posing a general block-banded problem as

$$\boldsymbol{\mathcal{A}}\delta\boldsymbol{Q} = \boldsymbol{b}\,, \tag{4.30}$$

where bold notation represents matrices and vectors that have arisen in approximations to *systems* of differential equations, rather than from a single equation.

We observe that in light of the LU-decomposition theorem, if $\boldsymbol{\mathcal{A}}$ is nonsingular, then there exists a lower triangular (block-structured) matrix $\boldsymbol{\mathcal{L}}$ and a corresponding upper triangular matrix $\boldsymbol{\mathcal{U}}$ such that

$$\boldsymbol{\mathcal{L}}\boldsymbol{\mathcal{U}} = \boldsymbol{\mathcal{A}}\,. \tag{4.31}$$

This permits us to express the problem (4.30) as

$$\boldsymbol{\mathcal{L}}\boldsymbol{\mathcal{U}}\delta\boldsymbol{Q} = \boldsymbol{b}\,, \tag{4.32}$$

and just as in the scalar case we define $\boldsymbol{r}$ such that

$$\boldsymbol{\mathcal{U}}\,\delta\boldsymbol{Q} = \boldsymbol{r}\,, \tag{4.33}$$

so that

$$\boldsymbol{\mathcal{L}}\boldsymbol{r} = \boldsymbol{b}\,. \tag{4.34}$$

We will present the details of finding the solution vector $\delta\boldsymbol{Q}$ by this procedure in the next subsection. Here we simply note that the approach is completely analogous to that of the scalar case. In particular, it will be shown that $\boldsymbol{\mathcal{L}}$ is a block bi-diagonal lower triangular matrix; hence the system (4.34) is readily solved for $\boldsymbol{r}$. Similarly, the matrix $\boldsymbol{\mathcal{U}}$ is a block bi-diagonal upper triangular matrix with identity matrix blocks on the main diagonal. Hence, the system (4.33) is then easily solved for $\delta\boldsymbol{Q}$ once $\boldsymbol{r}$ has been found.

4.3.2 Block-banded LU decomposition details

From the preceding discussion it should be clear that the main remaining task is construction of the lower and upper triangular matrices $\boldsymbol{\mathcal{L}}$ and $\boldsymbol{\mathcal{U}}$ to be used in the solution process. As already hinted, this is done in a manner quite analogous to that used in the scalar case, but now with any required scalar divisions replaced by (formal) matrix inversion.

The first step in this process is to assume that the matrices $\boldsymbol{\mathcal{L}}$ and $\boldsymbol{\mathcal{U}}$ possess the same band structure found in the respective lower and upper triangular parts of the original matrix $\boldsymbol{\mathcal{A}}$. Thus we seek the elements of

$$\boldsymbol{\mathcal{L}} = \begin{bmatrix} [\boldsymbol{\beta}]_1 & \mathbf{0} & \cdots & \cdots & \cdots & \cdots & \cdots & \mathbf{0} \\ [\boldsymbol{\alpha}]_2 & [\boldsymbol{\beta}]_2 & \mathbf{0} & & & & & \vdots \\ \mathbf{0} & \ddots & \ddots & \ddots & & & & \vdots \\ \vdots & \ddots & [\boldsymbol{\alpha}]_i & [\boldsymbol{\beta}]_i & \ddots & & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & & \ddots & [\boldsymbol{\alpha}]_{N_x-1} & [\boldsymbol{\beta}]_{N_x-1} & \mathbf{0} \\ \mathbf{0} & \cdots & \cdots & \cdots & \cdots & \mathbf{0} & [\boldsymbol{\alpha}]_{N_x} & [\boldsymbol{\beta}]_{N_x} \end{bmatrix},$$

and

$$\boldsymbol{\mathcal{U}} = \begin{bmatrix} [\boldsymbol{I}]_1 & [\boldsymbol{\gamma}]_1 & \mathbf{0} & \cdots & \cdots & \cdots & \cdots & \mathbf{0} \\ \mathbf{0} & [\boldsymbol{I}]_2 & [\boldsymbol{\gamma}]_2 & \mathbf{0} & & & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & & & \vdots \\ \vdots & & \ddots & [\boldsymbol{I}]_i & [\boldsymbol{\gamma}]_i & \ddots & & \vdots \\ \vdots & & & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & & \ddots & \ddots & \ddots & \mathbf{0} \\ \vdots & & & & & \ddots & [\boldsymbol{I}]_{N_x-1} & [\boldsymbol{\gamma}]_{N_x-1} \\ \mathbf{0} & \cdots & \cdots & \cdots & \cdots & \cdots & \mathbf{0} & [\boldsymbol{I}]_{N_x} \end{bmatrix},$$

where in the present case

$$[\boldsymbol{\alpha}]_i = \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} \\ \alpha_{31} & \alpha_{32} & \alpha_{33} \end{bmatrix}_i, \quad [\boldsymbol{\beta}]_i = \begin{bmatrix} \beta_{11} & \beta_{12} & \beta_{13} \\ \beta_{21} & \beta_{22} & \beta_{23} \\ \beta_{31} & \beta_{32} & \beta_{33} \end{bmatrix}_i \quad \text{and} \quad [\boldsymbol{\gamma}]_i = \begin{bmatrix} \gamma_{11} & \gamma_{12} & \gamma_{13} \\ \gamma_{21} & \gamma_{22} & \gamma_{23} \\ \gamma_{31} & \gamma_{32} & \gamma_{33} \end{bmatrix}_i,$$

with $[\boldsymbol{I}]$ denoting the 3×3 identity matrix.

We now recall from Eq. (4.31) that $\boldsymbol{\mathcal{A}} = \boldsymbol{\mathcal{L}}\boldsymbol{\mathcal{U}}$, so if we carry out the indicated multiplication using the above matrices and equate the result to the matrix $\boldsymbol{\mathcal{A}}$, as indicated, we can expect to be able to determine the elements of the $[\boldsymbol{\alpha}]_i$, $[\boldsymbol{\beta}]_i$ and $[\gamma]_i$. Thus, we have

$$\boldsymbol{\mathcal{L}\mathcal{U}} = \left[\begin{array}{ccccccc}
[\boldsymbol{\beta}]_1 & [\boldsymbol{\beta}]_1[\gamma]_1 & \mathbf{0} & \cdots & \cdots & \cdots\ \cdots & \mathbf{0} \\
[\boldsymbol{\alpha}]_2 & [\boldsymbol{\beta}]_2+[\boldsymbol{\alpha}]_2[\gamma]_1 & [\boldsymbol{\beta}]_2[\gamma]_2 & \ddots & & & \vdots \\
\mathbf{0} & \ddots & \ddots & & \ddots & & \vdots \\
\vdots & \ddots & \ddots & \ddots & \ddots & & \vdots \\
\vdots & \ddots & [\boldsymbol{\alpha}]_i & [\boldsymbol{\beta}]_i+[\boldsymbol{\alpha}]_i[\gamma]_{i-1} & [\boldsymbol{\beta}]_i[\gamma]_i & \ddots & \vdots \\
\vdots & & \ddots & \ddots & \ddots & & \vdots \\
\vdots & & \ddots & \ddots & \ddots & \ddots & \vdots \\
\vdots & & \ddots & \ddots & \ddots & \ddots &
\end{array}\right] . \qquad (4.35)$$

We first note that $[\boldsymbol{\beta}]_1 = [\boldsymbol{B}]_1$ by comparing Eqs. (4.35) and (4.28). Similarly, $[\boldsymbol{\alpha}]_i = [\boldsymbol{A}]_i \ \forall\ i = 2,\dots,N_x$. Next, we observe that $[\boldsymbol{\beta}]_i[\boldsymbol{\gamma}]_i = [\boldsymbol{C}]_i$ for all $i = 1,\dots,N_x-1$; hence, if $[\boldsymbol{\beta}]_i$ is known, $[\boldsymbol{\gamma}]_i$ can be directly computed. Formally, this involves matrix inversion: $[\boldsymbol{\gamma}]_i = [\boldsymbol{\beta}]_i^{-1}[\boldsymbol{C}]_i$; but in practice we simply solve a series of linear systems. For our present specific case we have

$$\begin{bmatrix} \beta_{11} & \beta_{12} & \beta_{13} \\ \beta_{21} & \beta_{22} & \beta_{23} \\ \beta_{31} & \beta_{32} & \beta_{33} \end{bmatrix}_i \begin{bmatrix} \gamma_{11} & \gamma_{12} & \gamma_{13} \\ \gamma_{21} & \gamma_{22} & \gamma_{23} \\ \gamma_{31} & \gamma_{32} & \gamma_{33} \end{bmatrix}_i = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix}_i , \qquad (4.36)$$

where $[\boldsymbol{\beta}]_i$ and $[\boldsymbol{C}]_i$ are known, and the elements of $[\boldsymbol{\gamma}]_i$ are to be found. Clearly, this can be done one column at a time as, for example,

$$\begin{bmatrix} \beta_{11} & \beta_{12} & \beta_{13} \\ \beta_{21} & \beta_{22} & \beta_{23} \\ \beta_{31} & \beta_{32} & \beta_{33} \end{bmatrix}_i \begin{bmatrix} \gamma_{11} \\ \gamma_{21} \\ \gamma_{31} \end{bmatrix}_i = \begin{bmatrix} c_{11} \\ c_{21} \\ c_{31} \end{bmatrix}_i .$$

We remark that for systems as small as the present 3×3 one considered here, and certainly if the system is only a 2×2, one would probably simply code Cramer's rule (see, *e.g.*, [14]) as the solution procedure. On the other hand, for systems larger than 3×3 one would almost certainly employ some form of Gaussian elimination, and particularly for the large systems that can arise in the context of finite-rate chemistry simulations, one would probably want to save the Gauss multipliers in order to keep the total arithmetic at $\mathcal{O}(N_v^3)$ for the solves required at each grid point.

The final step in determining the elements of $\mathcal{L}$ and $\mathcal{U}$ is determination of the $[\boldsymbol{\beta}]_i$s for $i > 1$. Again comparing Eqs. (4.35) and (4.28) shows that

$$[\boldsymbol{\beta}]_i + [\boldsymbol{\alpha}]_i[\gamma]_{i-1} = [B]_i ,$$

from which it follows that

$$\begin{bmatrix} \beta_{11} & \beta_{12} & \beta_{13} \\ \beta_{21} & \beta_{22} & \beta_{23} \\ \beta_{31} & \beta_{32} & \beta_{33} \end{bmatrix}_i = \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix}_i - \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} \\ \alpha_{31} & \alpha_{32} & \alpha_{33} \end{bmatrix}_i \begin{bmatrix} \gamma_{11} & \gamma_{12} & \gamma_{13} \\ \gamma_{21} & \gamma_{22} & \gamma_{23} \\ \gamma_{31} & \gamma_{32} & \gamma_{33} \end{bmatrix}_{i-1} . \qquad (4.37)$$

It is easily seen that all components on the right-hand side of this equation are known; hence, at the general i^{th} grid point, the elements of $[\boldsymbol{\beta}]_i$ can be directly computed in $\mathcal{O}(N_v^3)$ arithmetic operations (mainly a matrix-matrix multiplication).

We have by now constructed all of the elements of $\boldsymbol{\mathcal{L}}$ and $\boldsymbol{\mathcal{U}}$ needed to carry out the solution process indicated by Eqs. (4.33) and (4.34). The second of these is solved first to obtain the intermediate vector $\boldsymbol{r}$. The first row of (4.34) written in expanded form is

$$\begin{bmatrix} \beta_{11} & \beta_{12} & \beta_{13} \\ \beta_{21} & \beta_{22} & \beta_{23} \\ \beta_{31} & \beta_{32} & \beta_{33} \end{bmatrix}_1 \begin{bmatrix} r_1 \\ r_2 \\ r_3 \end{bmatrix}_1 = \begin{bmatrix} rhs_1 \\ rhs_2 \\ rhs_3 \end{bmatrix}_1 . \qquad (4.38)$$

Here, the rhs_is come from Eqs. (4.27), so it is clear that this system can be directly solved for $\boldsymbol{r}_1$ using $\mathcal{O}(N_v^3)$ arithmetic in the absence of any special structure in the matrix $[\boldsymbol{\beta}]_1$. The second row of (4.34) can be rearranged to the form

$$\begin{bmatrix} \beta_{11} & \beta_{12} & \beta_{13} \\ \beta_{21} & \beta_{22} & \beta_{23} \\ \beta_{31} & \beta_{32} & \beta_{33} \end{bmatrix}_2 \begin{bmatrix} r_1 \\ r_2 \\ r_3 \end{bmatrix}_2 = \begin{bmatrix} rhs_1 \\ rhs_2 \\ rhs_3 \end{bmatrix}_2 - \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} \\ \alpha_{31} & \alpha_{32} & \alpha_{33} \end{bmatrix}_2 \begin{bmatrix} r_1 \\ r_2 \\ r_3 \end{bmatrix}_1 , \qquad (4.39)$$

and it is clear from this that the entire solution vector $\boldsymbol{r}$ can be computed using $\mathcal{O}(N_v^3)$ arithmetic operations per grid point.

Once this vector has been computed we are prepared to complete the solution of Eq. (4.30) by solving Eq. (4.33). For this equation we start with the last row, but from the form of the upper triangular matrix $\boldsymbol{\mathcal{U}}$ we see that only the 3×3 identity matrix appears in the last block. This implies that $\delta\boldsymbol{Q}_{N_x}^T = \boldsymbol{r}_{N_x}^T$. It can also be readily seen that for the general i^{th} equation

$$\begin{bmatrix} \delta Q_1 \\ \delta Q_2 \\ \delta Q_3 \end{bmatrix}_i = \begin{bmatrix} r_1 \\ r_2 \\ r_3 \end{bmatrix}_i - \begin{bmatrix} \gamma_{11} & \gamma_{12} & \gamma_{13} \\ \gamma_{21} & \gamma_{22} & \gamma_{23} \\ \gamma_{31} & \gamma_{32} & \gamma_{33} \end{bmatrix}_i \begin{bmatrix} \delta Q_1 \\ \delta Q_2 \\ \delta Q_3 \end{bmatrix}_{i+1} . \qquad (4.40)$$

We see from this that the backward substitution step of this block tridiagonal procedure requires only $\mathcal{O}(N_v^2)$ arithmetic operations per grid point, and it can be carried out in a very straightforward way.

4.3.3 Arithmetic operation counts

We conclude this section on solution of block-banded tridiagonal systems with a more detailed accounting of arithmetic operations than we have given above, and from this we will be able to deduce in at least a qualitative way when it is better to employ block-banded coupled procedures rather than the sequential methods described in the preceding section.

We have already seen that construction of the $\boldsymbol{\mathcal{L}}$ and $\boldsymbol{\mathcal{U}}$ matrices requires $\mathcal{O}(N_v^3)$ arithmetic operations per grid point. We have carried this out in the context of a time-split solution method applied to a 2-D problem, implemented in a line-by-line solution mode. We assumed for x-direction solves that there would be N_x points per line; there are N_y lines, so the total arithmetic needed to construct the $\boldsymbol{\mathcal{L}}$ and $\boldsymbol{\mathcal{U}}$ matrices is $\mathcal{O}(N_v^3 N)$, where $N \equiv N_x N_y$. Most of this arithmetic is used to construct the matrices $[\boldsymbol{\beta}]_i$ and $[\boldsymbol{\gamma}]_i$ and to perform the forward substitution step requiring formal inversion of the $[\boldsymbol{\beta}]_i$s. Thus, it is not unreasonable to choose the constant in $\mathcal{O}$ to be approximately 25. This would correspond to four times $\mathcal{O}(N_v^3)$ for the block solves, three iterations per time step, and two split steps per iteration.

It is worthwhile to consider the alternative of simply applying Gaussian elimination to the entire unsplit system during each iteration. This would require $\mathcal{O}((N_v N)^3)$ total arithmetic, which is roughly a factor N^2 more than required for the block-tridiagonal method. On the other hand, we might also consider the sequential solution technique. This requires $\mathcal{O}(N_v N)$ arithmetic per iteration with a constant of approximately four—two split steps, and then the arithmetic required for scalar tridiagonal solves. This is to be compared with $\sim 6\,\mathcal{O}(N_v^3 N)$ per iteration for the block solves. Hence, the latter requires a factor $\sim 3\,\mathcal{O}(N_v^2)$ more arithmetic per iteration. At the same time, the sequential method can be expected to require signifcantly more iterations—typically a factor of between two and four. But what this simple analysis indicates is that one can afford a factor of approximately $\mathcal{O}(N_v^2)$ more iterations in the sequential approach before its total arithmetic will exceed that of the block solves. Moreover, the storage requirements are far lower for the sequential approach, first because storage for the solution procedure, itself, can be reused. But beyond this is the fact that the block-banded matrices $\boldsymbol{\mathcal{L}}$ and $\boldsymbol{\mathcal{U}}$ each contain a factor N_v^2 more elements than their sequential solution counterparts.

4.4 Cell-*Re* and Aliasing Treatments

When writing computer codes for solving problems containing both first- and second-order derivatives, one must be able to handle a form of non-physical solutions those arising from the so-called cell-*Re* problem, and also from aliasing. We will consider each of these in the current section. In the first subsection we will give a basic treatment of the cell-*Re* problem, indicating how it arises and some well-known remedies. In a second subsection we provide an introductory analysis of the more general problem of aliasing, the treatments of which generally also work for the cell-*Re* problem as well.

4.4.1 The cell-*Re* problem—its definition and treatment

In this subsection we begin by showing how the cell-*Re* problem arises as a consequence of specific properties of solutions to difference equations in general—not simply those associated with transport phenomena. This will provide some insight into how the problem might be treated, and we will introduce some examples of this. It will be seen that, contrary to what has sometimes been claimed, the problem does not arise simply from nonlinearities. Indeed, linear equations can, when discretized, exhibit the same kind of behavior as is seen in solutions to nonlinear equations. Thus, our analysis will be performed with linear model problems for which exact solutions can often be derived. As we have seen earlier, treatment of nonlinear problems essentially always involves linearization at each iteration. Furthermore, introduction of time dependence has little effect on the basic phenomenon involved, so we will restrict attention to the simpler linear, steady-state case.

A Burgers' Equation Model Problem

Burgers' equation has been widely used as a (usually) 1-D model of the incompressible (and, sometimes, compressible) momentum equations of the N.–S. system. The homogeneous Burgers' equation,

$$u_t + uu_x = \nu u_{xx}\,, \tag{4.41}$$

is one of the few nonlinear equations possessing exact solutions. Here, we will consider a significant simplification that will be adequate for our purposes. Namely, as noted above, we will treat the steady-state, linearized form

$$Uu_x - \nu u_{xx} = 0\,, \tag{4.42}$$

where U is an assigned constant.

This equation provides a 1-D model of the balance between advection and diffusion in a constant-velocity flow; but, more generally, it permits analysis of differential equations containing both first- and second-order derivative terms that may be very different in size. We now discretize Eq. (4.42) with centered differences to obtain

$$UD_{0,x}u_i - \nu D^2_{0,x}u_i = 0\,,$$

or

$$\frac{U}{2h}(u_{i+1} - u_{i-1}) - \frac{\nu}{h^2}(u_{i-1} - 2u_i + u_{i+1}) = 0\,,$$

where h is the discretization step size. Multiplication by h/U leaves this in the form

$$\frac{1}{2}(u_{i+1} - u_{i-1}) - \frac{\nu}{Uh}(u_{i-1} - 2u_i + u_{i+1}) = 0\,. \tag{4.43}$$

We now define the *cell Reynolds number* as

$$Re_h \equiv \frac{Uh}{\nu}\,, \tag{4.44}$$

analogous to the physical (dimensionless) Reynolds number, $Re \equiv UL/\nu$, appearing in various forms of the Navier–Stokes equations. It is of interest to note that unlike the corresponding physical counterpart,

Re_h may be negative; that is, U may take on either sign. Furthermore, we point out that if the governing equation is already cast in dimensionless form so that, *e.g.*, $\nu \to 1/Re$ as in the Navier–Stokes equations, then the cell Reynolds number is calculated as $Re_h = Re\, h$, where Re is the usual dimensionless Reynolds number, and h is a dimensionless (scaled with the same length used to define Re) grid spacing. Obviously, in this formulation $Re_h \ge 0$ must hold.

Some Basic Theory of Difference Equations

In this subsection we will describe, both heuristically and with fairly rigorous mathematics (the theoretical structure of solutions to difference equations), how the cell-Re problem arises. The mathematical treatment will suggest ways to construct (partial) remedies, but we will also see that there is no completely universal and affordable (from the standpoint of computer run times) way to deal with this problem.

The starting point toward understanding what the cell-Re problem is comes from a property of elliptic and parabolic differential equations known as a *maximum principle.* There are numerous of these, and the reader is referred to the various references on PDEs already cited in these lectures for detailed treatments. Here, we will provide just an heuristic description. Suppose $\mathcal{L}$ is a linear partial differential operator of elliptic type, and consider the problem

$$\mathcal{L}u = 0 \qquad \text{on} \quad \Omega \subseteq \mathbb{R}^d\,, \quad d = 1, 2, 3\,,$$

with boundary conditions

$$u(\boldsymbol{x}) = g(\boldsymbol{x})\,, \qquad \boldsymbol{x} \in \partial\Omega\,,$$

where $g(\boldsymbol{x})$ is a prescribed function. Then the maximum principle states that the maximum and minimum values taken on by $u \in \overline{\Omega}$ must occur on $\partial\Omega$. We remark that this principle is widely used in the theory of elliptic PDEs to demonstrate uniqueness of solutions without ever having to actually construct the solution.

For our purposes herein it is useful to note that a similar principle exists for difference approximations to elliptic equations. Moreover, it can be shown (see [16]) that the following two conditions are sufficient to guarantee satisfaction of such a maximum principle for discretizations of the general form

$$\mathcal{L}_h u_{\boldsymbol{i}} = a_{\boldsymbol{0}} u_{\boldsymbol{i}} - \sum_{\substack{\boldsymbol{\alpha}=\boldsymbol{s}^- \\ \boldsymbol{\alpha}\neq\boldsymbol{0}}}^{\boldsymbol{s}^+} a_{\boldsymbol{\alpha}} u_{\boldsymbol{i}+\boldsymbol{\alpha}} = 0\,.$$

i) <u>nonnegativity</u>: $a_{\boldsymbol{0}} > 0$ and $a_{\boldsymbol{\alpha}} \ge 0$, $\boldsymbol{\alpha} \in [\boldsymbol{s}^-, \boldsymbol{s}^+]$, the set of neighbors of $\boldsymbol{i}$, included in the mesh stencil of the discretization, where bold-faced quantities represent multi-indices with dimension consistent with the spatial dimension of the problem being considered.

ii) <u>diagonal dominance</u>: $\displaystyle\sum_{\substack{\boldsymbol{\alpha}=\boldsymbol{s}^- \\ \boldsymbol{\alpha}\neq\boldsymbol{0}}}^{\boldsymbol{s}^+} a_{\boldsymbol{\alpha}} \le a_{\boldsymbol{0}}$ $\forall$ $\boldsymbol{i}$ in the grid function index set, and with strict inequality for at least one $\boldsymbol{i}$.

We note that these requirements do not depend on the spatial dimension of the problem or on the specific form of differencing used to construct the approximation. In particular, they are sufficient even on unstructured grids. Moreover, in light of our brief discussion in Chap. 1 concerning the relationship between elliptic and parabolic operators, we would expect the above conditions to be correct also for discrete forms of parabolic equations.

We remark that the importance of the maximum principle in the present context is with regard to local, rather than global (over the whole spatial domain) behavior. In particular, if we consider placing an "internal boundary" around each mesh star, the maximum principle implies that the grid function value at the point (i, j) cannot exceed that of any of its (nearest) neighbors, and because there is a corresponding minimum principle, the grid-function value cannot be lower than that of any neighbor. In particular, it cannot oscillate from point to point.

As we implied at the beginning of this section, this initial treatment is intended to be heuristic. We will now take a more precise look at the cell-Re problem by studying the theoretical behavior of solutions to linear difference equations. In particular, we will solve Eq. (4.43) analytically and investigate the nature of the solution(s) as cell Re is varied.

If we use Eq. (4.44), the definition of Re_h, in (4.43) we obtain

$$\frac{Re_h}{2}(u_{i+1} - u_{i-1}) - (u_{i-1} - 2u_i + u_{i+1}) = 0\,,$$

or, after rearrangement,

$$-\left(1 + \frac{Re_h}{2}\right) u_{i-1} + 2u_i - \left(1 - \frac{Re_h}{2}\right) u_{i+1} = 0\,. \tag{4.45}$$

It is immediately clear from this form of the difference equation that $|Re_h| > 2$ will result in failure to satisfy the above non-negativity condition, which in turn means (possible) failure of the maximum principle and consequent possible appearance of local maxima and minima in the interior of the solution. We will now show in some detail how this occurs. In order to do this we will need to introduce some facts regarding linear difference equations.

Definition 4.1 *The* <u>*order*</u> *of a difference equation is the difference between the highest and lowest subscripts appearing on its terms.*

Thus, in the present 1-D case, Eq. (4.45), we see that the order is $(i + 1) - (i - 1) = 2$. (Note that in multi-dimensional problems, order must be defined in each separate "direction." We also have the following.

Theorem 4.1 *The number of independent solutions to a linear difference equation is equal to its order.*

The reader should recall that this result is completely analogous to that for differential equations. Further, we note that (linear) difference equations are solved (analytically) in a manner quite similar to what is done for differential equations, as we will now demonstrate for a simple example that is of the same form as Eq. (4.45). In particular, consider the general second-order linear, constant-coefficient difference equation

$$a_2 y_{i+1} + a_1 y_i + a_0 y_{i-1} = 0\,, \tag{4.46}$$

whose *characteristic equation* is

$$a_2 z^2 + a_1 z + a_0 = 0\,,$$

or

$$z^2 + \frac{a_1}{a_2} z + \frac{a_0}{a_2} = 0. \tag{4.47}$$

In the cases of second-, third- and fourth-order difference equations this characteristic equation can be solved exactly (in terms of elementary functions) since it is just a polynomial of degree equal to the order. In particular, the solutions to Eq. (4.47) are

$$z_\pm = \frac{-\dfrac{a_1}{a_2} \pm \sqrt{\left(\dfrac{a_1}{a_2}\right)^2 - 4\left(\dfrac{a_0}{a_2}\right)}}{2}\,, \tag{4.48}$$

and the corresponding independent solutions to Eq. (4.46) are

$$y_{i,\pm} = (z_\pm)^i\,, \qquad i = 1, \ldots, N\,, \tag{4.49}$$

with N being the number of discrete points in space, or discrete time indices, as appropriate.

The final piece of information needed for a sufficient understanding of solutions to difference equations is the following which, again, is analogous to the differential equation case.

Theorem 4.2 *The general solution to the difference equation* (4.46) *is a linear combination of the independent solutions given by* (4.49)*.*

Thus, we have

$$y_i = c_1 z_+^i + c_2 z_-^i\,, \qquad i = 1, 2, \ldots, N\,, \tag{4.50}$$

where c_1 and c_2 are constants to be determined from either initial or boundary conditions associated with a specific problem.

Before continuing, we remark that this level of information on difference equations can be found in most standard texts on numerical analysis, and further details can be found in more advanced treatments such as given in the monograph by Mickens [53].

The Cell-*Re* Problem

At this point it is time to focus on the cell-*Re* "problem," *per se.* In numerical computations involving transport equations the symptom of this problem is (non-physical) oscillation of the numerical solution from one grid point to the next, as displayed schematically in Fig. 4.1. It is clear from (4.49) and (4.50)

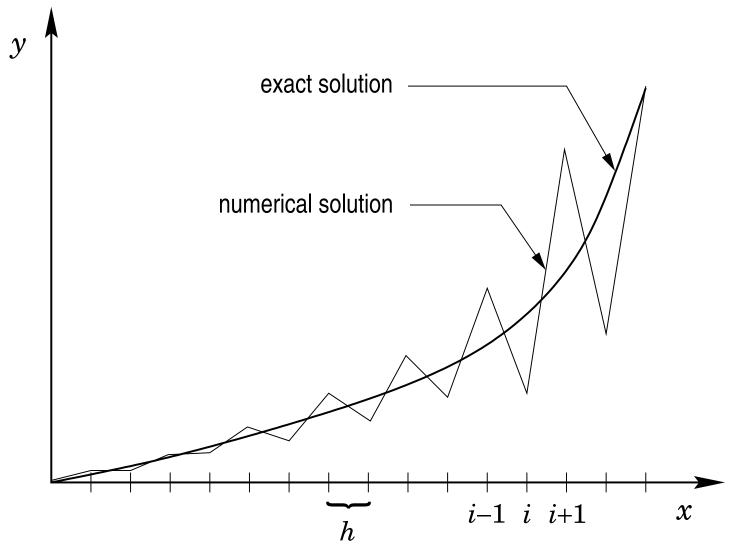

Figure 4.1: Grid point to grid point oscillations caused by cell-*Re* problem.

that if z_+ and z_- are not both nonnegative, it it possible for cell-*Re* "wiggles," as displayed in the figure, to occur. It should further be observed that the magnitude of these oscillations tends to increase with increasing solution magnitude, typically corresponding to increasing cell *Re*. In particular, such increases result in loss of the non-negativity and diagonal dominance properties of the difference equation required to guarantee satisfaction of a maximum principle.

As we have already indicated, our Burgers' equation model problem is in a form identical to that of Eq. (4.46), and it can be shown (an exercise for the reader) that solutions to the characteristic equation (4.47) take the form

$$z_+ = \frac{1 + \frac{1}{2}Re_h}{1 - \frac{1}{2}Re_h}\,, \qquad \text{and} \qquad z_- = 1\,.$$

We see from this that $z_+ \to 1$ as $Re_h \to 0$, and $z_+ \to -1$ as $Re_h \to \pm\infty$. Moreover, z_+ first becomes negative as $|Re_h|$ exceeds 2, and also $z_+ \to \pm\infty$ as $Re_h \to 2$ from below and above, respectively—in accord with the result we obtained from the earlier heuristic analysis based on the maximum principle. Thus, the well-known cell-*Re* restriction associated with centered-difference approximations can be derived purely from the mathematical theory of difference equations without invoking any physical arguments associated with transport equations such as, for example, appealing to flow direction to suggest inadequacy of centered approximations.

It is also worthwhile to plot values of z_+ as a function of Re_h. We note from the definition of Re_h that we can view such a plot as representing effects of changing grid spacing with velocity and viscosity fixed, of changing velocity with grid spacing and viscosity fixed, or of changing viscosity with velocity and grid spacing fixed. In actual physical problems any of these combinations might be appropriate (even within the same problem). Figure 4.2 displays the variation of the non-constant solution of Eq. (4.47) as a function of

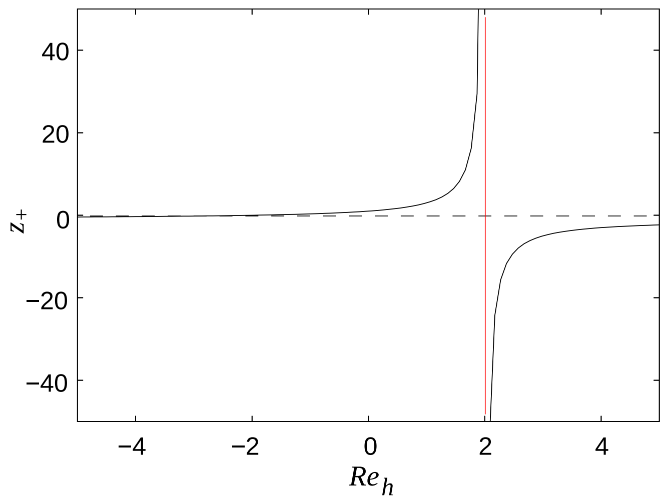

Figure 4.2: Dependence of z_+ on cell *Re*.

Re_h. There are several interesting points to be made with regard to this figure. First, we see, as is already evident from the analytical formula, that $z_+ < 0$ when $|Re_h| > 2$. Second, it is also clear that the range of cell Re over which the magnitude of z_+ is much greater than that of z_- is rather limited (essentially $1 < Re_h < 20$). This means that for certain combinations of boundary conditions the cell-*Re* problem might not be particularly significant even though the basic centered-difference approximation may have violated the discrete maximum principle. Moreover, we also see that the situation is, in general, much worse for $Re_h > 0$ than for $Re_h < 0$. Since U is the only factor in the definition of Re_h that can take on both signs, it is apparent, in the context of CFD, that the cell-*Re* problem is evidently less severe in reversed-flow situations than in high-speed forward flows, of course, modulo boundary-condition effects.

We now demonstrate for the model problem that the conditions needed to prevent oscillations in the exact solutions to the difference equation are, in fact, those that guarantee satisfaction of a maximum principle. We first note that along with the required nonnegativity of $z_\pm$, for physical problems we should also require $z_\pm \in \mathbb{R}$. This latter requirement implies that in Eq. (4.48)

$$\left(\frac{a_1}{a_2}\right)^2 \geq 4\left(\frac{a_0}{a_2}\right) ,$$

or

$$|a_1| \geq 2\sqrt{a_0 a_2} . \tag{4.51}$$

Now if we are to permit equality to hold in (4.51), the solution given by Eq. (4.48) can be positive only if

$$\text{sgn}(a_1 a_2) = -1 . \tag{4.52}$$

On the other hand, if strict inequality holds in (4.51), then we must require in addition to (4.52) that

$$\frac{a_1}{a_2} \geq \sqrt{\left(\frac{a_1}{a_2}\right)^2 - 4\left(\frac{a_0}{a_2}\right)} .$$

But this can be guaranteed only if

$$\mathrm{sgn}(a_0 a_2) = +1\,. \tag{4.53}$$

We note that conditions (4.52) and (4.53), taken together, are equivalent to the usual non-negativity requirement.

We will now show that satisfaction of the inequality (4.51) is equivalent to diagonal dominance. Again using (4.52) and (4.53), we see that we can assume $a_0, a_2 \geq 0$ and $a_1 < 0$ hold. In this case, diagonal dominance would imply

$$|a_1| \geq |a_0| + |a_2|\,.$$

Now since a_0 and a_2 are nonnegative, their square roots are real, and we have

$$(\sqrt{a_0} - \sqrt{a_2})^2 \geq 0 \quad \Rightarrow \quad a_0 - 2\sqrt{a_0 a_2} + a_2 \geq 0\,.$$

Thus,

$$2\sqrt{a_0 a_2} \;\leq\; a_0 + a_2 = |a_0| + |a_2|\,. \tag{4.54}$$

Now suppose diagonal dominance holds. Then

$$|a_1| \;\geq\; |a_0| + |a_2| \;\geq\; 2\sqrt{a_0 a_2}\,;$$

hence, inequality (4.51) is satisfied. On the other hand, suppose (4.51) does not hold; *i.e.*,

$$|a_1| \;<\; 2\sqrt{a_0 a_2}\,.$$

Then, from inequality (4.54) it follows that diagonal dominance does not hold, completing the equivalence proof.

Remedies for the Cell-*Re* Problem

In this subsection we will introduce several of the more widely-used treatments of the cell-*Re* problem. As we have hinted earlier, none of these provide a complete solution to the problem; moreover, from discussions in the preceding section, it would appear likely that there cannot be a single approach that is guaranteed to always work without causing other difficulties. We will analyze only one of these in detail and then provide a table summarizing properties of many of the most widely-used methods.

Through the years the cell-*Re* problem has received much attention in CFD, and many alterative treatments are available in most commercial CFD codes. To understand why this is so, recall the basic centered approximation of the Burgers' equation model problem, expressed here as

$$\left(1 + \frac{Re_h}{2}\right) u_{i-1} - 2u_i + \left(1 - \frac{Re_h}{2}\right) u_{i+1} = 0\,.$$

We again note that to prevent non-physical oscillatory solutions it is generally necessary to require

$$|Re_h| \leq 2\,, \tag{4.55}$$

which is termed the *cell-Re restriction* for centered-difference approximations. The consequence of this restriction is readily seen. Namely, suppose the flow velocity is $\mathcal{O}(1)$ and viscosity $\nu \sim \mathcal{O}(10^{-4})$, both of which are reasonable magnitudes for physical problems. Now from inequality (4.55) it follows that the grid spacing h must satisfy

$$h \leq \frac{2\nu}{|U|}\,.$$

Hence, in the present case we have $h \leq 2{\times}10^{-4}$, which in a 3-D calculation on the unit cube would require over 10^{11} grid points. Clearly, this is a nearly intractable problem on current computing hardware, and it will probably not be routinely possible to achieve the industrial standard of "overnight turnaround" for problems of this size any time in the next 10 years.

First-Order Upwinding. So-called *first-order upwind* differencing was probably the first proposed remedy for the cell-*Re* problem. It was introduced on the basis of *ad hoc* "physical reasoning" (see, *e.g.*, Patankar [54]). Namely, it was argued that the reason centered differencing led to non-physical solution oscillations was that it could sense both upstream and downstream flow behavior, and it combined these in its representation of the velocity gradient. It was further argued that a difference approximation using only information carried in the flow direction would be more appropriate (even if formally less acurate); *i.e.*, the difference approximations should be one-sided (or at least biased) into the flow direction. The simplest such procedure is the basic first-order difference.

For the Burgers' equation model problem with $U > 0$, this takes the form

$$UD_-u_i - \nu D_0^2 u_i = 0\,,$$

or after expanding the difference operator symbolism, introducing the definition of cell *Re* and rearranging the results,

$$\left(1 + \frac{1}{Re_h}\right)u_{i-1} - \left(1 + \frac{2}{Re_h}\right)u_i + \frac{1}{Re_h}u_{i+1} = 0\,.$$

It is easily checked (after multiplication by -1) that this difference equation satisfies both the non-negativity and diagonal dominance conditions needed to guarantee that no interior maxima or minima exist, independent of the value of Re_h, and it is also easy to directly solve this difference equation, as we have already done for the centered-difference case. The result of this is that both independent roots of the characteristic equation analogous to Eq. (4.47) are nonnegative for any value of $Re_h > 0$. This latter requirement is guaranteed to hold, in general, if we switch to a forward-difference approximation in cases where $U < 0$. It is left as an exercise for the reader to analyze this case; the results are essentially the same as those just given. Thus, we can summarize the first-order upwind discretization as follows, expressed here for non-constant U in non-conserved form:

$$U_i u_x|_i = \begin{cases} U_i D_- u_i\,, & U_i > 0\,, \\ U_i D_+ u_i\,, & U_i < 0\,. \end{cases} \tag{4.56}$$

There are at least two major difficulties with first-order upwind treatment of the cell-*Re* problem, and these were recognized fairly soon after its introduction. The treatment of this that we present here follows that of de Vahl Davis and Mallinson [55], who were among the first to analyze this approach in detail.

It is clear that the simple first-order differences employed in Eq. (4.56) degrade the overall accuracy of a discretization in any case that second-, or higher-, order methods are being used for terms other than advective terms, and in conjunction with this is the fact that this reduction in accuracy is global since, presumably, we are usually solving boundary value problems. But there are additional, more subtle, problems that arise from first-order upwind differencing of transport-like equations. To see this we consider the dimensionless form of the Burgers' equation model problem,

$$Uu_x - \frac{1}{Re}u_{xx} = 0\,, \tag{4.57}$$

with $U > 0$. Equation (4.56) implies that upwind differencing will employ the backward difference

$$D_-u_i = \frac{u_i - u_{i-1}}{h}$$

in the advective term, and expansion of u_{i-1} yields the following:

$$u_{i-1} = u_i - hu_x\big|_i + \frac{h^2}{2}u_{xx}\big|_i - \frac{h^3}{6}u_{xxx}\big|_i \pm \cdots\,.$$

This shows that Eq. (4.57) can be expressed as

$$Uu_x - \left(\frac{1}{Re} + \frac{Uh}{2}\right)u_{xx} + \frac{h^2}{6}Uu_{xxx} \pm \cdots = 0\,,$$

or

$$Uu_x - \frac{1}{Re^*}u_{xx} + \mathcal{O}(h^2) = 0\,, \tag{4.58}$$

where Re^* is the *effective Reynolds number* defined as

$$Re^* \equiv \frac{1}{\dfrac{1}{Re} + \dfrac{Uh}{2}}\,. \tag{4.59}$$

Equation (4.58) is often termed the "modified equation," and its significance is that it, rather than the original PDE, is solved to within <u>second-order</u> accuracy instead of first-order accuracy implied by the discretization. This suggests that numerical solutions are more accurate representations of solutions to modified equations than to solutions of the actual (in the case of the N.–S. equations, physical) equation(s). Thus, it is important to consider this modified equation in more detail.

What is rather obvious is that the only difference between physical and modified equations in the case of first-order upwinding is the coefficient of the diffusion term. That is, in the modified equation the original physical Reynolds number is replaced with the effective Reynolds number, Re^* given in Eq. (4.59), and it is essential to understand the consequences of this. Indeed, it has often been argued that for very high Re, the (physical) flow is convection dominated, and since upwinding imposes the "correct physics," use of this approximation should be quite accurate. But this is a spurious argument, especially when simulating wall-bounded flows. In such flows, no matter what the value of Re, some regions of the flow will be significantly affected by viscous forces; consequently, if anything is done to the equations of motion to alter this aspect of the physics (the diffusion terms), it cannot be expected that accurate solutions will be obtained.

With this in mind, it is of interest to examine the specific effects of $Re \to \infty$ in the modified equation. In particular, consider the limit of Re^* as $Re \to \infty$. It is easily seen from Eq. (4.59) that

$$\lim_{Re\to\infty} Re^* = \frac{2}{Uh}\,. \tag{4.60}$$

Hence, as the physical Reynolds number approaches infinity, the computational one approaches a finite limit—in fact, one that could be quite small if coarse grids are employed, or if $Re \to \infty$ due to increasing velocity (rather than increasing length scale, or decreasing viscosity. This provides an explanation for a phenomenon observed in early uses of first-order upwinding; namely, it was seen that solutions for a given physical situation and discrete approximation did not appear to change much with Re after the value of this parameter exceeded $\sim \mathcal{O}(10^3)$. Of course, in most situations this is counter intuitive on a physical basis, and it provided motivation for investigations of the sort we have just presented.

Clearly, the ultimate effect of first-order upwinding is to increase the coefficient of the diffusion term(s) in the governing equations. The factor producing this effect, $Uh/2$, (and similar factors arising in other methods) is often called *numerical diffusion* or *numerical viscosity.* Although these factors are related to, and in many ways similar to, "artificial viscosity," or "artificial dissipation," in the present lectures we will consider them to be distinct. In particular, numerical diffusion is, in general, an <u>unwanted</u> effect arising in an <u>uncontrolled</u> way from (usually) inadequate numerical analytic procedures—basically mistakes, whereas in essentially all cases artificial dissipation is added deliberately and at least to some extent, controllably, in a manner that has minimal effect on the physics represented by the final discrete equations. Furthermore, as is evident from (4.60), numerical diffusion can often completely dominate physical diffusion, and this is not usually the case with artificial dissipation except when it is employed with inviscid forms (the Euler equations) of the equations of motion.

The Hybrid Scheme. One of the early modifications that is still widely used today, especially as an option in commercial CFD software, is the so-called "hybrid" scheme of Spalding [**?**] (also, see [54]). In this approach the solution is tested locally to determine whether a centered difference cell-*Re* restriction violation is likely, and if it is the scheme is automatically switched to first-order upwinding. To construct this method for the model Burgers' equation problem, we now consider the conservation form and allow U to be the variable U_i, leading to

$$(Uu)_x - \nu u_{xx} = 0\,.$$

We first express the discrete equation in the general form valid for any 1-D three-point grid stencil,

$$a_1 u_{i-1} + a_2 u_i + a_3 u_{i+1} = 0\,,$$

and then define the difference equation coefficients as follows:

$$a_1 = \max\left[-U_{i-1}h, \nu - \frac{U_{i-1}h}{2}, 0\right]\,, \tag{4.61a}$$

$$a_2 = -a_1 - a_3 - h\left(U_{i+1} - U_{i-1}\right)\,, \tag{4.61b}$$

$$a_3 = \max\left[U_{i+1}h, \nu + \frac{U_{i+1}h}{2}, 0\right]\,. \tag{4.61c}$$

It is important to recognize that if use of the hybrid method results in switching to first-order upwinding at very many grid points, the solution behavior will be no different than that of first-order upwinding itself, and this is often the case. Thus, we do not recommend its use. We have included it here simply for the sake of completeness; as we have already mentioned, it is widely used in commercial CFD codes.

In summarizing this presentation on first-order upwind differencing, we again emphasize that, while it uniformly cures the cell-*Re* problem, its use results in a possibly drastic alteration of the physics being considered; the end result is that solutions obtained in this way cannot, in general, be trusted to accurately represent the physical phenomena being simulated. Thus, this approach should almost never be employed. But we note that there is one particular valid application for first-order upwind differencing. We will in Chap. 3 present a detailed algorithm for solving the N.–S. equations in which δ-form quasilinearization will be used to treat nonlinearities appearing in the momentum equations. In this particular instance it is quite convenient, and effective, to utilize first-order upwinding in constructing the left-hand side Jacobian matrix with respect to which the Newton iterations are computed. This is a valid approach provided the right-hand side of the equation is discretized in a physically-correct manner (*i.e.*, different from first-order upwinding) because approximate Jacobian matrices can be useful in Newton iteration procedures, and the one produced via first-order upwinding is numerically very stable.

In closing this section we provide in Table 4.1 a summary of the methods for treating the cell-*Re* problem that we have discussed in at least a cursory way. The table contains the independent roots of the difference equation and the cell-*Re* restriction for the Burgers' equation model problem, if any, for each procedure. Use of absolute-value symbols in all of the formulas except that for centered differencing arises from the fact that all other techniques involve a switch in formulas so as to always be differencing in the (mainly) flow direction. As a consequence, there are actually two possible difference equations in each such case; but their roots are the same if one expresses them in terms of $|Re_h|$.

Table 4.1 Difference equation roots and cell-*Re* restrictions for model problem

There are numerous other upwind-like treatments of the cell-*Re* problem. Some have at times been fairly widely used while most have been rather seldom employed. We do not intend to attempt an encyclopedic coverage of such methods, in large part because all have their shortcomings; and the ones discussed here are most often used. It is worth mentioning, however, that a related treatment has received considerable attention in recent years, and it is usually quite successful. It is to employ techniques originally derived for use in shock capturing associated with solving the Euler equations of compressible flow. There are

Scheme	Roots of Difference Equation	Cell-*Re* Restriction
1st-order upwind	$1+\lvert Re_h\rvert\,,\;1$	*none*
2nd-order upwind	$\frac{\left(1+\frac{3}{2}\lvert Re_h\rvert\right)\pm\sqrt{\left(1+\frac{3}{2}\lvert Re_h\rvert\right)^2-2\lvert Re_h\rvert}}{2}\,,\;1$	*none*
2nd-order centered	$\frac{1+\frac{1}{2}Re_h}{1-\frac{1}{2}Re_h}\,,\;1$	$\lvert Re_h\rvert \le 2$
QUICK	$\frac{\left(1+\frac{3}{4}\lvert Re_h\rvert\right)\pm\sqrt{\left(1+\frac{3}{4}\lvert Re_h\rvert\right)^2-\frac{1}{2}\lvert Re_h\rvert\left(1+\frac{3}{8}\lvert Re_h\rvert\right)}}{2\left(1+\frac{3}{8}\lvert Re_h\rvert\right)}\,,\;1$	$\lvert Re_h\rvert \le 8/3$

several classes of these methods, and we merely mention them here: flux-corrected transport (FCT), total variation diminishing (TVD) and essentially non-oscillatory (ENO). While use of such methods can be highly effective in terms of removing non-physical oscillations, they are in general somewhat more difficult to code, and they typically increase required arithmetic very significantly.

4.4.2 Treatment of effects of aliasing

In this section we will first indicate in fairly precise detail just what "aliasing" is, and how it arises. We will see that its symptoms are somewhat similar to those of the cell-*Re* problem treated above, but its root cause is rather different. We will then consider techniques for treating this problem. It will be seen that since aliasing arises from an inability of a numerical approximation to represent high-wavenumber components in a solution, the remedy must always be some form of smoothing, or mollification, that introduces additional dissipation into the numerical scheme, and thus damps effects of high wavenumbers. In the context of, *e.g.*, the Navier–Stokes equations, there have been three main approaches used to do this: *i*) flux modification, *ii*) artificial dissipation and *iii*) filtering. Here, we will emphasize only the last possibility. This approach has been utilized mainly in the context of meteorological flows, but Yang and McDonough [56] and McDonough and Yang [57] have recently demonstrated its effectiveness in simulations involving a Burgers' equation model of turbulence. Similar techniques are employed by Visbal and Gaitonde [58] and others.

How Aliasing Occurs

A straightforward, intuitive view of aliasing comes from the following simple plot, Fig. 4.3, in which we demonstrate an extreme case of under sampling of a signal. The basic signal is a sine function (plus its first subharmonic) with wavelength h, and it is being sampled at this same wavelength, presumably for later reconstruction. It is evident that the sampled signal is a (nonzero) constant. Moreover, it can also be seen that a Fourier analysis of the actual signal would show that its lowest mode (the average) has zero magnitude while the lowest (and only) mode of the sampled signal has magnitude near unity. We see from this that not only has the sampled signal lost the high-wavenumber content of the original signal, but it has also misrepresented even the lowest wavenumber, which in principle it should have been capable of representing. Thus, what we term aliasing has two main effects. It results in failure to represent high-wavenumber (and/or high-frequency) behavior, and (as it turns out, a consequence of this) it incorrectly represents contributions from lower wavenumbers (or frequencies).

The mathematics associated with this has been known for a long time, especially in the context of signal processing. Here, we provide a description taken from the text by Ames [50]. Let $f(x)$ be a function

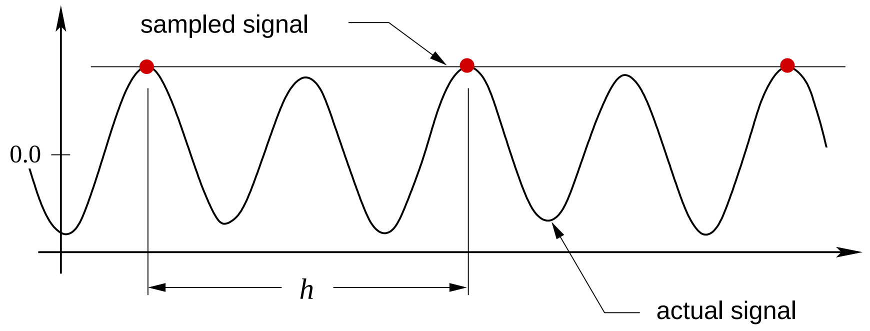

Figure 4.3: Under-sampled oscillatory function demonstrating effects of aliasing.

in $L^2(-1,1)$, and consider its Fourier representation given by

$$f(x) = \sum_{k=-\infty}^{\infty} a_k e^{ik\pi x}$$

with

$$a_k = \frac{1}{2}\int_{-1}^{1} f(x) e^{-ik\pi x}\, dx\,.$$

Now if one partitions the interval $[-1,1]$ with $2N$ uniformly-spaced points such that $x_j = j/N$, for $-N \le j < N$, one can construct the (exact) Fourier polynomial which when evaluated at $x = x_j$ gives the value

$$f_j = \sum_{m=-N}^{N-1} A_m e^{im\pi j/N}\,,$$

where

$$A_m = \frac{1}{2N}\sum_{j=-N}^{N-1} f_j e^{-im\pi j/N}\,. \tag{4.62}$$

The basic question at this point is, "How are the A_ms, obtained from the discrete approximation, related to the actual Fourier coefficients, the a_ks?" To find this relationship we evaluate $f(x)$ at the discrete point $x = x_j = j/N$:

$$f(x_j) = \sum_{k=-\infty}^{\infty} a_k e^{ik\pi x_j} = \sum_{k=-\infty}^{\infty} a_k e^{ik\pi j/N}\,.$$

We first observe that due to periodicity of the complex exponential there can be only $2N$ distinct values of $e^{ik\pi j/N}$. Then for any finite N we can rewrite the infinite sum as

$$\sum_{k=-\infty}^{\infty} a_k e^{ik\pi j/N} = \sum_{k=-\infty}^{\infty}\sum_{n=-N}^{N-1} a_{n+2Nk} e^{i(n+2Nk)\pi j/N}\,. \tag{4.63}$$

We now substitute the right-hand side of Eq. (4.63) (representing f_j) into Eq. (4.62) to obtain

$$\begin{aligned} A_m &= \frac{1}{2N}\sum_{j=-N}^{N-1}\sum_{k=-\infty}^{\infty}\sum_{n=-N}^{N-1} a_{n+2Nk} e^{i(n+2Nk)\pi j/N} e^{-im\pi j/N} \\ &= \sum_{k=-\infty}^{\infty} a_{m+2Nk} \\ &= a_m + \sum_{|k|>0} a_{m+2Nk}\,, \end{aligned} \tag{4.64}$$

with the second equality following from discrete orthonormality of the complex exponentials.

The first thing to notice regarding this result is that if it happens that only $2N$ Fourier coefficients of a signal are nonzero, the coefficients of the Fourier polynomial will agree with these (consistent with the exact polynomial construction). Similarly, if the function being represented by the Fourier series and Fourier polynomial is very smooth so that high-wavenumber coefficients in the Fourier series go to zero rapidly as wavenumbers become large, the Fourier polynomial coefficients will not differ greatly from the Fourier series coefficients. On the other hand, if the function f is not very smooth so that high-wavenumber coefficients have significant magnitude, and at the same time N is not sufficiently large, the difference between a_m and A_m will be significant because a_{m+2Nk} will still be large enough, even at very high values of k (and for all m, including $m = 0$), to contribute to the discrepancy. This is the fundamental cause of aliasing.

At this point one might question what any of this has to do with the finite-difference approximations that are the main topic of these lectures. But the answer to this is quite straightforward (and is easily seen to be analogous to a similar argument given earlier in Chap. 2 the context of restriction operators for multigrid methods). It is trivial to check that the relationship between the discrete function values f_j and the coefficients A_m of the Fourier polynomial in Eq. (4.61a) is a linear transformation consisting of a $2N \times 2N$ matrix. That is, given any set of $2N$ function values we can construct the $2N$ A_ms. Furthermore, the result of computing with a finite-difference or finite-volume method is a grid function consisting of a finite number of values that is supposed to approximate, in some sense, the exact solution to the PDE that has been discretized. We also expect that this exact solution possesses a Fourier series representation since in essentially all cases we will be working with solutions in the Hilbert space L^2—*i.e.*, in a function space corresponding to finite energy. Thus, our finite-difference approximation is, to within a (bounded) linear transformation, a Fourier polynomial intended to approximate the Fourier series of the exact solution. Hence, it is clear that if there are insufficient grid points in the representation, the effect will be identical to under sampling of a signal, and the result will be aliased. Conversely, a fully-resolved, fine-grid solution will not exhibit any effects of aliasing.

It is important to recognize that in the case of attempting to solve nonlinear equations, this situation is exacerbated by generation of high-wavenumber modes by the nonlinearities—even when they are not present in initial data—in the time evolving solutions of these equations. This combination makes the problem of aliasing very serious, and it is thus crucial to be able to treat it effectively.

Treatment of the Aliasing Problem

We have already mentioned the three main approaches currently employed to treat aliasing, and we note that independent of the details the end result must be an increase in dissipation in order for a treatment to be effective. As noted earlier, flux-modification schemes can provide a suitable approach despite the fact that they were originally designed to mitigate a rather different problem (Gibbs phenomenon oscillations in the presence of shocks), but one exhibiting similar symptoms. The same can be said for use of artificial dissipation. The main difference between these approaches is in the details of implementation. Flux modification is specifically numerical (*i.e.*, it is applied to the discrete equations), whereas introduction of artificial dissipation can be viewed also in an analytical context. Furthermore, flux-modification methods are relatively difficult to implement while addition of artificial dissipation terms is fairly straightforward, except possibly near boundaries. But it should also be mentioned that the end result of flux modification is introduction of additional truncation error of a diffusive nature. Thus, in principle, given a flux-modification scheme it is possible to find an equivalent artificial dissipation, although this is not generally easy. Finally, we note (as we have already hinted) that considerable programming effort and/or extra floating-point arithmetic is involved with implementing either of these general classes of methods, especially the former.

In the current lectures we will present a different treatment of the aliasing problem. This approach is not new; indeed, it is actually older than flux-modification and cell-Re treatments (that sometimes are mistakenly attempted). It is use of post-processing filters as is routinely done in essentially all signal processing. (After all, a collection of numbers comprising a discrete solution can easily be interpreted as a "signal.") Use of filters in the manner to be described here was probably first introduced by Shuman

[59] and analyzed in considerable detail by Shapiro [60]—but not in the context of CFD. Our description herein will be based on the modern treatment of PDEs as given, for example, by Gustafson [61].

We first introduce the notion of a solution operator and consider a discrete case of this. A *solution operator* is a mathematical construct that acts on initial data (prescribed at $t = t_0$) to produce a solution at a later time $t > t_0$. For example, if $u_0(x)$ provides initial data for some PDE and S is its solution operator, then we can formally express the solution at time t as

$$u(t) = S(t)u_0 .$$

It is useful for our purposes here to associate $S(t)$ with some form of (temporal) integration over the interval $(t_0, t]$, although this is not the only possible form.

Observe that discrete solution operators can be easily constructed via formal numerical integration of this form. Consider a nonlinear PDE with the representation

$$u_t = \mathcal{N}(u) ,$$

where $\mathcal{N}(u)$ represents some, generally, nonlinear (*e.g.*, Navier–Stokes) spatial operator. We can integrate this to obtain

$$u(t) = u(t_0) + \int_{t_0}^{t} \mathcal{N}(u)\, d\tau ,$$

and replacing the integral on the right-hand side with a quadrature will yield a discrete solution operator. For example, if we were to use simple forward-Euler time integration, we would obtain

$$u^{n+1} = u^n + k\mathcal{N}(u^n) ,$$

and we can express this formally as

$$u_h^{n+1} = S_h(k)u_h^n \tag{4.65}$$

to indicate that spatial discretization has been done with a grid of spacing h, and discrete time integration is being performed with time step k.

We next note that one of the tools of modern analytical PDE theory is use of *mollification*; this is just convolution of the PDE solution (or the PDE, itself, in the linear, constant-coefficient case) with a C_0^∞ function to smooth an otherwise not very regular solution to a point permitting classical analysis. Mollification is applied both to initial data, and to solutions as they evolve, by composition with the solution operator. We also observe that mollification is actually just a filtering process. In particular, mollified variables are constructed as

$$u_\epsilon(x) = \int_{-\epsilon}^{\epsilon} u(y)\delta_\epsilon(x-y)\, dy , \tag{4.66}$$

where δ_ϵ is a normalized C_0^∞ function with support $\rightarrow 0$ as $\epsilon \rightarrow 0$. This corresponds to a filter with kernel δ_ϵ. (The Gaussian filter that is widely used in large-eddy simulation of turbulent fluid flow is similar—but not identical–to this.) We will utilize these ideas here to construct a numerical mollification process that can be composed with discrete solution operators to provide additional dissipation needed to damp aliasing effects.

We define a function δ_h (which is not C^∞ but which does have compact support) as indicated in Fig. 4.4. Notice, in particular, that the support of δ_h is slightly greater than $2h$ about any particular point x_i, and that δ_h is constructed from two straight lines. It is easily seen from the figure that for any x_i

$$\delta_h(x_i - h) = \delta_h(x_i + h) = \frac{\xi}{h+\xi} \equiv \eta ,$$

where ξ is an adjustable parameter, and such that $\xi \ll h$ is assumed to hold. Next we define the discretely-mollified grid function

$$\widetilde{u}_h(x_i) = \int_{x_i-h}^{x_i+h} u_h(y)\delta_h(x_i - y)\, dy \equiv F(\delta_h)u_h . \tag{4.67}$$

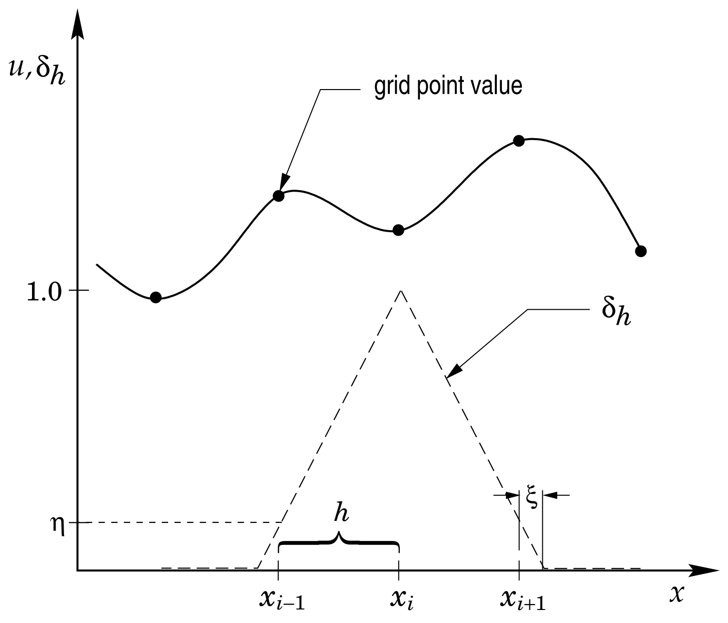

Figure 4.4: Schematic of a discrete mollifier.

We can now express the solution given in Eq. (4.64) in terms of the discrete solution operator as

$$u_h^{n+1} = S_h(k)F(\delta_h)u_h^n \,.$$

We observe that integration in (4.67) should formally be over the interval $[x_i - h - \xi, x_i + h + \xi]$, but there are no grid function values in the border regions of length ξ, so results (numerical) from the above formula would be unchanged by including this. We now perform trapezoidal quadrature between $x_i - h$ and x_i, and again between x_i and $x_i + h$. This results in

$$\begin{aligned}
\widetilde{u}_h(x_i) &= \frac{h}{2}\Big[2u_h(x_i)\delta_h(x_i - x_i) + u_h(x_{i-1})\delta_h(x_i - x_i + h) + u_h(x_{i+1})\delta_h(x_i - x_i - h)\Big] \\
&= \frac{h}{2}\Big[\eta u_h(x_{i-1}) + 2u_h(x_i) + \eta u_h(x_{i+1})\Big] \\
&= \frac{\eta h}{2}\left[u_h(x_{i-1}) + \frac{2}{\eta}u_h(x_i) + u_h(x_{i+1})\right] \,. \qquad (4.68)
\end{aligned}$$

But for proper normalization of this discrete case we must require $\widetilde{u}_h(x_i) = u_h(x_i)$ if $u_h \equiv \text{const}$. Thus, we must have

$$\widetilde{u}_h(x_i) = \frac{u_h(x_{i-1}) + \beta u_h(x_i) + u_h(x_{i+1})}{2 + \beta}\,, \qquad (4.69)$$

with $\beta \equiv 2/\eta$. This is the filter used by Shuman [59], and we call β the *filter parameter.*

It is clear that $\eta \to 0$ as $\xi \to 0$, and in this limit $\beta \to \infty$. This has the same effect as $\epsilon \to 0$ in the definition of mollifier given in Eq. (4.66) and would be equivalent if $h \to 0$ also holds. Furthermore, it is clear from Eq. (4.67), the definition of the discrete mollifier, that $\widetilde{u}_h(x_i) \to u_h(x_i)$ as $h \to 0$. While this is not immediately obvious from the final form of the Shuman filter given in Eq. (4.69), it can be demonstrated via a simple truncation error analysis which we now carry out as follows.

We first express (4.69) in the more concise notation

$$\widetilde{u}_i = \frac{u_{i-1} + \beta u_i + u_{i+1}}{2 + \beta}\,, \qquad (4.70)$$

and then expand u_{i-1} and u_{i+1} in Taylor series:

$$u_{i-1} = u_i - hu_x\Big|_i + \frac{h^2}{2}u_{xx}\Big|_i - \frac{h^3}{6}u_{xxx}\Big|_i \pm \cdots ,$$

and

$$u_{i+1} = u_i + hu_x\Big|_i + \frac{h^2}{2}u_{xx}\Big|_i + \frac{h^3}{6}u_{xxx}\Big|_i + \cdots .$$

(We remark here that existence of derivatives needed to construct these expansions is, in general, open to question, especially for the N.–S. equations, and it may be necessary to view these in the sense of distributions, as discussed in Chap. 1.) Then substitution of these into the above yields

$$\widetilde{u}_i = u_i + \frac{h^2}{2+\beta}u_{xx}\Big|_i + \mathcal{O}(h^4) .$$

This representation highlights two important features of this filter. First, it demonstrates that u_i is being replaced with a quantity explicitly containing added dissipation—one of the main requirements for successful treatment of aliasing. In particular, it can be seen that the dominant truncation error is diffusive, corresponding to addition of a scaled (discrete in this case) Laplacian. At the same time, the actual amount of added diffusion is controllable through the parameter β and the grid spacing h. Thus, even though a modified equation would contain extra diffusion at the level of the (physical) second-order operators, this goes to zero with h^2 rather than only as h in first-order upwinding. Moreover, it turns out that the $\mathcal{O}(h^4)$ term is anti-diffusive, leading to some cancellation of the effects at second order.

There are several additional items that should be investigated for the filter given in Eq. (4.70). These include the filter wavenumber response and time evolution of filtering errors as the filter is repeated applied. Here, we give only a brief treatment of these; the interested reader is encouraged to consult the more thorough analyses provided by McDonough [62]. The first of these is usually termed "frequency response" in the context of signal processing (in which the signals are typically functions of time), but we here more appropriately call this *wavenumber response* since we will usually apply the filter spatially. The wavenumber response shows the effect of the filter on the magnitude of the Fourier coefficients as a function of wavenumber. For the Shuman filter being treated here, this is given by

$$\widetilde{a}_m = \left[1 - \frac{2}{2+\beta}\left(1 - \cos mh\right)\right] a_m ,$$

with scaling such that if there are N grid points, $h = \pi/N$.

It is of interest to observe that as $h \to 0$ (or $N \to \infty$), $\widetilde{a}_m \to a_m \ \forall \ m$, independent of the value of the parameter β. At the same time, for fixed h, as $\beta \to \infty$ we obtain the same result. In addition, we can see the specific wavenumber response for fixed h, as a function of wavenumber, plotted in Fig. 4.5. From the figure we see that for small values of β the high-wavenumber content of the Fourier representation is almost completely removed—exactly what is needed to treat aliasing (recall Eq. (4.64)), and as β is increased less of the original Fourier coefficients is removed by the filter.

The next task is to determine, at least in a qualitative way, the effects of filtering on the numerical solution. The basic result we will present below corresponds to Burgers' equation, but unlike what was done in our previous analyses we have retained both time dependence and nonlinearity because these both are major aspects of the filtering problem. Details of the derivation are very tedious; they can be found in [62]. Here, we simply observe that to leading order (in spatial discretization step size) dominant filtering errors for both linear and nonlinear terms in a discrete solution operator based on centered spatial differencing and backward Euler temporal integration are of similar form and lead to a dominant error due filtering after n discrete time steps of

$$n\,\frac{h^2}{2+\beta}\,u^n_{xx} \, . \tag{4.71}$$

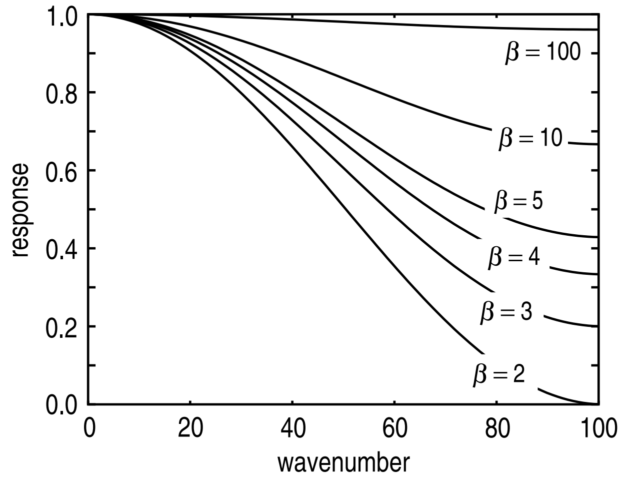

Figure 4.5: Wavenumber response of Shuman filter for various values of filter parameter.

It is clear from this that if the number of time steps becomes excessive, this term could potentially completely damp essentially all aspects of the computed solution; this is the major disadvantage in this approach. But it is seen from (4.71) that at least in principle this can be controlled with the parameter β, and it is generally found that this is adequate to guarantee robust behavior of this treatment of aliasing.

Now that it has been demonstrated that the Shuman filter is able to accomplish the desired goal, the remaining issue is the cost of the additional required arithmetic. It is well known that flux-modification approaches can easily double the required arithmetic, while use of artificial dissipation is considerably less expensive (and often less effective) if it can be implemented explicitly (which is not always the case). It is clear from Eq. (4.70) that in 1D the Shuman filter requires only two adds and two multiplies per grid point, per application (time step). The forward-Euler discrete solution operator uses five adds and five multiplies per grid point, per time step. Thus, even in the context of this very simple explicit solution operator, application of the filter increases total arithmetic by less than 50%, which is less than the increase in arithmetic incurred by use of typical (explicit) fourth-order artificial dissipation.

The final topic upon which we will touch briefly regarding use of filtering is the multi-dimensional case. All of our analyses have been one-dimensional, but we note that there is a formal, although *ad hoc*, extension to multi-dimensions, which in 2D is

$$\widetilde{u}_{i,j} = \frac{u_{i-1,j} + u_{i,j-1} + \beta u_{i,j} + u_{i,j+1} + u_{i+1,j}}{4+\beta} \,. \tag{4.72}$$

Moreover, an extension of this form to 3D is obvious. We note that all of the preceding development relies on uniform grid spacing in Cartesian coordinates. Little beyond preliminary studies has been completed for more complicated problem domains and non-uniform gridding. Moreover, a rigorous extension of the Shuman filter to higher dimensions is likely to result in a more complex grid stencil than the one corresponding to (4.72).

4.5 More Advanced Spatial Discretizations

Discretization of mixed derivatives and more general derivative operators in self-adjoint form is seldom treated in basic numerical analysis courses, and it is often absent from typical textbooks on tis topic. Some treatment is provided in the book by Mitchell and Griffiths [16], although much of what is presented by these authors seems rather impractical. In the late 1970s and early to mid 1980s considerable was devoted to this topic in the context of time-splitting methods similar to those studied in Chap. 3 of these notes.

Various works of Beam and Warming, *e.g.*, [48] are typical. The motivation for these studies was to produce accurate and efficient methods for treating mixed derivatives in generalized-coordinate formulations, as will be discussed in the next chapter of these notes, and where it will be seen that such transformed PDEs essentially always include mixed derivatives. In fact, they typically will contain the self-adjoint form of differential operators, so it is important to be able to accurately approximate such constructs. Fortunately, these approximations are not nearly so difficult to assemble as might be inferred from much of the extant literature.

In this section we will begin with a subsection devoted to approximation of simple miixed-derivative terms, and we follow this with a subsection in which we treat the more general case of mixed derivatives containing a variable coefficient. We then present a straightforward analysis of several alternatives to discretizing second-order operators in self-adjoint (sometimes referred to as "conserved") form. We remark that the last of these has application to operators associated with diffusive transport when diffusion coefficients are not constant, even in Cartesian coordinates.

4.5.1 Basic approximations for mixed derivatives

Our goal in this section is to provide simple, reasonably-accurate approximations to the differential operator $\partial^2/\partial y \partial x$ applied to a grid function $\{u_{i,j}\}$ at discrete locations on a uniform (in each separate direction) finite-difference grid, a small portion of which is displayed in Fig. 4.6. It should be clear that the simplest

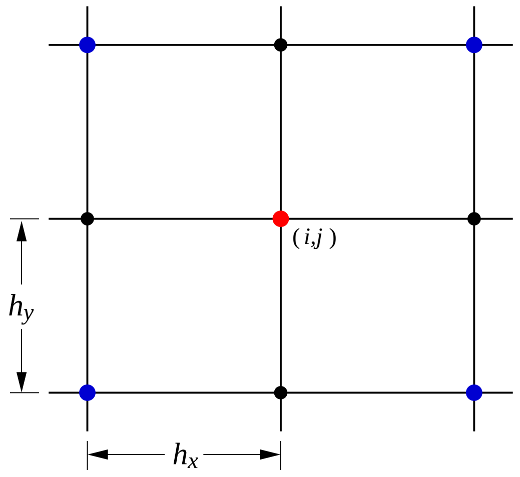

Figure 4.6: Finite-difference grid with uniform grid spacings $h_x \neq h_y$ in neighborhood of grid point (i, j).

approximation is given by

$$D_{+,y}D_{+,x}u_{i,j} = \left.\frac{\partial^2}{\partial x \partial y}\right|_{(x_i,y_j} + \mathcal{O}(h_x + h_y)\,.$$

We would intuitively expect this to be only first-order accurate, and we can prove this if $u \in C^{2,2}$ in a neighborhood of (x_i, y_j). To see this we first write the explicit form of the difference operators as

$$\begin{aligned} D_{+,y}D_{+,x}u_{i,j} &= D_{+,y}\left[D_{+,x}u_{i,j}\right] \\ &= D_{+,y}\left[\frac{1}{h_x}\big(u_{i+1,j} - u_{i,j}\big)\right] \\ &= \frac{1}{h_x h_y}\big(u_{i+1,j+1} - u_{i+1,j} - u_{i,j+1} + u_{i,j}\big)\,, \end{aligned} \tag{4.73}$$

and expand all of the grid function values about the point (x_i, y_j). In particular, we have

$$u_{i+1,j} = u_{i,j} + h_x u_x\big|_{(x_i,y_j)} + \frac{h_x^2}{2}u_{xx}\big|_{(x_i,y_j)} + \frac{h_x^3}{6}u_{xxx}\big|_{(x_i,y_j)} + \cdots\,,$$

$$u_{i,j+1} = u_{i,j} + h_y u_y\big|_{(x_i,y_j)} + \frac{h_y^2}{2} u_{yy}\big|_{(x_i,y_j)} + \frac{h_y^3}{6} u_{yyy}\big|_{(x_i,y_j)} + \cdots ,$$

$$u_{i+1,j+1} = u_{i,j} + h_x u_x\big|_{(x_i,y_j)} + h_y u_y\big|_{(x_i,y_j)} + \frac{1}{2}\left(h_x^2 u_{xx}\big|_{(x_i,y_j)} + 2h_x h_y u_{xy}\big|_{(x_i,y_j)} + h_y^2 u_{yy}\big|_{(x_i,y_j)}\right)$$
$$+\frac{1}{6}\left(h_x^3 u_{xxx}\big|_{(x_i,y_j)} + 3h_x^2 h_y u_{xxy}\big|_{(x_i,y_j)} + 3h_x h_y^2 u_{xyy}\big|_{(x_i,y_j)} + h_y^3 u_{yyy}\big|_{(x_i,y_j)}\right) + \cdots .$$

Then combining these as indicated in the right-hand side of Eq. (4.73) yields

$$u_{i+1,j+1} - u_{i+1,j} - u_{i,j+1} + u_{i,j} = h_x h_y u_{xy}\big|_{(x_i,y_j)} + \frac{1}{2}\left(h_x^2 h_y u_{xxy}\big|_{(x_i,y_j)} + h_x h_y^2 u_{xyy}\big|_{(x_i,y_j)}\right) + \mathcal{O}(h^4) ,$$

where h is a combination of h_x and h_y. Division by $h_x h_y$ then produces the desired result:

$$\frac{1}{h_x h_y}\big(u_{i+1,j+1} - u_{i+1,j} - u_{i,j+1} + u_{i,j}\big) = u_{xy}\big|_{(x_i,y_j)} + \frac{1}{2}\left(h_x u_{xxy}\big|_{(x_i,y_j)} + h_y u_{xyy}\big|_{(x_i,y_j)}\right) + \mathcal{O}(h^2) . \quad (4.74)$$

It is clear, in fact, that this same order of accuracy can be achieved in three other ways (using combinations of forward and backward differences) via the points immediately neighboring (i,j). Moreover, the average of these four results—(4.74) plus the three additional ones—yields

$$\frac{1}{4h_x h_y}\big(u_{i+1,j+1} - u_{i+1,j-1} - u_{i-1,j+1} + u_{i-1,j-1}\big) = u_{xy}\big|_{(x_i,y_j)} + \mathcal{O}\Big(h_x^2 + h_y^2\Big) , \quad (4.75)$$

as the ambitious reader can verify. Furthermore, this result is identical to the centered approximation

$$\begin{aligned} D_{0,y} D_{0,x} u_{i,j} &= D_{0,y}\left[D_{0,x} u_{i,j}\right] \\ &= D_{0,y}\left[\frac{1}{2h_x}\big(u_{i+1,j} - u_{i-1,j}\big)\right] \\ &= \frac{1}{4h_x h_y}\big(u_{i+1,j+1} - u_{i+1,j-1} - u_{i-1,j+1} + u_{i-1,j-1}\big) . \end{aligned} \quad (4.76)$$

We remark that this should not be a surprising result because one way to construct the centered approximation of a first derivative is to average forward and backward difference approximations. Beyond this we note that proof of second-order accuracy of Eq. (4.76), the centered approximation, is most easily done via the averaging process leading to Eq. (4.75).

4.5.2 Mixed derivatives with variable coefficients

As will be obvious in Chap. 5, second- (or higher-) order differential operators, when transformed to generalized coordinates, essentially always contain terms of the form

$$\frac{\partial}{\partial x}\left(a(x,y)\frac{\partial}{\partial y}\right) \quad \text{and/or} \quad \frac{\partial}{\partial y}\left(a(x,y)\frac{\partial}{\partial x}\right) . \quad (4.77)$$

Thus, it is important that we have methods available for treatingsuch cases. We remark that forms such as, *e.g.*,

$$a(x,y)\frac{\partial^2}{\partial y \partial x} ,$$

could also arise (although not usually from a coordinate transformation), but these are treated as we have already done in the preceding subsection provided values of the coefficient $a(x,y)$ are either known for all points at which the mixed-derivative term must be approximated, or they, themselves, can be approximated at all required points viia interpolation. Here, we consider only the form given in Eqs. (4.77).

We observe that the two mixed derivatives of (4.77) are, in general, not equal. But rather obviously, any form of approximation that works for one of them will also work for the other, *modulo* some changes in indexing. Hence, we will explicitly analyze only the first of these and leave a similar treatment of the second as an exercise for the reader. Similar to our approach for the constant-coefficient case, we begin with formal application of difference notation to obtain

$$\begin{aligned} D_{0,x}\Big(a(x_i,y_j)D_{0,y}u_{i,j}\Big) &= D_{0,x}\left[\frac{a_{i,j}}{2h_y}\big(u_{i,j+1}-u_{i,j-1}\big)\right] \\ &= \frac{1}{4h_xh_y}\Big[a_{i+1,j}\big(u_{i+1,j+1}-u_{i+1,j-1}\big)-a_{i-1,j}\big(u_{i-1,j+1}-u_{i-1,j-1}\big)\Big]. \end{aligned} \tag{4.78}$$

Note that the same grid function entries appear here as were present for the constant-coefficient mixed-derivative operator treated in the previous section. Also, recall that in that case we demonstrated second-order accuracy, and since in the present case $a(x,y)$ has merely been evaluated—and not approximated—we expect this still to hold. Nevertheless, this is not completely obvious, and it should be proven. In particular, the point at which the mixed derivative operator is being approximated is (x_i,y_j), but neither $u_{i,j}$ nor $a_{i,j}$ appear in the approximation (4.78). Hence, we must expand all of the grid function entries in this expression about the point (x_i,y_j).

We begin with the grid function corresponding to the variable coefficient $a(x,y)$. Assuming this function is reasonably smooth permits us to write the Taylor expansions

$$a_{i+1,j} = a_{i,j} + h_xa_x\big|_{(x_i,y_j)} + \frac{h_x^2}{2}a_{xx}\big|_{(x_i,y_j)} + \frac{h_x^3}{6}a_{xxx}\big|_{(x_i,y_j)} + \frac{h_x^4}{24}a_{xxxx}\big|_{(x_i,y_j)} + \cdots,$$

and

$$a_{i-1,j} = a_{i,j} - h_xa_x\big|_{(x_i,y_j)} + \frac{h_x^2}{2}a_{xx}\big|_{(x_i,y_j)} - \frac{h_x^3}{6}a_{xxx}\big|_{(x_i,y_j)} + \frac{h_x^4}{24}a_{xxxx}\big|_{(x_i,y_j)} + \cdots.$$

Expansions of the grid function values corresponding to an approximate solution $u_{i,j}$ take the following forms.

$$\begin{aligned} u_{i+1,j+1} = u_{i,j} &+ h_xu_x\big|_{(x_i,y_j)} + h_yu_y\big|_{(x_i,y_j)} + \frac{1}{2}\Big(h_x^2u_{xx}+2h_xh_yu_{xy}+h_y^2u_{yy}\Big)\Big|_{(x_i,y_j)} \\ &+\frac{1}{6}\Big(h_x^3u_{xxx}+3h_x^2h_yu_{xxy}+3h_xh_y^2u_{xyy}+h_y^3u_{yyy}\Big)\Big|_{(x_i,y_j)} \\ &+\frac{1}{24}\Big(h_x^4u_{xxxx}+4h_x^3h_yu_{xxxy}+6h_x^2h_y^2u_{xxyy}+4h_xh_y^3u_{xyyy}+h_y^4u_{yyyy}\Big)\Big|_{(x_i,y_j)} + \cdots, \end{aligned}$$

$$\begin{aligned} u_{i+1,j-1} = u_{i,j} &+ h_xu_x\big|_{(x_i,y_j)} - h_yu_y\big|_{(x_i,y_j)} + \frac{1}{2}\Big(h_x^2u_{xx}-2h_xh_yu_{xy}+h_y^2u_{yy}\Big)\Big|_{(x_i,y_j)} \\ &+\frac{1}{6}\Big(h_x^3u_{xxx}-3h_x^2h_yu_{xxy}+3h_xh_y^2u_{xyy}-h_y^3u_{yyy}\Big)\Big|_{(x_i,y_j)} \\ &+\frac{1}{24}\Big(h_x^4u_{xxxx}-4h_x^3h_yu_{xxxy}+6h_x^2h_y^2u_{xxyy}-4h_xh_y^3u_{xyyy}+h_y^4u_{yyyy}\Big)\Big|_{(x_i,y_j)} + \cdots, \end{aligned}$$

$$\begin{aligned} u_{i-1,j+1} = u_{i,j} &- h_xu_x\big|_{(x_i,y_j)} + h_yu_y\big|_{(x_i,y_j)} + \frac{1}{2}\Big(h_x^2u_{xx}-2h_xh_yu_{xy}+h_y^2u_{yy}\Big)\Big|_{(x_i,y_j)} \\ &-\frac{1}{6}\Big(h_x^3u_{xxx}-3h_x^2h_yu_{xxy}+3h_xh_y^2u_{xyy}-h_y^3u_{yyy}\Big)\Big|_{(x_i,y_j)} \\ &+\frac{1}{24}\Big(h_x^4u_{xxxx}-4h_x^3h_yu_{xxxy}+6h_x^2h_y^2u_{xxyy}-4h_xh_y^3u_{xyyy}+h_y^4u_{yyyy}\Big)\Big|_{(x_i,y_j)} + \cdots, \end{aligned}$$

$$\begin{aligned}u_{i-1,j-1} = u_{i,j} - h_x u_x\big|_{(x_i,y_j)} - h_y u_y\big|_{(x_i,y_j)} + \frac{1}{2}\Big(h_x^2 u_{xx} + 2h_xh_yu_{xy} + h_y^2u_{yy}\Big)\Big|_{(x_i,y_j)} \\ -\frac{1}{6}\Big(h_x^3u_{xxx} + 3h_x^2h_yu_{xxy} + 3h_xh_y^2u_{xyy} + h_y^3u_{yyy}\Big)\Big|_{(x_i,y_j)} \\ +\frac{1}{24}\Big(h_x^4u_{xxxx} + 4h_x^3h_yu_{xxxy} + 6h_x^2h_y^2u_{xxyy} + 4h_xh_y^3u_{xyyy} + h_y^4u_{yyyy}\Big)\Big|_{(x_i,y_j)} + \cdots .\end{aligned}$$

Substitution of these expansions into the right-hand side of Eq. (4.78) leads to

$$au_{xy} + a_xu_y + \left[\frac{1}{6}\Big(au_{xxxy} + a_{xxx}u_y\Big) + \frac{1}{2}\Big(a_xu_{xxy} + a_{xx}u_{xy}\Big)\right]h_x^2 + \frac{1}{6}\Big(au_{xyyy} + a_xu_{yyy}\Big)h_h^2 + \cdots , \quad (4.79)$$

where we have suppressed all grid-point indexing since all terms are evaluated at the same (x_i, y_j) point.

Now observe that the first two terms of this expression are what would result from analytically expanding $(a(x,y)u_y)_x$, so it follows that

$$D_{0,x}\Big(a(x_i,y_j)D_{0,y}u_{i,j}\Big) = \frac{\partial}{\partial x}\left(a(x,y)\frac{\partial u}{\partial y}\right)\bigg|_{(x_i,y_j)} + \mathcal{O}\Big(h_x^2 + h_y^2\Big) , \quad (4.80)$$

as we expected on intuitive grounds. But it is important to recognize that this formal second-order accuracy can be maintained only with considerable smoothness for both the variable coefficient $a(x,y)$ and the PDE solution $u(x,y)$. In particular, we must at least have $a \in C^3$ and $u \in C^{3,3}$; in fact, because $(au_y)_x \neq (au_x)_y$, and both of these mixed derivatives will often occur in the same equation, we must actually have $a \in C^{3,3}$.

We will see in Chap. 5 that functions analogous to $a(x,y)$ appear as "metric coefficients" in generalized-coordinate PDEs. It will also be seen that these are produce, usually, as grid functions during a (numerical) grid-generation process. As a consequence, they are intrinsically nonsmooth and, hence, must be mollified using techniques such as those discussed earlier in the present chapter if formal order of accuracy is to be preserved in computed results. We will consider this in more detail in Chap. 5.

We should point out that extensions of the preceding 2-D analyses to the three-dimensional case is, in principle, straightforward, but it is quite tedious. The same holds for analysis of higher-order mixed derivatives, even in 2D, for example, $\partial^3/\partial x^2\partial y$. The former of these is now very important because present-day computers are sufficiently powerful to permit solution of 3-D PDE problems. The latter occurs only infrequently, and we will not attempt to treat it here.

With regard to the 3-D casee we first note that, especially in the context of coordinate transformations, more different mixed derivatives will appear in the (transformed) PDEs. But these will all be second mixed derivatives, so the preceding analyses still hold; we simply need to be careful to not omit any such terms from the equations—and from any analysis of these.

Finally, we note that during the 1980s considerable effort was devoted to treatment of mixed derivatives in the context of time-split temporal dicretizations. In general, there are both stability and accuracy problems to be addressed. But beyond this is efficiency of the solution algorithm. Douglas and Gunn [47] had already in 1964 presented a version of their implicit split schemes which were accurate and stable but which already in 2D required approximately double the arithmetic needed to solve equations not containing mixed derivative—and the situation is worse in 3D. Such results motivated Beam and Warming, *e.g.*, [48] and others to develop new techniques in which the mixed derivative was handled explicitly while maintaining formal order of accuracy, without iteration (for linear problems). These methods were somewhat more complicated than the D–G approach, andd they were only conditionally stable. They are now seldom used. This is true at least in part because for nonlinear problems (such as, *e.g.*, the Navier–Stokes equations) one can simply move all mixed-derivative terms to the right-hand side of the equation(s), and iterate their coupling to the remaining differential operators during nonlinear iterations. Experience has shown that such procedures do not slow convergence rates, nor do they have a significant effect on stability of the time-stepping algorithm. Hence, this approach is now widely used; it is particularly effective when equations are cast in δ form.

4.5.3 Discretization of self-adjoint form second (unmixed) derivatives

In this subsection we will analyze truncation errors resulting from approximation of second-order (unmixed) derivatives in self-adjoint form, *i.e.*, in the form

$$\frac{\partial}{\partial x}\left(a(x)\frac{\partial u}{\partial x}\right) . \tag{4.81}$$

Here, we have used 1-D notation simply because the derivatives are unmixed, so any other dependent variables will behave as constants with respect to, in this case, differentiation with respect to x .

There are four separate cases of (4.81), but which occur in pairs. The first two correspond to $a(x)$ being a given analytical (and differentiable) function, with specific use being made of this. In this framework one can either choose to perform differentiations analytically and then difference any resulting derivatives of grid functions, or one can directly difference the self-adjoint from followed by use of exact evaluations of the variable coefficient. The second pair involves situations in which only discrete data are available for $a(x)$; *i.e.*, $a(x)$ is also a grid function, known only at the same locations as the u_is. In one of the two cases we still begin by formally carrying out required differentiations, but we replace these derivatives with difference approximations. In the final case we again employ direct discretization of the self-adjoint form, but now we can no longer exactly evaluate the coefficient $a(x)$. We remark that in the context of generalized coordinates to be treated in the next chapter, this is the usual situation.

Analytical $a(x)$

As indicated above, we begin here with the formula (4.81) and $a(x)$ a known, differentiable function, and we carry out the indicated differentiations. Thus, we obtain

$$\frac{\partial}{\partial x}\left(a(x)\frac{\partial u}{\partial x}\right) = a_x(x)u_x + a(x)u_{xx} , \tag{4.82}$$

with a_x being an analytically-derived derivative which can be exactly evaluated for any value of x (of course, within its domain of definition). We now employ centered approximations for u_x and u_{xx} leading to

$$\frac{\partial}{\partial x}\left(a(x)\frac{\partial u}{\partial x}\right)\bigg|_{x_i} = a_x\Big|_{x_i}\frac{u_{i+1}-u_{i-1}}{2h} + a\Big|_{x_i}\frac{u_{i+1}-2u_i+u_{i-1}}{h^2} - \frac{1}{6}\Big(a_x u_{xxx} + \frac{1}{2}a u_{xxxx}\Big)\Big|_{x_i} h^2 + \cdots , \tag{4.83}$$

where we have employed the Taylor expansions

$$u_{i+1} = u + hu_x + \frac{h^2}{2}u_{xx} + \frac{h^3}{6}u_{xxx} + \frac{h^4}{24}u_{xxxx} + \cdots , \tag{4.84a}$$

$$u_{i-1} = u - hu_x + \frac{h^2}{2}u_{xx} - \frac{h^3}{6}u_{xxx} + \frac{h^4}{24}u_{xxxx} + \cdots , \tag{4.84b}$$

where, again, we have suppressed indexing on the right-hand sides because all indices are the same.

An alternative to this is to difference the original self-adjoint form operator in a manner quite similar to that employed for mixed derivatives. In particular, we have

$$\begin{aligned} D_{0,x}\Big[a_i D_{0,x}u_i\Big] &= D_{0,x}\left[a_i\left(\frac{u_{i+1/2}-u_{i-1/2}}{h}\right)\right] \\ &= \frac{1}{h^2}\Big[a_{i+1/2}\big(u_{i+1}-u_i\big) - a_{i-1/2}\big(u_i - u_{i-1}\big)\Big] \\ &= \frac{1}{h^2}\Big[a_{i-1/2}u_{i-1} - \big(a_{i-1/2}+a_{i+1/2}\big)u_i + a_{i+1/2}u_{i+1}\Big] , \end{aligned} \tag{4.85}$$

where we suppose the a_is can be evaluated exactly, even at 1/2 indices. We can now use the Taylor expansions (4.84) given above to derive the truncation error for this expression. This leads to

$$D_{0,x}\Big[a_i D_{0,x}u_i\Big] = \frac{(a_{i+1/2}-a_{i-1/2})}{h}u_x + \frac{(a_{i+1/2}+a_{i-1/2})}{2}u_{xx} + \frac{(a_{i+1/2}-a_{i-1/2})}{h}\frac{h^2}{6}u_{xxx} + \frac{(a_{i+1/2}+a_{i-1/2})}{2}\frac{h^2}{12}u_{xxxx} + \cdots \,. \tag{4.86}$$

Next, we recall from Eq. (4.82) that the first two terms in this expression would have coefficients a_x and a, respectively, if it were exact, and in fact this would also be true for the next two terms. Thus, even though $a_{i-1/2}$ and $a_{i+1/2}$ are analytical evaluations of the coefficient function $a(x)$, their combinations that appear here are not exact representations of the actual coefficients of the differentiated form; *i.e.*, there is additional truncation error associated with these evaluations. To examine the error arising from this we need Taylor expansions of $a(x)$ similar to those used earlier in the mixed-derivative case. We have

$$a_{i+1/2} = a + \frac{h}{2}a_x + \frac{h^2}{8}a_{xx} + \frac{h^3}{48}a_{xxx} + \frac{h^4}{384}a_{xxxx} + \cdots \,,$$

and

$$a_{i+1/2} = a - \frac{h}{2}a_x + \frac{h^2}{8}a_{xx} - \frac{h^3}{48}a_{xxx} + \frac{h^4}{384}a_{xxxx} \mp \cdots \,.$$

Averaging these gives

$$\frac{(a_{i+1/2}+a_{i-1/2})}{2} = a + \frac{h^2}{8} + \cdots \,,$$

and differencing yields

$$\frac{(a_{i+1/2}-a_{i-1/2})}{h} = a_x + \frac{h^2}{24} + \cdots \,,$$

where the next term in either expression would be $\mathcal{O}(h^4)$, so we have neglected these.

We now substitute these expressions into the right-hand side of (4.86) to obtain

$$D_{0,x}\Big[a_i D_{0,x}u_i\Big] = a_x u_x + a u_{xx} \;+ \underbrace{\left(\frac{1}{6}a_x u_{xxx} + \frac{1}{12}a u_{xxxx}\right)h^2}_{\text{original error}} + \underbrace{\left(\frac{1}{24}a_{xxx}u_x + \frac{1}{8}a_{xx}u_{xx}\right)h^2}_{\text{additional error}} \,. \tag{4.87}$$

We see from this that differencing in self-adjoint form leads to two additional truncation error terms at $\mathcal{O}(h^2)$, even when exact evaluations of coefficients of the self-adjoint form are used.

$a(x)$ Known Only As Grid Function

There are again two subcases: we can formally differentiate first (even though $a(x)$ is a grid function), and then replace all derivatives with difference quotients, and we can also immediately apply the difference operators, as would be done in the self-adjoint case. We first carry out the formal differentiations.

We have

$$\begin{aligned}\frac{\partial}{\partial x}\left(a(x)\frac{\partial u}{\partial x}\right)\bigg|_{x_i} &= \left(a_x u_x + a u_{xx}\right)\Big|_{x_i} \\ &\simeq \frac{a_{i+1}-a_{i-1}}{2h}\left(\frac{u_{i+1}-u_{i-1}}{2h}\right) + a_i\left(\frac{u_)i-1-2u_i+u_{i-1}}{h^2}\right) \\ &= \frac{1}{4h^2}\big(a_{i+1}-a_{i-1}\big)\big(u_{i+1}-u_{i-1}\big) + \frac{a_i}{h^2}\big(u_{i-1}-2u_i+u_{i+1}\big)\,. \end{aligned} \tag{4.88}$$

At this point we need to determine the truncation error for this approximation. Obviously, since we have employed centered differences throughout, we would expect to obtain second-order accuracy; in fact, we will be able to rigorously demonstrate this when a and u are sufficiently smooth. To do this we first expand the grid-function values a_{i-1} and a_{i+1} in Taylor series:

$$a_{i+1} = a + h_x a_x + \frac{h^2}{2} a_{xx} + \frac{h^3}{6} a_{xxx} + \frac{h^4}{24} a_{xxxx} + \cdots\,, \tag{4.89a}$$

$$a_{i-1} = a - h_x a_x + \frac{h^2}{2} a_{xx} - \frac{h^3}{6} a_{xxx} + \frac{h^4}{24} a_{xxxx} \mp \cdots\,. \tag{4.89b}$$

Then we use these with Eqs. (4.84) to find that

$$u_{i+1} - u_{i-1} = 2hu_x + \frac{h^3}{3} u_{xxx} + \cdots\,,$$

$$a_{i+1} - a_{i-1} = 2ha_x + \frac{h^3}{3} a_{xxx} + \cdots\,,$$

$$u_{i+1} + u_{i-1} = 2u + h^2 u_{xx} + \frac{h^4}{12} u_{xxxx} \cdots\,,$$

and

$$u_{i-1} - 2u_i + u_{i+1} = h^2 u_{xx} + \frac{h^4}{12} u_{xxxx} + \cdots\,.$$

It then follows that

$$\begin{aligned}
\frac{\partial}{\partial x}\left(a(x) \frac{\partial u}{\partial x}\bigg|_{x_i} \right) &= D_{0,x}\Big[a_i D_{0,x} u_i \Big] + \mathcal{O}(h^2) \\
&= \frac{1}{4h^2}\big(a_{i+1} - a_{i-1}\big)\big(u_{i+1} - u_{i-1}\big) + \frac{a_i}{h^2}\big(u_{i-1} - 2u_i + u_{i+1}\big) \\
&= \big(a_x u_x + a u_{xx}\big)\Big|_{x_i} + \underbrace{\frac{1}{6}\Big(a_x u_{xxx} + \frac{1}{2} a u_{xxxx}\Big) h^2}_{\text{original error}} + \underbrace{\frac{1}{6} a_{xxx} u_x h^2}_{\text{additional error}} + \cdots\,.
\end{aligned} \tag{4.90}$$

We observe that there is one additional truncation error term at second order. This has arisen from having to approximate the derivative of $a(x)$.

The last case to consider involves direct differencing of the self-adjoint form. This leads to the following.

$$\begin{aligned}
\frac{\partial}{\partial x}\left(a(x) \frac{\partial u}{\partial x}\bigg|_{x_i} \right) &= D_{0,x}\Big[a_i D_{0,x} u_i \Big] + \mathcal{O}(h^2) \\
&= D_{0,x}\left[\frac{a_i}{h}\Big(u_{i+1/2} - u_{i-1/2}\Big) \right] \\
&= \frac{1}{h^2}\Big[a_{i+1/2}\Big(u_{i+1} - u_i\Big) - a_{i-1/2}\Big(u_i - u_{i-1}\Big) \Big] \\
&= \frac{1}{h^2}\Big[a_{i-1/2} u_{i-1} - \Big(a_{i-1/2} + a_{i+1/2}\Big) u_i + a_{i+1/2} u_{i+1} \Big]\,.
\end{aligned} \tag{4.91}$$

We observe that the second-order accuracy indicated in the first line of these expressions must be proven. But before doing this we must recall that in the present case $a(x)$ is a grid function known only at the full-index points at which $\{u_i\}$ is defined. Thus, our truncation error analysis must include effects of interpolating the half-index quantities to full-index locations.

To begin we express these half-index quantities in terms of the required full-index ones. It should be clear that on a uniform grid linear interpolation results in

$$a_{i-1/2} = \frac{1}{2}\Big(a_{i-1} + a_i\Big) + \mathcal{O}(h^2)\,, \qquad a_{i+1/2} = \frac{1}{2}\Big(a_{i+1} + a_i\Big) + \mathcal{O}(h^2)\,,$$

and

$$a_{i-1/2} + a_{i+1/2} = \frac{1}{2}\Big(a_{i-1} + 2a_i + a_i\Big) + \mathcal{O}(h^2)\,.$$

Then the discretization of Eq. (4.91) becomes

$$D_{0,x}\Big[a_i D_{0,x} u_i\Big] = \frac{1}{2h^2}\Big[\Big(a_{i-1} + a_i\Big)u_{i-1} - \Big(a_{i-1} + 2a_i + a_{i+1}\Big)u_i + \Big(a_{i+1} + a_i\Big)u_{i+1}\Big]\,. \tag{4.92}$$

We must be concerned at this point that use of only linear interpolation could result in $\mathcal{O}(1)$ errors. For example, consider the the first term on the right-hand side of (4.91):

$$\frac{1}{h^2}a_{i-1/2}u_{i-1} = \frac{1}{h^2}\Big[\frac{1}{2}\Big(a_{i-1} + a_i\Big) + \mathcal{O}(h^2)\Big]u_{i-1} \overset{?}{=} \frac{1}{2h^2}\Big(a_{i-1} + a_i\Big)u_{i-1} + \mathcal{O}(1)\,.$$

Obviously, this raises serious concern for this straightforward approach to constructing self-adjoint form approximations, and in the absence of detailed analyses as we present here one would be inclined to abandon such techniques. Indeed, this was initially done for it implies that $\{u_i\} \not\to u(x)$ as $h \to 0$. But we will prove that the combination of centered differencing and linear interpolation employed here with uniform grid spacing leads to second-order accuracy in the overall approximation.

We begin by using Eqs. (4.89) to show that

$$a_i + a_{i-1} = 2a - ha_x + \frac{h^2}{2}a_{xx} - \frac{h^3}{6}a_{xxx} + \frac{h^4}{24}a_{xxxx} \mp \cdots\,,$$

$$a_i + a_{i+1} = 2a + ha_x + \frac{h^2}{2}a_{xx} + \frac{h^3}{6}a_{xxx} + \frac{h^4}{24}a_{xxxx} + \cdots\,,$$

and

$$a_{i-1} + 2a_i + a_{i+1} = 4a + h^2 a_{xx} + \frac{h^4}{12} + \cdots\,.$$

Then use of these and Eqs. (4.84) in (4.92) leads to the result

$$\begin{aligned} D_{0,x}\Big[a_i D_{0,x} u_i\Big] = \big(a_x u_x + a u_{xx}\big)\Big|_{x_i} &+ \underbrace{\frac{1}{6}\Big(a_x u_{xxx} + \frac{1}{2}a u_{xxxx}\Big)h^2}_{\text{original error}} \\ &+ \underbrace{\frac{1}{2}\Big(\frac{1}{2}a_{xx}u_{xx} + \frac{1}{3}a_{xxx}u_x\Big)h^2}_{\text{additional error}}\,. \end{aligned} \tag{4.93}$$

This is, of course, the desired result. It shows that very straightforward differencing and interpolation methods can be used while still maintaining second-order accuracy. It is also of interest to note that these results show that, in general, differencing in self-adjoint conservative form is formally less accurate, contrary to a long-held myth in computational fluid dynamics.

4.6 Treatment of Advanced Boundary Conditions

Implementation of boundary conditions (BCs) is nearly always the most difficult part of PDE (and, even ODE) code constructions, at least if any degree of generality is to be attained. But despite this, the usual linear BCs, Dirichlet, Neumann and Robin, are relatively easy to treat; and very little is typically devoted to this aspect of numerically solving differential equations (but see Keller [72] or McDonough [1]). In fact, nonlinear versions of these can be handled fairly directly via δ-form quasilinearization, as we briefly indicated earlier in these notes. There are nevertheless boundary conditions that are much more difficult to treat, and some of these will comprise the topics of the current section. In particular, we will consider two main boundary condition types, one of which further generalizes previous linear BC analyses, and the other, which be either linear or nonlinear, may itself require additional PDE solutions.

We will present these in two subsections. The first will involve mainly (but not exclusively) linear "general" BCs that include so-called "mixed" conditions (an example of which is a periodicity condition) and yet more general BCs containing both normal and tangential derivative terms. The latter arises in generalized coordinates when a Neumann or Robin condition was imposed in the original Cartesian coordinate system. The second class of BCs to be considered is that of "outflow" or "flow-through" conditions and/or "conditions at infinity." A subsection will be devoted to each of these two main classes of BCs.

4.6.1 General linear BCs

As noted above, we will treat two main types of BCs in this context. The first of these are properly called *mixed conditions* and in one space dimension are distinguished by the fact that they contain information from <u>both</u> ends of the interval of definition of the problem. They take the form, *e.g.*,

$$a(0,y)u(0,y) + b(0,y)u_x(0,y) + c(1,y)u(1,y) + d(1,y)u_x(1,y) = f(x,y)\,, \tag{4.94}$$

where $x = 0$ or $x = 1$ might hold on the right-hand side of (4.94).

The second type of condition to be considered in this section is of the form

$$u(0,y) + a(0,y)u_x(0,y) + b(0,y)u_y(0,y) = g(0,y)\,. \tag{4.95}$$

Here, we have rescaled the equation with respect to any possible non-unity coefficient of the first term on the left-hand side. We remark that, in principle, these two types of conditions can be combined.

Mixed conditions

Mixed BCs of the form (4.94) are usually first encountered in a first (graduate) course in ODE boundary-value problems (BVPs). The theory of such conditions has been completely understood for nearly a century, but they are seldom treated in numerical analysis courses. This is true, in part, because they occur rather infrequently. But there is one very important exception to this—namely, as mentioned above, periodicity conditions of the form, *e.g.*,

$$u(0) = u(1)\,,$$

in the simplest case.

The first important item to note regarding mixed conditions is that even though a single boundary condition contains information from <u>both</u> ends of the problem domain, such a single condition is not generally sufficient to produce a well-posed BVP—but it is sufficient in the case of first-order ODEs and PDEs. Indeed, for second-order equations, such those mainly considered in these lectures, it is necessary to supply <u>two</u> such BCs. We remark, furthermore, that it is often no longer useful to associate each of these with a particular end of the domain because, in general, each will contain information from both ends. An exception to this arises either when one of the two BCs is not of mixed type, or when the inhomogeneities indicated in Eqs. (4.94) and (4.95) are evaluated at a specific end of the domain. In these cases it is clearly possible to distinguish the end of the domain at which the boundary condition should be applied.

A second item concerns treatment of the differential equation, itself, at boundaries where mixed conditions are imposed. In general, what one notices, even in the simplest case of a periodicity condition, is that the solution is never specified on either boundary. This, in turn, implies that the differential equation <u>must</u> also be solved at any such boundary. From this we see that in cases involving mixed BCs on both ends of a problem domain there must be a total of four discrete equations: two corresponding to boundary condition approximations, and two representing the differential equation. The latter two, of course, will be specifically associated with a particular end of the problem domain just as would be the case for unmixed conditions of Neumann or Robin type.

Conditions containing tangential derivative

4.6.2 Outflow BCs

"Outflow" boundary conditions have been a topic of concern in numerical solution of PDEs (and in ODE two-point boundary-value problems) from the beginning of attempts at their solution. Outflow (and analogous inflow) conditions can arise in numerous contexts, but except for the somewhat special case of "non-reflecting" BCs, they involve problems posed on infinite (or, at least, semi-infinite) domains. These, of course, cannot be treated without modification on finite computational hardware. The non-reflecting case, on the other hand, may involve boundaries at finite locations; but, similar to the outflow case, it is desired that these boundaries have <u>no</u> effect on the interior solution away from the boundaries. From the standpoint of numerical implementation, the outflow case corresponds to setting the location of an "artificial" boundary whose sole purpose is to aid machine computation and, at the same time, prescribing BCs on this boundary will either have minimal effect on the interior solution, or will have the "correct" effect.

Here, we will treat several different approaches to address this problem. The first, and simplest, is to extend the problem domain to permit application of standard BCs. The second is to implement a non-reflecting condition at any required boundaries. The final technique to be treated here is use of auxiliary conditions derived to be (mathematically) consistent with the given specific problem.

Extension of computational domain

It is often the case for infinite-domain (and semi-infinite-domain) problems that solutions at infinity are not actually needed (or may be known—often, trivially) despite the fact that the problem is posed on an infinite domain. A particular example of this is flow of fluid through a pipe of finite length that is open at the outflow end. It is well known from fluid mechanics that the pressure can typically be specified, but velocity of the fluid exiting into (probably) some other fluid is not known at the pipe (but it is almost certainly zero at infinity). Hence, an outflow condition is needed at the pipe exit. Again, from fluid mechanics, it is known that if the pipe is "long enough" details of the flow will cease to change in the flow direction, and in this case outflow conditions are easy to deduce; namely, for any dependent variable, say u, we must satisfy the homogeneous Neumann condition

$$\frac{\partial u}{\partial x} = 0\,,$$

where we have taken x to be the direction of flow. Here, we will henceforth assume the domain is not sufficiently long to permit use of this condition, but in addition, that extending the domain a length which would permit the above to hold introduces an unacceptable computational burden.

To treat this problem we assume there is some domain length beyond that of the problem specification which still would be computationally feasible. Then the basic question is, "How can we solve the PDE in this extended—but still too short—domain and still apply the simple homogeneous Neumann condition at the outflow boundary?" Obviously, if the domain is too short, the solution will be such that

$$\frac{\partial u}{\partial x} \neq 0\,,$$

at the location of the extended boundary. Hence, forcing the homogeneous condition to hold (*i.e.*, assigning it) will alter the computed solution throughout the domain, at least to some extent. Moreover, the shorter the domain extension, the worse will be the computed results.

What is required is a <u>smooth</u> alteration of the computed solution in the extended domain beyond the length of the specified problem in such a way that $\partial u/\partial x = 0$ actually holds at the extended boundary, and at the same time such that numerical solutions computed beyond the end of the original domain still are consistent with the PDE and discretization on the original domain.

An easy way to achieve this is to recursively apply a filter (*e.g.*, Eq. (4.69) such as analyzed earlier in this chapter) to the solution in the extended domain. It is well known (and easily demonstrated) that if sufficient recursions are performed, the solution becomes constant; hence, $\partial u/\partial x = 0$ is satisfied. There are numerous details involved with this approach, and despite its general ease of application, one must choose filter parameters and number of recursive iterations in addition to the number of grid points by which to extend the original domain. Moreover, as one might expect, results are clearly problem dependent, and grid spacing dependent, in addition to being influenced by the distribution of filter parameters and/or number of recursive applications, as a function of location within the extended domain.

Nevertheless, this approach is conceptually simple and can be quite effective. Its main advantages are that the PDEs are satisfied to within numerical error at the specified problem boundaries (hence, a numerically-correct, but not necessarily accurate, solution) and the solution modifications imposed thereafter in the extended domain are such that associated errors can be controlled. The disadvantage of this procedure is the need for numerical experimentation to set various required parameter values.

Non-reflecting boundary conditions

Derivation and use of auxiliary conditions

4.7 Summary

Chapter 5

Numerical Solution of Partial Differential Equations on Irregular Domains—Grid Generation

Solution of PDEs on irregular domains essentially always requires use of what is termed "grid generation," so it is important to first note just what is meant by this term. *Grid generation* is the determination of the locations of discrete points (selected from a space-time continuum) at which solutions to discrete equations will be sought, and assigning to these points an index system permitting an ordering with respect to one another in the context of the discretization applied to the original continuum problem. At least in a formal sense, grid generation must be carried out for every possible numerical PDE problem, no matter how simple; *i.e.*, some indexing system must be established for the grid points even in essentially trivial 1-D cases.

There are currently two distinct classes of grid generation techniques in wide use, leading to two corresponding types of grids: *i*) structured and *ii*) unstructured. In addition, these are often used in combination with such grids being called *hybrid*. Within each of these classes there are several different types of grids, and gridding strategies, that must be considered to completely define the overall grid-generation problem. The main ones of these are block, chimera and adaptive—and their various combinations. Finally, we observe that within recent years a different (but not altogether new) approach has begun to be frequently used. As is true with structured and unstructured grids this actually consists of a variety of individual, but related, methods variously termed immersed boundary, embedded boundary and immersed interface techniques.

We begin this chapter with a few details providing comparisons between the two main classes of grids presently in wide use for numerically solving PDE problems—namely, structured and unstructured grids. It will be clear from discussions of the next section that there is only one main advantage, and many disadvantages, associated with the latter—despite its current popularity. In a subsequent section we then present a detailed treatment of structured grids. This will include such topics as the transformations that must be applied to differential and boundary operators to achieve the required "generalized-coordinate" formulation, the differential geometry associated with such transformations, and details of how these are implemented in practical PDE solution methods. In a fourth section we will present an alternative treatment that is beginning to see considerable increases in use, especially for problems containing a wide range of length scales, *viz.*, the immersed boundary technique.

5.1 Overview of Grid Structures

As noted above, there are two main general classes of grids now in wide use for numerical solution of PDEs—structured and unstructured grids, or meshes, as they are often termed, especially in the former case. In this section we will provide an overview, and thus a comparison, of these two gridding approaches without giving much detail of either. Our main goal here is to elucidate the advantages and disadvantages of these two approaches to grid generation to allow the reader to make informed decisions regarding which method to apply for any particular PDE problem.

Figure **??** presents some sketches to aid in understanding the following discussions. Part (a) of this figure shows a structured grid for a fairly general (but nearly convex) 2-D domain and the associated

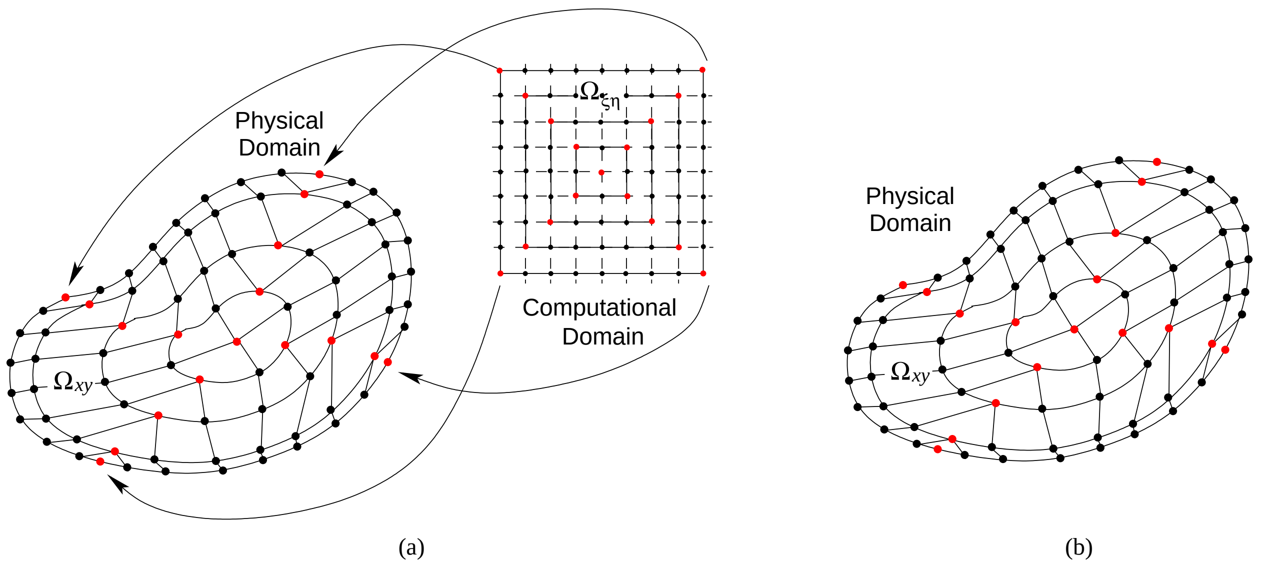

Figure 5.1: Geometry of (a) structured and (b) unstructured grids.

"computational" grid. Part (b) displays a corresponding unstructured grid for the same domain. We first note that generation of the unstructured grid is relatively simple. One has only to select points at which the approximate solution to the PDE(s) is to be computed, and then (nearly) arbitrarily label these points (and, of course, store the grid-point coordinates and labels). The PDE(s) and any associated boundary conditions remain unaltered in form. In particular, Cartesian coordinates can be used quite generally, so the mathematical formulation remains straightforward—but the numerical analysis becomes more complicated. On the other hand, as suggested by Fig. **??**(a), the structured-grid approach requires a transformation of the PDE(s) and boundary conditions, resulting in a generally more complicated mathematical structure, but one that is more easily treated with standard numerical methods.

In the following subsections we will treat both of these cases and then briefly consider how to handle moving/adaptive grids.

5.1.1 Unstructured grids

Use of unstructured grids is a relatively recent development in computational PDEs. It is now widely used in commercial software, especially for computational fluid dydnamics (CFD), and is also appearing in other areas, *e.g.*, computational electromagnetics. The main, and nearly only, advantage of an unstructured grid is its ease of generation. Such grid generation algorithms are almost entirely based on well-understood techniques from computer science. They require little intervention on the part of the "user;" that is, they are (nearly) automatic. Of course, as noted in [6], it is not yet possible to produce completely automated methods for grid generation, but in this regard generation of unstructured grids is far simpler than that of their structured counterparts.

It is essentially only this aspect of unstructured gridding that leads to its wide use. In particular, as CFD became more heavily used in US industry it was recognized that typically as much as 80% of the human time required to solve any given fluid dynamics problem was spent in generating the (structured) grid. This was considered to be far too disproportionate with respect to other (perceived to be) more important aspects of such problems—*e.g.*, solution validation and analysis and interpretation of results. At about this same time computer scientists were making significant advances in surface rendering and image reconstruction, and it was recognized that some of these techniques might be applied to grid generation.

An additional, but in some respects not so clear-cut, advantage of the unstructured grid is that, as mentioned above, the PDEs to be solved remain in their original coordinate system. Hence, they are, in principle, usually easier to solve—at least on a conceptual basis. In particular, no derivation or processing of metric information (see below) is necessary during the numerical solution procedure, and this, in turn, reduces the required arithmetic. On the other hand, the basic discretizations of the PDEs will now be more complicated. Points at which this must be done are distributed in an irregular, nonuniform way leading to far more complicated derivative approximation formulas for any desired formal order of accuracy. Moreover, the data used in such constructions must be retrieved from memory with use of pointers. The first of these leads to increases in arithmetic, while the second results in computationally inefficient data structures. Thus, even though the original PDE is simpler than in the structured-grid case, it is unclear that anything has been gained in terms of overall arithmetic.

In general, in 2D a total of the point and its five nearest-neighbor point grid-function values will be needed to produce a second-order accurate first derivative approximation. This is because at least a second-degree polynomial approximation of the function (which is third-order accurate) will be needed. In Cartesian coordinates this function approximation would take the form

$$u(x,y) = ax^2 + bxy + cy^2 + dx + ey + f + \mathcal{O}(h_x^3 + h_y^3) \tag{5.1}$$

on a rectangular grid. Such an approximation can be easily constructed via exact interpolation polynomial methods. Then u_x, for example, is given by

$$u_x(x,y) = 2ax + by + d + \mathcal{O}(h_x^2 + h_y^3)\,. \tag{5.2}$$

Observe that, in principle, only three coefficients (and hence, three grid-point values) are needed to construct this approximation—which is formally the same as needed for one-sided approximations on uniform structured grids. But it is clear that these coefficients contain information from other grid points via the construction needed to produce Eq. (5.1), implying that construction of (5.1) followed by (analytical) differentiationis needed to maintain accuracy.

We remark here that the exact interpolation polynomial works equally well in the context of unstructured grids, and although derivative approximations employed, especially in CFD, are different in detail from that indicated here, their results must be equivalent because of uniqueness of interpolation polynomials. It is also clear from Eqs. (5.1) and (5.2) that if second-order derivative approximations are to be constructed with irregularly-spaced grid points as occur with unstructured grids, then more than six points and corresponding grid-function values will be needed. To illustrate this we consider the case of using only six points and demonstrate that this is only first-order accurate. We carry this out for a simply 1-D case consisting only of nonuniform grid spacing, and in place of the exact interpolation polynomial construction discussed above, we employ Taylor expansions. (Note that once these are truncated, they are actually Taylor polynomials—up to a controllable truncation error—so uniqueness of interpolation polynomials implies equivalence of results with those obtained by other methods.) Figure 5.2 provides a graphical depiction of the situation being considered. We begin with Taylor expansions using the two different step sizes h_{i-1}

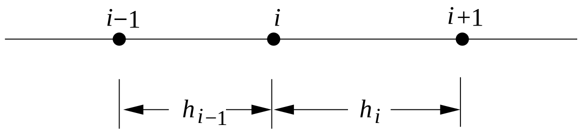

Figure 5.2: Simple 1-D geometry for nonuniform-grid construction of second derivative approximations.

and h_i:

$$u_{i-1} = u_i - h_{i-1}u_{x,i} + \frac{h_{i-1}^2}{2}u_{xx,i} - \frac{h_{i-1}^3}{6}u_{xxx,i} + \frac{h_{i-1}^4}{24}u_{xxxx,i} - \ + \cdots\,,$$

and

$$u_{i+1} = u_i + h_i u_{x,i} + \frac{h_i^2}{2}u_{xx,i} + \frac{h_i^3}{6}u_{xxx,i} + \frac{h_i^4}{24}u_{xxxx,i} \ + \cdots\,.$$

We add these two expansions, and then solve for the term containing $u_{xx,i}$, the quantity we wish to approximate, to obtain

$$\frac{1}{2}\left(h_i^2 + h_{i-1}^2\right)u_{xx,i} = u_{i-1} + u_{i+1} - 2u_i - \big(h_i - h_{i-1}\big)u_{x,i} + \mathcal{O}(h^3)\,,$$

where $h \equiv \max(h_{i-1}, h_i)$. It is easy to see that if grid spacing were uniform so that $h_{i-1} = h_i$, then the usual centered approximation would be recovered, and in this case the $\mathcal{O}(h^3)$ would become $\mathcal{O}(h^4)$. But for nonuniform gridding we need an approximation to $u_{x,i}$ to complete the construction.

It is easy to show for nonuniform grid spacing that a second-order accurate approximation to $u_{x,i}$ is given as

$$u_{x,i} = \frac{1}{h_i + Rh_{i-1}}\big[u_{i+1} - Ru_{i-1} + (R-1)u_i\big]\,, \tag{5.3}$$

with R defined as

$$R \equiv \left(\frac{h_i}{h_{i-1}}\right)^2\,. \tag{5.4}$$

Then the above expression for $u_{x,i}$ becomes

$$u_{xx,i} = \frac{2}{h_i^2 + h_{i-1}^2}\left\{u_{i-1} - 2u_i + u_{i+1} - \frac{h_i^2 - h_{i-1}^2}{h_i + Rh_{i-1}}\Big[u_{i+1} - Ru_{i-1} + (R-1)u_i\Big]\right\} + \mathcal{O}(h)\,. \tag{5.5}$$

In contrast, we will see in the next section that on a structured grid only the grid-point value and those of its two nearest neighbors are needed to obtain second-order accurate approximations of second derivatives.

It is also worthwhile to note that derivative approximations similar to those obtained from Eqs. (5.1) and (5.2) had already been investigated at least as early as the mid 1960s, although in a slightly different context (see [16]), and at that time they were not considered to be very favorable. In fact, the transformation functions that we consider next in the context of structured grids were viewed as being much more useful.

Finally, we observe from Fig. 5.1(b) that when unstructured grids are employed, the meshes needed to solve systems of equations arising in any implicit method will, in general, not be amenable to use of highly-efficient time splitting methods such as those of Chap. 3. Instead, techniques based on the elliptic solvers of Chap. 2 must be used. For time-dependent problems this will result in significant increases in arithmetic far beyond that needed for evaluation of metric coefficients that occur on structured grids.

5.1.2 Structured grids

References

[1] J. M. McDonough. *Lectures in Basic Computational Numerical Analysis*, 2007; available from the website http://www.engr.uky.edu/∼acfd

[2] E. Isaacson and H. B. Keller. *Analysis of Numerical Methods.* John Wiley & Sons, Inc., New York, NY, 1966 (now available as the Dover paperback first published in 1994).

[3] T. M. Apostol. *Mathematical Analysis.* Addison-Wesley Pub. Co., Reading, MA, second edition, 1974.

[4] R. D. Richtmyer and K. W. Morton. *Difference Methods for Initial-Value Problems.* John Wiley & Sons, Inc., New York, NY, second edition, 1967.

[5] P. W. Berg and J. L. McGregor. *Elementary Parial Differential Equations.* Holden-Day, San Francisco, 1966.

[6] J. F. Thompson, B. K. Soni and N. P. Weatherill. *Handbook of Grid Generation.* CRC Press, Boca Raton, 1999.

[7] G. Strang and G. J. Fix. *An Analysis of the Finite Element Method.* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1973.

[8] B. L. Buzbee. A Capacitance Matrix Technique. in *Sparse Matrix Computations*, J. R. Bunch and D. J. Rose (eds), Academic Press, Inc., New York, 1976.

[9] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Methods for Sparse Matrices*, Clarendon Press, Oxford, 1986.

[10] C. A. J. Fletcher. *Computational Techniques for Fluid Dynamics, Vols. I and II*, Springer-Verlag, Berlin, 1988.

[11] K. Umeshankar and A. Taflove. *Computational Electromagnetics*, Artech House, Boston, 1993.

[12] Y. Saad. *Iterative Methods for Sparse Linear Systems*, PWS Pub. Co., Boston, 1996.

[13] O. Axelsson. Solution of Linear Systems of Equations: Iterative Methods, in *Sparse Matrix Techniques*, Lecture Notes in Mathematics 572, Springer-Verlag, Berlin, 1977.

[14] J. Stoer and R. Bulirsch. *Introduction to Numerical Analysis*, Springer-Verlag, New York, 1980.

[15] D. M. Young. *Iterative Solution of Large Linear Systems*, Academic Press, New York, 1971.

[16] A. R. Mitchell and D. F. Griffiths. *The Finite Difference Method in Partial Differential Equations*, John Wiley & Sons, Inc., Chichester, 1980.

[17] R. S. Varga. *Matrix Iterative Analysis*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1962.

[18] L. A. Hageman and D. M. Young. *Applied Iterative Methods*, Academic Press, New York, 1981.

[19] G. Birkhoff and R. E. Lynch. *Numerical Solution of Elliptic Problems.* SIAM, Philadelphia, 1984.

[20] E. H. Cuthill and R. S. Varga. A Method of Normalized Block Iteration, *J. Assoc. Comput. Mach.* **6**, 236–244, 1959.

[21] S. V. Parter. *Numer. Math.* **1**, 240–252, 1959.

[22] D. W. Peaceman and H. H. Rachford, Jr. The Numerical Solution of Parabolic and Elliptic Differential Equations, *J. Soc. Indust. Appl. Math.* **3**, 28–41, 1955.

[23] W. Hackbusch. *Iterative Solution of Large Sparse Systems of Equations*, Springer-Verlag, New York, 1994.

[24] O. Axelsson. *Iterative Solution Methods*, Cambridge University Press, Cambridge, 1994.

[25] E. L. Wachspress. Optimum Alternating-Direction-Implicit Iteration Parameters for a Model Problem, *J. Soc. Indust. Appl. Math.* **10**, 339–350, 1962.

[26] T. A. Manteuffel. The Tchebychev Iteration for Non-Symmetric Linear Systems, *Numer. Math.* **28**, 307–327, 1977.

[27] H. A. Van der Voorst. A Preconditioned Tchebychev Iterative Solution Method for Certain Large Sparse Linear Systems with a Non-Symmetric Matrix, in *Numerical Integration of Differential Equations and Large Linear Systems*, J. Hinze (ed), Lecture Notes in Mathematics 968, Springer-Verlag, Berlin, 1982.

[28] H. L. Stone. Iterative Solution of Implicit Approximations of Multidimensional Partial Differential Equations, *SIAM J. Numer. Anal.* **5**, 530–558, 1968.

[29] S. G. Rubin and P. K. Khosla. Navier–Stokes Calculations with a Coupled Strongly Implicit Method—I Finite Difference Solutions, *Computers and Fluids* **9**, 163–180, 1981.

[30] P. K. Khosla and S. G. Rubin. A Diagonally Dominant Second Order Accurate Implicit Scheme, *Computers and Fluids* **2**, 207–209, 1974.

[31] G. E. Schneider and M. Zedan. A Modified Strongly Implicit Procedure for the Numerical Solution of Field Problems, *Numer. Heat Transf.* **4**, 1–19, 1981.

[32] M. R. Hestenes and E. L. Stiefel. Methods of Conjugate Gradients for Solving Linear Systems, *Nat. Bur. Std. J. Res.* **49**, 409–436, 1952.

[33] R. P. Fedorenko. A Relaxation Method for Solving Elliptic Difference Equations, *USSR Comput. Math. and Math. Phys.* **1**, 1092–1096, 1961.

[34] N. S. Bachvalov. On the Convergence of a Relaxation Method with Natural Constraints on the Elliptic Operator, *USSR Comput. Math. and Math. Phys.* **6**, 101–135, 1966.

[35] A. Brandt. Multi-Level Adaptive Solutions to Boundary-Value Problems, *Numer. Math.* **31**, 333–390, 1977.

[36] J. H. Bramble, J. E. Pasciak, J. Wang and J. Xu. Convergence Estimates for Multigrid Algorithms without Regularity Assumptions, *Math. Comp.* **57**, 23–45, 1991.

[37] K. Stüben and U. Trottenberg. Multigrid Methods: Fundamental Algorithms, Model Problem Analysis and Applications, in *Multi-Grid Methods*, Hackbusch and Trottenberg (eds), Lecture Notes in Mathematics 960, Springer-Verlag, Berlin, 1982.

[38] W. Briggs. *A Multigrid Tutorial*, SIAM, Philadelphia, 1987.

[39] K. Böhmer and H. J. Stetter. *Defect Correction Methods Theory and Applications*, Springer-Verlag, Wien, 1984.

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

[41] H. A. Schwarz. in *Vierteljahrsschrift Naturforsch. Ges. Zürich* **15**, 272–286, 1870.

[42] A. R. Mitchell. *Computational Methods in Partial Differential Equations*, John Wiley & Sons, London, 1969.

[43] B. Smith, P. Bjørstad and W. Gropp. *Domain Decomposition, Parallel Multilevel Methods for Elliptic Partial Differential Equations*, Cambridge University Press, Cambridge, 1996.

[44] A. W. Schueller and J. M. McDonough. A Multilevel, Parallel, Domain Decomposition, Finite-Difference Poisson Solver, to appear in *Proceedings of Parallel CFD 2001*, North-Holland Elsevier, Amsterdam, 2002.

[45] J. Douglas and H. H. Rachford. On the Numerical Solution of Heat Conduction Problems in Two and Three Space Variables, *Trans. Amer. Math. Soc.* **82**, 421–439, 1956.

[46] N. N. Yanenko. *The Method of Fractional Steps*, Springer-Verlag, Berlin, 1971.

[47] J. Douglas, Jr. and J. E. Gunn. A general formulation of alternating direction methods, part I. parabolic and hyperbolic problems, *Numer. Math.* **6** 428–453, 1964.

[48] R. M. Beam and R. F. Warming. Alternating Direction Implicit Methods for Parabolic Equations with a Mixed Derivative, *SIAM J. Sci. Stat. Comput.* (now *SIAM J. Sci. Comput.*) **1** 131–159, 1980.

[49] L. V. Kantorovich and G. P. Akilov. *Functional Analysis* Second Edition, Pergamon Press, Oxford, 1982.

[50] W. F. Ames. *Numerical Methods for Partial Differential Equations* Second Edition, Academic Press, New York, 1977.

[51] R. E. Bellman and R. E. Kalaba *Quasilinearization and Nonlinear Boundary-Value Problems*, American Elsevier Publishing Co., Inc., New York, 1965.

[52] D. A. Anderson, J. C. Tannehill and R. H. Pletcher. *Computational Fluid Mechanics and Heat Transfer*, Hemisphere Publishing Co., New York, 1984.

[53] R. E. Mickens. *Difference Equations*, Van Nostrand Rheinhold Co., New York, 1987.

[54] S. V. Patankar. *Numerical Heat Transfer and Fluid Flow*, McGraw-Hill Book Co., New York, 1980.

[55] G. de Vahl Davis and G. D. Mallinson. An evaluation of upwind and central difference approximations by a study of recirculating flow, *Computers and Fluids* **4**, 29–43, 1976.

[56] T. Yang and J. M. McDonough. Solution filtering technique for solving Burgers' equation, accepted for special issue of *Discrete and Continuous Dynamical Systems*, 2003.

[57] J. M. McDonough and T. Yang. Solution filtering technique for solving turbulence problems, in preparation for submission to *Int. J. Comput. Fluid Dyn.* , 2008.

[58] M. R. Visbal and D. V. Gaitonde. On the Use of Higher-Order Finite-Difference Schemes on Curvilinear and Deforming Meshes, *J. Comput. Phys.* **181**, 155–185, 2002.

[59] F. G. Shuman. Numerical method in weather prediction: smoothing and filtering, *Mon. Weath. Rev.* **85**, 357–361, 1957.

[60] R. Shapiro. Smoothing, filtering and boundary effects, *Rev. Geophys. and Space Phys.* **8**, 359–387, 1970.

[61] K. E. Gustafson *Introduction to Partial Differential Equations and Hilbert Space Methods*, John Wiley & Sons, New York, 1980.

[62] J. M. McDonough Lectures in Computational Fluid Dynamics of Incompressible Flow: Mathematics, Algorithms and Implementations, 2007. Available as a downloadable PDF file at http://www.engr.uky.edu/∼acfd

[63] G. Dahlquist. *A Special Stability Problem for Linear Multistep Methods.* BIT 3, New York, 1963.

[64] P. J. Davis and P. Rabinowitz. *Methods of Numerical Integration.* Academic Press, New York, NY, 1975.

[65] C. deBoor. *A Practical Guide to Splines.* V. A. Barker (ed.), Springer-Verlag, New York, NY, 1978.

[66] G. Forsythe and C. B. Moler. *Computer Solution of Linear Algebraic Systems.* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1967.

[67] C. W. Gear. *Numerical Initial Value Problems in Ordinary Differential Equations.* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1971.

[68] P. Henrici. *Elements of Numerical Analysis.* John Wiley & Sons, New York, 1964.

[69] R. W. Hornbeck. *Numerical Methods.* Quantum Publishers, Inc., New York, NY, 1975.

[70] C. B. Moler J. J. Dongarra, J. R. Bunch and G. W. Stewart. *LINPACK User's Guide.* SIAM, Philadelphia, PA, 1979.

[71] L. W. Johnson and R. D. Riess. *Numerical Analysis.* Addison-Wesley Pub. Co., Reading, MA, second edition, 1982.

[72] H. B. Keller. *Numerical Methods for Two-Point Boundary-Value Problems.* Dover Pub., Inc., New York, NY, 1992.

[73] H. Kreiss and J. Oliger. *Methods for the Approximate Solution of Time Dependent Problems.* GARP Pub, 1973.

[74] L. Lapidus and J. H. Seinfield. *Numerical Solution of Ordinary Differential Equations.* Academic Press, New York, NY, 1971.

[75] P. D. Lax and B. Wendroff. *Systems of Conservation Laws*, volume 13. Can. Pure Appl. Math., 1960.

[76] A. M. Ostrowski. *Solution of Equations and Systems of Equations.* Academic Press, New York, second edition, 1966.

[77] A. Ruhe. "Computation of Eigenvalues and Eigenvectors" *in Sparse Matrix Techniques.* V. A. Barker (ed.), Springer-Verlag, Berlin, 1977.

[78] J. C. Strikwerda. *Finite Difference Schemes and Partial Differential Equations.* Wadsworth & Brooks/Cole, Pacific Groove, CA, 1989.

[79] A. H. Stroud. *Approximate Calculation of Multiple Integrals.* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1971.

[80] L. H. Thomas. "Elliptic Problems in Linear Difference Equations over a Network," *Watson Sci. Comput. Lab. Rept.*, Columbia University, NY, 1949.

[81] J. F. Traub. *Iterative Methods for the Solution of Equations.* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1964.

[82] J. H. Wilkinson. *The Algeibraic Eigenvalue Problem.* University Press, London, 1965.

[83] J. H. Wilkinson and C. Reinsch. *Linear Algebra.* Springer-Verlag, New York, NY, 1971.