Multigrid Techniques

Books in the Classics in Applied Mathematics series are monographs and textbooks declared out of print by their original publishers, though they are of continued importance and interest to the mathematical community. SIAM publishes this series to ensure that the information presented in these texts is not lost to today's students and researchers.

Classics in Applied Mathematics

C. C. Lin and L. A. Segel, *Mathematics Applied to Deterministic Problems in the Natural Sciences*

Johan G. F. Belinfante and Bernard Kolman, *A Survey of Lie Groups and Lie Algebras with Applications and Computational Methods*

James M. Ortega, *Numerical Analysis: A Second Course*

Anthony V. Fiacco and Garth P. McCormick, *Nonlinear Programming: Sequential Unconstrained Minimization Techniques*

F. H. Clarke, *Optimization and Nonsmooth Analysis*

George F. Carrier and Carl E. Pearson, *Ordinary Differential Equations*

Leo Breiman, *Probability*

R. Bellman and G. M. Wing, *An Introduction to Invariant Imbedding*

Abraham Berman and Robert J. Plemmons, *Nonnegative Matrices in the Mathematical Sciences*

Olvi L. Mangasarian, *Nonlinear Programming*

*Carl Friedrich Gauss, *Theory of the Combination of Observations Least Subject to Errors: Part One, Part Two, Supplement.* Translated by G. W. Stewart

U. M. Ascher, R. M. M. Mattheij, and R. D. Russell, *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations*

K. E. Brenan, S. L. Campbell, and L. R. Petzold, *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*

Charles L. Lawson and Richard J. Hanson, *Solving Least Squares Problems*

J. E. Dennis, Jr. and Robert B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*

Richard E. Barlow and Frank Proschan, *Mathematical Theory of Reliability*

Cornelius Lanczos, *Linear Differential Operators*

Richard Bellman, *Introduction to Matrix Analysis, Second Edition*

Beresford N. Parlett, *The Symmetric Eigenvalue Problem*

Richard Haberman, *Mathematical Models: Mechanical Vibrations, Population Dynamics, and Traffic Flow*

Peter W. M. John, *Statistical Design and Analysis of Experiments*

Tamer Başar and Geert Jan Olsder, *Dynamic Noncooperative Game Theory, Second Edition*

Emanuel Parzen, *Stochastic Processes*

Petar Kokotović, Hassan K. Khalil, and John O'Reilly, *Singular Perturbation Methods in Control: Analysis and Design*

Jean Dickinson Gibbons, Ingram Olkin, and Milton Sobel, *Selecting and Ordering Populations: A New Statistical Methodology*

James A. Murdock, *Perturbations: Theory and Methods*

*First time in print.

Classics in Applied Mathematics (continued)

Ivar Ekeland and Roger Témam, *Convex Analysis and Variational Problems*

Ivar Stakgold, *Boundary Value Problems of Mathematical Physics, Volumes I and II*

J. M. Ortega and W. C. Rheinboldt, *Iterative Solution of Nonlinear Equations in Several Variables*

David Kinderlehrer and Guido Stampacchia, *An Introduction to Variational Inequalities and Their Applications*

F. Natterer, *The Mathematics of Computerized Tomography*

Avinash C. Kak and Malcolm Slaney, *Principles of Computerized Tomographic Imaging*

R. Wong, *Asymptotic Approximations of Integrals*

O. Axelsson and V. A. Barker, *Finite Element Solution of Boundary Value Problems: Theory and Computation*

David R. Brillinger, *Time Series: Data Analysis and Theory*

Joel N. Franklin, *Methods of Mathematical Economics: Linear and Nonlinear Programming, Fixed-Point Theorems*

Philip Hartman, *Ordinary Differential Equations, Second Edition*

Michael D. Intriligator, *Mathematical Optimization and Economic Theory*

Philippe G. Ciarlet, *The Finite Element Method for Elliptic Problems*

Jane K. Cullum and Ralph A. Willoughby, *Lanczos Algorithms for Large Symmetric Eigenvalue Computations, Vol. I: Theory*

M. Vidyasagar, *Nonlinear Systems Analysis, Second Edition*

Robert Mattheij and Jaap Molenaar, *Ordinary Differential Equations in Theory and Practice*

Shanti S. Gupta and S. Panchapakesan, *Multiple Decision Procedures: Theory and Methodology of Selecting and Ranking Populations*

Eugene L. Allgower and Kurt Georg, *Introduction to Numerical Continuation Methods*

Leah Edelstein-Keshet, *Mathematical Models in Biology*

Heinz-Otto Kreiss and Jens Lorenz, *Initial-Boundary Value Problems and the Navier-Stokes Equations*

J. L. Hodges, Jr. and E. L. Lehmann, *Basic Concepts of Probability and Statistics, Second Edition*

George F. Carrier, Max Krook, and Carl E. Pearson, *Functions of a Complex Variable: Theory and Technique*

Friedrich Pukelsheim, *Optimal Design of Experiments*

Israel Gohberg, Peter Lancaster, and Leiba Rodman, *Invariant Subspaces of Matrices with Applications*

Lee A. Segel with G. H. Handelman, *Mathematics Applied to Continuum Mechanics*

Rajendra Bhatia, *Perturbation Bounds for Matrix Eigenvalues*

Barry C. Arnold, N. Balakrishnan, and H. N. Nagaraja, *A First Course in Order Statistics*

Charles A. Desoer and M. Vidyasagar, *Feedback Systems: Input-Output Properties*

Stephen L. Campbell and Carl D. Meyer, *Generalized Inverses of Linear Transformations*

Alexander Morgan, *Solving Polynomial Systems Using Continuation for Engineering and Scientific Problems*

I. Gohberg, P. Lancaster, and L. Rodman, *Matrix Polynomials*

Galen R. Shorack and Jon A. Wellner, *Empirical Processes with Applications to Statistics*

Richard W. Cottle, Jong-Shi Pang, and Richard E. Stone, *The Linear Complementarity Problem*

Rabi N. Bhattacharya and Edward C. Waymire, *Stochastic Processes with Applications*

Robert J. Adler, *The Geometry of Random Fields*

Mordecai Avriel, Walter E. Diewert, Siegfried Schaible, and Israel Zang, *Generalized Concavity*

Rabi N. Bhattacharya and R. Ranga Rao, *Normal Approximation and Asymptotic Expansions*

Françoise Chatelin, *Spectral Approximation of Linear Operators*

Yousef Saad, *Numerical Methods for Large Eigenvalue Problems, Revised Edition*

Achi Brandt and Oren E. Livne, *Multigrid Techniques: 1984 Guide with Applications to Fluid Dynamics, Revised Edition*

Multigrid Techniques

1984 Guide with Applications to Fluid Dynamics

Revised Edition

Achi Brandt
The Weizmann Institute of Science
Rehovot, Israel
University of California at Los Angeles
Los Angeles, California

Oren E. Livne
University of Utah
Salt Lake City, Utah

siam.
Society for Industrial and Applied Mathematics
Philadelphia

This SIAM edition is an updated republication of the work first published by the Institute for Mathematics and Data Processing, St. Augustin, Germany, in 1984.

10 9 8 7 6 5 4 3 2 1

Library of Congress Cataloging-in-Publication Data
Brandt, Achi.
Multigrid techniques : 1984 guide with applications to fluid dynamics / Achi Brandt, Oren E. Livne. -- Rev. ed.
p. cm.
Includes bibliographical references and index.
ISBN 978-1-611970-74-6 (pbk.)
1. Fluid dynamics--Mathematics. 2. Multigrid methods (Numerical analysis) 3. Differential equations, Partial--Numerical solutions. I. Livne, Oren E. II. Title.
QA911.B667 2011
532'.05015182--dc22

2011011296

Contents

List of Figures

List of Tables

Preface to the Classics Edition

The *Multigrid Guide* presents the best known practices and techniques for developing multigrid solvers. As best practices evolve with on-going developments, the history of the *Guide* mirrors the history of the field of multigrid research. We delineate between two eras that must be borne in mind when reading this book: 1984 and earlier, and 1984 to the present time.

The earlier period (summarized in the 1984 Guide). Parts I and II of that *Guide* were based on [Bra82b], the first being an expansion of an even earlier mini-guide [Bra80c]. The present Classics edition of the *1984 Guide* includes quite a few minor corrections, additional comments and clarifications of the original manuscript; still, it overall describes the state of the art of multigrid as of 1984 and cites other works of that time. Multigrid solvers for discretized elliptic partial differential equations on well-structured grids, including various CFD systems, are well represented, as they had already matured at that time; but later important multigrid developments are absent. To maintain consistency with the rest of the book, the Introduction (§0) has not been updated, so "recent" developments referenced therein are now nearly thirty years old.

Equipped with the hindsight of contemporary research, yet faithful to the spirit of the *1984 Guide*, only few essential modifications were made. Chapter 14 was thoroughly revised to emphasize general solver performance predictors rather than Local Mode Analysis (LMA). In the early days, the latter was the best approach to practical quantitative performance analysis; hence it is extensively used throughout the entire Part I of this book. While LMA predictions are still perfectly valid today, the new predictors are simpler and preferable in many circumstances. Additionally,

- The original `CycleV` model Fortran program was replaced with a modern object-oriented MATLAB program in §1.5 and Appendix A.

- The local relaxation rule (§5.7) and its FMG application (§9.6) were added.

- The proper usage of a large cycle index, including fractional values, is now explained in §6.2.

Recent Developments (1984 to present). We caution the reader that this edition of the *Guide* falls short of representing later multigrid developments. We regard this book as a baseline for the future *Multigrid Guide 2.0* project, which will be continuously updated to match contemporary research and literature. The *2.0* project is accessible online at `http://www.siam.org/books/CL67`.

In particular, many bibliographical items in the present edition are outdated. Some of the cited technical reports are no longer available. An ever-growing multigrid literature has since emerged, including basic books [BHM00, Hac85, TOS00] and a plethora of works in the proceedings of over thirty Copper Mountain and European conferences on multigrid methods. Reviews of progressively more recent developments have been given in [Bra88, Bra89, Bra02]. These and many other articles are now electronically available at `http://www.wisdom.weizmann.ac.il/~achi/`.

Since 1984, multigrid development has been shifting towards Algebraic Multigrid (AMG), which aims at simplifying complex multigrid design scenarios by automatically constructing a grid hierarchy and inter-grid operators from the given fine-grid matrix. The basic idea is already described in §1.1 of the *1984 Guide* and its present edition, but the dedicated section (§13.1) lacks details as AMG was still in its infancy in 1984. "Classical AMG" was devised over the following decade, the Ruge-Stuben algorithm [RS87] becoming its most popular variant. In the *2.0 Guide* we will focus on a yet more recent approach called Bootstrap Algebraic Multigrid (BAMG) [Bra02, §17.2], which has a wider scope as well as inspires improvements to existing geometric multigrid solvers.

We plan to add new chapters on various generalizations and applications of the multiscale methodology, some of which are outlined in [Bra02]:

- Further work on anisotropic problems and various important PDE systems such as elasticity and magnetohydrodynamics.
- Wave equations, eigenproblems and electronic structures in quantum chemistry.
- Global optimization and stochastic simulations in statistical physics [BR02].
- "Systematic Upscaling," a general multiscaling methodology for deriving macroscopic equations from microscopic physical laws [Bra10].
- Graph problems with applications to image processing [SGS+06], data analysis [RSBss] and transportation networks.

Finally, we wish to invite you, the reader, to take an active role and contribute to the *Multigrid Guide* project. We welcome comments

and suggestions. We want the *Guide* to be a reflection of our collective knowledge and understanding of multigrid methods.

Preface

Starting with an elementary exposition of multigrid fast solvers with insights into their analyses and their most general algebraic applicability, detailed practical guidelines are then given how to obtain, stage by stage, the full multigrid efficiency for general elliptic and non-elliptic problems, linear as well as nonlinear, scalar or vectorial, smooth or strongly discontinuous, with various possible singularities, boundary conditions and supplementary global conditions.

Quantitative insights through local mode analyses, combined with gradual algorithm development, are emphasized throughout, and general rules and approaches are explained for the design of relaxation, coarsening and interpolation. Beyond these fast-solver aspects of multigrid, advanced methods are then described, including various applications of the Full Approximation Scheme (FAS), local refinement and local coordinate transformations, error estimation and grid adaptation criteria, small storage algorithms, and the double discretization and other techniques for high-order approximations.

Also briefly outlined are Algebraic Multigrid (AMG); multi-level reduction of complexity for integral equations and for chains of problems; treatment of time-dependent problems; eigenvalue problems; and optimization of PDEs with design parameters.

Dedicated chapters describe in detail the solution of Cauchy-Riemann, Stokes and incompressible and compressible Navier-Stokes equations, with numerical results for staggered and non-staggered grids.

Chapter 0

Introduction

0.1 Where and why multigrid can help

The starting point of the multigrid method (or more generally, the Multi-Level Adaptive Technique - MLAT), and indeed also its ultimate upshot, is the following "golden rule":

The amount of computational work should be proportional to the amount of real physical changes in the computed system. Stalling numerical processes must be wrong.

That is, whenever the computer grinds very hard for very small or slow real physical effect, there must be a better computational way to achieve the same goal. Common examples of such stalling are the usual iterative processes for solving the algebraic equations arising from discretizing partial-differential, or integro-differential, boundary-value (steady-state) problems, in which the error has relatively small changes from one iteration to the next. Another example is the solution of time-dependent problems with time-steps (dictated by stability requirements) much smaller than the real scale of change in the solution. Or, more generally, the use of too-fine discretization grids, where in large parts of the computational domain the meshsize and/or the timestep are much smaller than the real scale of solution changes. Etc.

If you have such a problem, multi-level techniques may help. The trouble is usually related to some "stiffness" in your problem; i.e., to the existence of several solution components with different scales, which conflict with each other. For example, smooth components, which are efficiently approximated on coarse grids but are slow to converge in fine-grid processes, conflict with high-frequency components which must be approximated on the fine grids. By employing interactively several scales of discretization, multilevel techniques resolve such conflicts, avoid stalling and do away with

the computational waste.

The main development of multilevel techniques has so far been limited to their role as fast solvers of the algebraic equations arising in discretizing boundary-value problems (steady-state problems or implicit steps in evolution problems). The multigrid solution of such problems usually requires just few (four to ten) work units, where a work unit is the amount of computational work involved in *expressing* the algebraic equations (see §7.3). This efficiency is obtained for all problems on which sufficiently research has been made, from simple model problems to complicated nonlinear systems on general domains, including diffusion problems with strongly discontinuous coefficients, integral equations, minimization problems with constraints; from regular to singular-perturbation and non-elliptic boundary-value problems. Due to the iterative nature of the method, nonlinear problems require no more work than the corresponding linearized problems. Linearization is thus neither needed nor usually recommended (see §8.3). Problems with global constraints are solved as fast as the corresponding unconstrained difference equations, using a technique of enforcing the constraints only at the coarse-grid stages of the algorithm (§5.6). Few work units are also all the work required in calculating each eigenfunction of discretized eigenproblems (§8.3.1). Moreover, all multigrid processes can be fully parallelized and vectorized. In 1984, a model multigrid program on the Cyber 205 solved 3 million equations per second [BB83].

Beyond the fast solvers, multilevel techniques can be very useful in other ways related to stiffness. They can provide very efficient grid-adaptation procedures for problems (either boundary-value or evolution problems) in which different scales of discretization are needed in different parts of the domain (see §9). They can give new dimension of efficiency to stiff evolution problems (§16). They can also resolve the conflict between higher accuracy and stability in case of non-elliptic and singular perturbation boundary-value problems (§10.2). In addition, multi-level techniques can enormously reduce the amount of discrete relations employed in solving chains of similar boundary-value problems (as in processes of continuation), and in optimization problems (see §§13, 15), or in solving integral equations (see §8.6). They can also be used to vastly cut the required computer storage (§8.7). Feasibility studies have already established the effectiveness of the multilevel approach for all of these applications, which continue to be active research areas.

Multilevel processes can also cut, sometimes by several orders of magnitude, the computer resources needed to solve some *large systems which do not originate from partial-differential or integral equations.* The common feature in those systems in that they involve many unknowns related in a low-dimensional space; i.e., each unknown u_P is defined at a point $P = (x_1, \ldots, x_d)$ of a low-dimensional space (d is usually 2 or 3) and the coupling between two values u_P and u_Q generally becomes weaker or smoother as the distance between P and Q increases, except perhaps for

a small number of particular pairs (P, Q). Examples are: the equations of multivariate interpolation of scattered data [Bra83, Mei79]; geodetic problems of finding the locations of many stations that best fit a large set of local observations [Mei80]; problems in transportation, economy [VDR79], and queuing theory [Kau82]; statistical problems on lattices, arising in statistical mechanics (e.g., Ising model) and in "gauge" theories of elementary particles; and various systems of tomography, image processing, picture reconstruction and pattern recognition [NOR81, San81, GG83]. There is in fact strong evidence that the human vision processes themselves are multilevelled [CR68, WB79, Ter83]. In several of these areas multigrid research has just recently started [Bra10].

0.2 About this guide (the 1984 edition)

The opening chapter of this Guide is dedicated to numerical analysts who have no previous acquaintance with multigrid methods. It also gives some references to other introductory material. (§§1.1 and 1.7 in that chapter may interest veteran multigridders, too.)

The main parts of this Guide are chiefly intended for people with some multigrid knowledge, or even experience. In fact, we were mainly motivated by the following situation, so often encountered in last few years: A good numerical analyst tries a multigrid solver on a new problem. He knows the basics, he has seen it implemented on another problem, so he has no trouble writing the program. He gets results, showing a certain rate of convergence, perhaps improving a former rate obtained with a one-grid program. Now he is confronted with the question: Is this the real multigrid efficiency? or is it many times slower, due to some conceptual error or programming bug? The algorithm has many parts and aspects: relaxation sweeps and coarse-to-fine and fine-to-coarse transfers at interior points and at points near boundaries; relaxation and transfers of the boundary conditions themselves; treatment of boundary and interior singularities and/or discontinuities; choosing the coarse-grid variables and defining its equations; the method of solving on the coarsest grid; the general flow of the algorithm; etc. A single error (a wrong scheme or a bug) in any of these parts may degrade the whole performance very much, but it is still likely to give an improvement over a one-grid method, misleading the analyst to believe he has done a good job. How can an error be suspected and detected? How can one distinguish between various possible troubles? What improved techniques are available?

The key to a fully successful code is to know in advance what efficiency is ideally obtainable, and then to construct the code gradually in a way that ensures approaching that ideal, telling us at each stage which process may be responsible for a slowdown. It is important to work in that spirit: Do not just observe what efficiency is obtained by a given multigrid algorithm, but ask yourself what is the *ideal* efficiency and *find out* how to obtain it.

To guide inexperienced multigridders in that spirit is the main purpose of this Guide.

We believe that any discrete system derived from a continuous problem is solvable "to the level of truncation errors" in just few "work units" (see §7.3). To obtain this performance, the first crucial step is to construct a relaxation scheme with a high "smoothing rate" (see §3). Then the interior inter-grid transfers and coarse-grid operator should be designed (§4), and full numerical experiments can be started with cycling algorithms, aiming at obtaining the interior rate (§§5, 6). Finally, "Full Multi-Grid" (FMG) algorithms can then be implemented, and "solvability in just few work units" can be tested (§7). These stages of development are outlined in Part I below, pointing out many possibilities and technical points, together with theoretical tools needed for quantitative insights into the main processes.

The quantitative aspect in these theoretical tools is important, since we want to distinguish between the efficiency of several candidate multigrid algorithms, all of which may be "asymptotically optimal" (i.e., solving the problem in a uniformly bounded number of work units), but some of which may still be several orders of magnitude faster than others. Except for some model problems, most present-day rigorous mathematical theories of multigrid algorithms do not give us accurate enough insights (see §14), hence the present guide will emphasize the role of "local mode analyses". These analyses (see §§2.1, 3.1, 4.1, 7.4, 7.5) neglect some of the less work-consuming processes so as to obtain a clear and precise picture of the efficiency of the more important processes. The predictions so obtained can be made accurate enough to serve in program optimization and debugging. Experience has taught us that careful incorporation of such theoretical studies is essential for producing reliable programs which fully utilize the potential of the method.

Part II of this Guide summarizes more advanced multigrid techniques and insights. Mainly, it is intended to show how to use the multilevel techniques far beyond their more familiar capacity as fast linear algebraic solvers. See the survey in §0.1 and the list of contents.

In part III we bring applications to fluid dynamics. Whereas in Part I the information about technique for all problems is ordered according to their common stages of development, in Part III we study specific problems, separately from each other. The order is again according to a certain line of development, namely, starting from simple problems and gradually learning our way to more complicated ones. The emphasis is on *systems* of differential equations; scalar problems are not separately treated.

This Guide can be viewed as an extension of an earlier work [Bra82b], with numerous updates, few new sections (§§3.8, 5.7), a new chapter (Chapter 1) and a whole new part (Part III). It is not intended just for teaching, but also for organizing and unifying the material. It is also used as an opportunity to mention some advances which have not appeared in the literature before. In particular, [Bra82b] already included some new relax-

ation schemes such as "Box Gauss-Seidel" (§3.4) and relaxation with only sub-principal terms (§10.3); the general rule of block relaxation (§3.3); an analysis of the orders of interpolation and residual transfers which should be used in solving *systems* of differential equations (§§4.3, 7.1); the multigrid treatment of global constraints (§5.6); an applications of FAS to obtain much more efficiency discretization to *integral* equations, leading sometimes to solutions in $O(n)$ operations, where n is the number of discrete unknowns (§8.6; [BL90]); some innovations in higher-order techniques (§10.2); unified grid switching and adaptation criteria (§9.6); multi-level approach to optimization (§13; [BR03]); and the Algebraic Multi-Grid (AMG) method (§13.1). Results from a recent work, still unpublished, on non-elliptic and singular perturbation problems [Bra81a]) are also mentioned, including a summary of stability requirements (§2.1); the double-discretization scheme (§10.2); the two-level FMG mode analysis, which tends to replace the usual two-level mode analysis (§§7.4, 7.5); the F cycle (a hybrid of V and W cycles; §6.2). (The main topic from [Bra81a] hardly mentioned here is the multigrid treatment of discontinuities, in which research is currently underway. But see §§2.2 and 8.5). The discussions on the real role of relaxation (§12), on the general approach to coarsening questions (§11), and on "algebraization" and "dealgebraization" trends in multigrid development (§13) were added as a general new viewpoints, related to each other, somewhat philosophical, but certainly useful.

This Guide includes in addition some remarks about the "principal linearization" used in relaxation (usually meaning no linearization at all – see §3.4); the general algebraic property of slowly converging relaxation schemes (§1.1); the superfluity of "perfect smoothers" for non-elliptic or slightly elliptic systems (§§3.3, 3.6); the principle of relaxing general PDE operators in terms of the factors of their subprincipal-part determinant (§3.7); some new debugging devices (e.g., §§4, 5.1); stabilizing coarsening by added global constraints (§5.6); the treatment of structural singularities such as reentrant corners (§§5.7, 9.6); and new numerical results for the Stokes and compressible and incompressible Navier-Stokes equations, including results for non-staggered grids (§§18.6, 19.5).

Chapter 1

Elementary Acquaintance With Multigrid

The following few pages would acquaint you with the conceptual basis of all multigrid solvers, with elementary mode-analysis and with an example of a simple algorithm along with its MATLAB implementation and output.

For a more detailed introduction to multigrid techniques, through a comprehensive treatment of some *model* problems by a variety of multigrid algorithms, mode analyses and numerical experiments, see [ST82]. The latter appears in a book [HT82] which also includes a previous version of the current Guide [Bra82b], a complete (as of 1982) multigrid bibliography, and many other multigrid papers. Additional material, and in fact summaries of all new multigrid papers, appear quarterly in the *MULTIGRID NEWSLETTER*, obtainable in North and South America from its Editor (Steve McCormick, Department of Mathematics, Colorado State University, Fort Collins, CO 80523, U.S.A.) and in other countries from the Managing Editor (Kurt Brand, GMD/FIT, Postfach 1240, D-5205 St. Augustin 1, Federal Republic of Germany). Periodically, it publishes a complete list of all past papers. Also available is a set of videoed multigrid lectures [Bra83].

1.1 Properties of slowly converging errors

The origin of multi-level (multigrid) fast solvers is a certain insight concerning the nature of the algebraic errors that become dominant when conventional iterative schemes are slow to converge. Let us first present this insight in its most general algebraic setting, where a matrix equation $Ax = b$ is being solved for any (possibly rectangular) matrix A.

For any approximate solution $\tilde{x}$, denote by $e = x - \tilde{x}$ the error vector and by $r = Ae = b - A\tilde{x}$ the vector of residuals. The common feature of all iterative schemes is that at each step some corrections to $\tilde{x}$ are calculated based on the magnitude of certain residuals. As a result, convergence must

be slow if the individual residuals do not show the true magnitude of the error, i.e., if r is in some sense small compared with e. The converse is also true: If convergence of a *suitable* relaxation scheme is slow, residuals must in some sense be small compared with e.

To see this more concretely, consider for example Kaczmarz relaxation applied to the above matrix equation and converging to a solution x. Denoting by a_i the i^{th} row of A, the Kaczmarz step corresponding to that row is to replace $\tilde{x}$ by $\tilde{x} + (r_i/a_i a_i^\top)a_i^\top$, thereby forcing r_i to zero. A full Kaczmarz sweep is the employment of such a step for each row of A, in the natural ordering . (We take this scheme as our example because it applies to the most general matrix: It converges whenever a solution exists [Tan71].) Let

$$E = e^\top e = \sum_i e_i^2 \quad \text{and} \quad R = \sum_i \frac{r_i^2}{a_i a_i^\top}$$

be square norms for errors and residuals, respectively, evidently scaled so that they are comparable. One can then prove the following result [Bra86, Theorem 3.4]:

Theorem 1.1. *A Kaczmarz sweep reduces E at least by* $\max\{\gamma_0 R_0, \gamma_1 R_1\}$, *where R_0 and R_1 are the values of R before and after the sweep, respectively, and*

$$\gamma_0 = ((1+\gamma_+)(1+\gamma_-))^{-1}, \qquad \gamma_1 = (\gamma_-\gamma_+)^{-1}$$
$$\gamma_- = \max_i \sum_{j<i} \frac{|a_i a_j^\top|}{a_i a_i^\top}, \quad \gamma_+ = \max_i \sum_{j>i} \frac{|a_i a_j^\top|}{a_i a_i^\top}.$$

The theorem in essence says that *slow convergence can occur only when R is small compared with E* (observe that in case A arises from the discretization of differential equations, γ_i are completely local quantities, independent of the size of A. The above values of γ_0 and γ_1 are close to the best possible ones). Similar theorems (with suitably modified E, R and γ_i) hold for all familiar relaxation schemes, such as Gauss-Seidel, Jacobi, SOR, and block schemes (line relaxation, etc. See [Bra86]).

Because E and R were scaled to be comparable, it is generally only for special types of error components that the slow convergence condition

$$R \ll E \tag{1.1}$$

is satisfied. The deeper (1.1) is satisfied, the more special must the error be, and hence the fewer the number of parameters needed to approximate it. Thus, broadly speaking, *relaxation efficiently reduces the information content of the error, and quickly makes it approximable by far fewer variables.* This can be shown to be true even for nonlinear systems.

When the matrix equation $Ax = b$ is a discretization of a differential system $Lu = f$ on some grid, condition (1.1), rewritten in the form $\|Ae\| \ll \|A\|\|e\|$, can be interpreted as saying that the error e approximates a continuous function v satisfying $\|Lv\| \ll \|L\|\|v\|$, in some corresponding norms. If L is a uniformly elliptic operator, this implies that e is a smooth function (in case L is not elliptic, a certain smoothness along characteristics is at least implied). Hence, *relaxation efficiently reduces non-smooth error components, thus making the error approximable on a coarser grid* (where solution is much less expensive). A precise measure for this efficiency is discussed next.

1.2 Error smoothing and its analysis: Example

For clarity, consider a simple example. Suppose the partial differential equation

$$Lu(x,y) \equiv a\frac{\partial^2 u(x,y)}{\partial x^2} + c\frac{\partial^2 u(x,y)}{\partial y^2} = f(x,y), \qquad (a,c>0) \tag{1.2}$$

is to be solved with some suitable boundary conditions. Denoting by u^h and f^h approximations to u and f, respectively, on a grid with meshsize h, the usual second-order discretization of (1.2) is

$$L^h u^h_{\alpha,\beta} \equiv a\frac{u^h_{\alpha-1,\beta} - 2u^h_{\alpha,\beta} + u^h_{\alpha+1,\beta}}{h^2} + c\frac{u^h_{\alpha,\beta-1} - 2u^h_{\alpha,\beta} + u^h_{\alpha,\beta+1}}{h^2} = f^h_{\alpha,\beta}, \tag{1.3}$$

where

$$u^h_{\alpha,\beta} = u^h(\alpha h, \beta h), \qquad f^h_{\alpha,\beta} = f(\alpha h, \beta h); \qquad \alpha,\beta \text{ integers}$$

(in the multigrid context it is important to define the difference equations in this divided form, without, for example, multiplying throughout by h^2, in order to get the proper relative scale at different grids). Given an approximation $\tilde{u}$ to u^h, a simple example of a relaxation scheme to improve it is the following.

Gauss-Seidel Relaxation. The points (α,β) of the grid are scanned one by one in some prescribed order; e.g., lexicographic order. At each point, the value $\tilde{u}_{\alpha,\beta}$ is replaced by a new value, $\overline{u}_{\alpha,\beta}$, such that (1.3) at that point is satisfied. That is, $\overline{u}_{\alpha,\beta}$ satisfies

$$a\frac{\overline{u}^h_{\alpha-1,\beta} - 2\overline{u}^h_{\alpha,\beta} + \tilde{u}^h_{\alpha+1,\beta}}{h^2} + c\frac{\overline{u}^h_{\alpha,\beta-1} - 2\overline{u}^h_{\alpha,\beta} + \tilde{u}^h_{\alpha,\beta+1}}{h^2} = f^h_{\alpha,\beta}, \tag{1.4}$$

where the new values $\overline{u}^h_{\alpha-1,\beta}, \overline{u}^h_{\alpha,\beta-1}$ are used because, in the lexicographic order, by the time (α,β) is scanned, new values have already replaced old ones at $(\alpha-1,\beta)$ and $(\alpha,\beta-1)$.

A complete pass, scanning in this manner all gridpoints, is called a (Gauss-Seidel lexicographic) *relaxation sweep.* The new approximation $\overline{u}$ does not satisfy (1.3), and further relaxation sweeps may be required to improve it. An important quantity therefore is the *convergence factor* μ, which may be defined by

$$\mu := \frac{\|\overline{v}\|}{\|v\|}, \qquad \text{where} \quad v := u^h - \tilde{u}, \quad \overline{v} := u^h - \overline{u}, \tag{1.5}$$

$\|\cdot\|$ being any suitable discrete norm. For the Gauss-Seidel scheme, with the possible exception of its first few sweeps, $\mu = 1 - O(h^2)$. This means that $O(h^{-2})$ relaxation sweeps are needed to reduce the error by an order of magnitude.

In multigrid methods, however, the role of relaxation is not to reduce the error, but to smooth it out so that it becomes well approximable on a coarser grid. This relaxation can do very effectively. Indeed, subtracting (1.3) from (1.4), the relation

$$a\left(\overline{v}^h_{\alpha-1,\beta} - 2\overline{v}^h_{\alpha,\beta} + v^h_{\alpha+1,\beta}\right) + c\left(\overline{v}^h_{\alpha,\beta-1} - 2\overline{v}^h_{\alpha,\beta} + v^h_{\alpha,\beta+1}\right) = 0 \tag{1.6}$$

shows that $\overline{v}^h_{\alpha,\beta}$ is a weighted average of neighboring values (of both v and $\overline{v}$), so that, if the old error v is not smooth, the new error $\overline{v}$ must be much smoother.

To analyze the smoothing effect of a relaxation sweep quantitatively, we take advantage of its *local* nature (points several meshsizes apart affecting each other exponentially little). It allows us, for the purpose of studying the smoothing well in the interior, to regard the grid as embedded in a rectangular domain. We can then expand both v and $\overline{v}$ in Fourier series

$$v_{\alpha,\beta} = \sum A_{\underline{\theta}} e^{i(\theta_1\alpha+\theta_2\beta)}, \qquad \overline{v}_{\alpha,\beta} = \sum \overline{A}_{\underline{\theta}} e^{i(\theta_1\alpha+\theta_2\beta)}, \tag{1.7}$$

where $\underline{\theta} := (\theta_1, \theta_2)$ and the summations are over a subset of the square $|\underline{\theta}| := \max(|\theta_1|, |\theta_2|) \le \pi$. Substituting (1.7) into (1.6) yields

$$\left(ae^{-i\theta_1} + ce^{-i\theta_2} - 2a - 2c\right) A_{\underline{\theta}} + \left(ae^{i\theta_1} + ce^{i\theta_2}\right) \overline{A}_{\underline{\theta}} = 0. \tag{1.8}$$

Hence, the *amplification factor* of the $\underline{\theta}$ component due to one relaxation sweep is

$$\mu(\underline{\theta}) = \left|\frac{\overline{A}_{\underline{\theta}}}{A_{\underline{\theta}}}\right| = \left|\frac{ae^{i\theta_1} + ce^{i\theta_2}}{2a + 2c - ae^{-i\theta_1} - ce^{-i\theta_2}}\right|. \tag{1.9}$$

Observe that $\mu(\underline{\theta}) \to 1$ as $\underline{\theta} \to (0,0)$. In domains of diameter $O(1)$, the lowest non-trivial Fourier components have $|\underline{\theta}| = O(h)$, for which $\mu(\underline{\theta}) = 1 - O(h^2)$, showing why convergence factors are that bad. Here, however, we are only interested in the smoothing effect, i.e., in the amplification factors of those components not approximable on a coarser grid. These are the components for which $(h/H)\pi \le |\underline{\theta}| \le \pi$, where H is the meshsize of

the next coarser grid. We usually assume $H/h = 2$, because it is the most convenient, and as effective as any other meshsize ratio (cf. §4.2). The *smoothing factor* is thus defined to be

$$\overline{\mu} := \max_{\frac{\pi}{2} \le |\underline{\theta}| \le \pi} \mu(\underline{\theta}). \tag{1.10}$$

It essentially gives the relaxation convergence factor for those components which need to converge only through relaxation; others will also converge through their approximation on the coarser grid.

Consider first the case $a = c$ (Poisson equation). A simple calculation shows that $\overline{\mu} = \mu(\pi/2, \arccos(4/5)) = .5$. This is a very satisfactory rate; it implies that *three relaxation sweeps reduce the high-frequency error-components by almost an order of magnitude.* Similar rates are obtained for general a and c, provided a/c is of moderate size.

The rate of smoothing is less remarkable in the degenerate case $a \ll c$ (or $c \ll a$). For instance,

$$\mu\left(\frac{\pi}{2}, 0\right) = \left(\frac{a^2 + c^2}{a^2 + (c + 2a)^2}\right)^{\frac{1}{2}}$$

which approaches 1 as $a \to 0$. Thus, for problems with such a degeneracy, Gauss-Seidel relaxation is not a suitable smoothing scheme. But better schemes exist, such as the following example.

Line Relaxation. Instead of treating each grid point (α, β) separately, one simultaneously takes a vertical line of points at a time, i.e., the set of all points (α, β) with the same α. All values $\tilde{u}_{\alpha,\beta}$ on such a line are simultaneously replaced by new values $\overline{u}_{\alpha,\beta}$ that simultaneously satisfy eqs. (1.3) on that line. (This is easy and inexpensive to do, because the system of equations to be solved for each line is a tridiagonal, diagonally dominant system.) As a result, we get the same relation as (1.4) above, except that $\tilde{u}_{\alpha,\beta+1}$ is replaced by $\overline{u}_{\alpha,\beta+1}$. Hence, instead of (1.9) we now obtain

$$\mu(\underline{\theta}) = \left|\frac{\overline{A}_{\underline{\theta}}}{A_{\underline{\theta}}}\right| = \left|\frac{a}{2(a + c - c\cos(\theta_2)) - ae^{-i\theta_1}}\right|. \tag{1.11}$$

from which one can derive the smoothing factor

$$\overline{\mu} = \max\left\{5^{-\frac{1}{2}}, \frac{a}{a + 2c}\right\}, \tag{1.12}$$

which is very satisfactory, even in the degenerate case $a \ll c$.

This situation is very general. Namely, for any stable discretization of a well-posed differential boundary-value problem, there exists a relaxation scheme which very efficiently reduces non-smooth error components (see §3 and §5.3). Moreover, the smoothing factor (1.10) for any candidate relaxation scheme is usually easy to calculate (e.g. a computer program

that numerically evaluates $\mu(\underline{\theta})$ on a sufficiently fine $\underline{\theta}$ grid; see examples in [Wei01]), even for nonlinear equations or equations with non-constant coefficients, by local linearization and coefficients freeze (see §3.1). This gives us a general tool for optimizing the relaxation scheme and predict its efficiency. It is the first example of *local mode analysis*, extensively used in multigrid analysis (see §§3.1, 4.1, 7.4 and 7.5).

1.3 Coarse grid correction

We have seen that relaxation sweeps very quickly reduce all high-frequency components of the error. Its smoother part should then be reduced by being approximated on a coarser grid, a grid with meshsize $H = 2h$, say. Generally, for any linear fine-grid equation $L^h u^h = f^h$ (for the nonlinear case, see §8.1), and any approximate solution $\tilde{u}^h$, the error $v^h = u^h - \tilde{u}^h$ satisfies

$$L^h v^h = r^h, \qquad \text{where} \qquad r^h := f^h - L^h \tilde{u}^h . \tag{1.13}$$

It can therefore be approximated by the coarse-grid function v^H that satisfies

$$L^H v^H = I_h^H r^h , \tag{1.14}$$

where L^H is some coarse-grid approximation to L^h (e.g., a finite-difference approximation on grid H to the same differential operator approximated by L^h), and I_h^H is a fine-to-coarse transfer operator, called *residual weighting* or *restriction*. That is, $I_h^H r^h$ is a coarse-grid function whose value at each point is a certain weighted average of the values of r^h at neighboring fine-grid points (see much more on this process of "coarsening" in §4 and 11, and on the treatment of boundary conditions and global conditions in §5.4, 5.5, and 5.6).

Having obtained an approximate solution $\tilde{v}^H$ to (1.14) (in a way to be discussed below), we use it as a correction to the fine-grid solution. Namely, we replace

$$\tilde{u}^h \leftarrow \tilde{u}^h + I_H^h \tilde{v}^H , \tag{1.15}$$

where I_H^h is a coarse-to-fine *interpolation* (also called *prolongation*). That is, at each fine-grid point, the value of $I_H^h \tilde{v}^H$ (designed to approximate the error v^h) is interpolated from values of $\tilde{v}^H$ at neighboring coarse-grid points. Linear interpolation can be use in most cases (further discussion of interpolation orders appears in §4.3). The whole process of calculating $I_h^H r^h$, solving (1.14) and interpolating the correction (1.15) is called a *coarse-grid correction*.

1.4 Multigrid cycle

To efficiently get an approximate solution to the coarse-grid equation (1.14), we employ the above solution process recursively; i.e., (1.14) is itself solved

by relaxation sweeps, combined with a still-coarser-grid corrections. We thus have a sequence of grids with meshsizes $h_1 > h_2 > \cdots > h_M$, where usually $h_k = 2h_{k-1}$. The grid-h_k equation is generally written as

$$L^k u^k = f^k , \tag{1.16}$$

where all the operators L^k approximate each other (e.g., they are all finite-difference approximations to the same differential operator), and unless k is the finest level ($k = M$), equation (1.16) is of the form (1.14), i.e., u^{k-1} is always designed to be the coarse correction to $\tilde{u}^k$ (the current approximation on the next finer grid), and hence

$$f^{k-1} = I_k^{k-1} \left(f^k - L^k \tilde{u}^k \right) \tag{1.17}$$

(superscripts and subscripts l are now used instead of h_l in the notation of §1.3. Also, u^{k-1} is used instead of v^{2h} for the purpose of uniform expressions at all levels).

The exact algorithm for improving a given approximate solution $\tilde{u}^k$ to (1.16) is usually the *multigrid cycle* (MGC)

$$\tilde{u}^k \leftarrow \mathrm{MGC}\left(k, \tilde{u}^k, f^k\right), \tag{1.18}$$

defined recursively as follows:

> If $k = 1$, solve (1.16) by Gaussian elimination or by several relaxation sweeps (either is usually inexpensive, because the grid is extremely coarse. For additional remarks concerning the coarsest-grid solution, see §6.3). Otherwise, do the following four steps:
>
> (A) Perform ν_1 relaxation sweeps on (1.16), resulting in a new approximation $\overline{u}^k$.
>
> (B) Starting with $\tilde{u}^{k-1} = 0$, make γ successive cycles of the type
>
> $$\tilde{u}^{k-1} \leftarrow \mathrm{MGC}\left(k-1, \tilde{u}^{k-1}, I_k^{k-1}\left(f^k - L^k \overline{u}^k\right)\right).$$
>
> (C) Calculate
>
> $$\overline{\overline{u}}^k = \overline{u}^k + I_{k-1}^k \tilde{u}^{k-1}. \tag{1.19}$$
>
> (D) Finally, perform ν_2 additional relaxation sweeps on (1.16), starting with $\overline{\overline{u}}^k$ and yielding the final $\tilde{u}^k$ of (1.18).

The sweep count ν_1 and ν_2 are usually either 0, 1 or 2, with $\nu = \nu_1 + \nu_2$ usually being 2 or 3 (see more about this in §4.1 and 6.1). The cycle count γ is usually either 1 or 2. The cycle with $\gamma = 1$ is called a *V cycle*, or $\mathrm{V}(\nu_1, \nu_2)$, in view of the shape of its flowchart (see Fig. 1.1). For a similar

reason, the cycle with $\gamma = 2$ is called a *W cycle*, or $\mathrm{W}(\nu_1, \nu_2)$ (see more about different cycles and alternative switching criteria in §6.2).

The ν sweeps performed in each V cycle on any grid h_k are expected to reduce error components with wave-length between $2h_k$ and $4h_k$ at least by the factor $\overline{\mu}^{\nu}$, where $\overline{\mu}$ is the smoothing factor (1.10). Because all grids are so traversed, *the cycle should reduce all error components by at least the factor* $\overline{\mu}^{\nu}$. Experience and more advanced theory show that for regular elliptic problems, this is indeed the case, provided the boundary conditions are properly relaxed, and correct inter-grid transfers are used. Thus, $\overline{\mu}$ can serve as an excellent predictor of the multigrid performance one should *be able to* obtain.

1.5 Model program and output

A simple unoptimized MATLAB program called `TestCycle` is included in Appendix A to illustrate multigrid algorithmic ideas and programming techniques. The software design easily carries over to any object-oriented language such as `C++` or `Java`(TM).

The main Class `TestCycle` solves the Poisson equation $\Delta u = F(x, y)$ with Dirichlet boundary conditions $u = G(x, y)$ on a rectangle by applying `numCycles` $V(\nu_1, \nu_2)$ cycles to a random initial guess.

The input options are centralized in Class `Options`. A sequence of `numLevels` grids is defined over the domain $[0, \texttt{domainSize(1)}] \times [0, \texttt{domainSize(2)}]$. The coarsest grid (level `1`) has $\texttt{nCoarsest(1)} \times \texttt{nCoarsest(2)}$ intervals of of length $\mathtt{h}_1$ each. Subsequent grids at levels $\mathtt{k} = 2, \ldots, \texttt{numLevels}$ are defined as uniform refinements with meshsizes $\mathtt{h}_k := 2^{1-k}\mathtt{h}_1$. Therefore, the problem is solved at the finest grid with meshsize $\mathtt{h}_{\texttt{numLevels}}$; coarse levels are only used to accelerate convergence.

The V-cycle flow depicted in Fig. 1.1 is implemented by Class `Cycle`, which manages an internal list of `Level` instances. Each `Level` instance encapsulates the discrete operator (the same 5-point approximation to the Laplace operator (1.3) with $a = c = 1$ is used at all levels), relaxation scheme (Gauss-Seidel), residual transfer I_k^{k-1} (full weighting) and interpolation of corrections I_{k-1}^{k} (bilinear). Note that the `Level` class is generic, and conveniently allows swapping in and out different multigrid ingredients without affecting the rest of the code.

The program output (generated with MATLAB 7.10.0.499 (R2010a)) is shown on the next page. At each step of the algorithm, the l_2 norm of the ("dynamic") residuals is printed, as well as a count of the cumulative relaxation work, where a finest level sweep serves as the work unit. Cycles exhibit an asymptotic convergence factor of about .11 per cycle, slightly better than $\overline{\mu} = .5^3 = .125$ predicted by the smoothing factor (the two-level mode analysis described in §4.1 is able to precisely predict the observed factor). This is equivalent to a convergence factor of $.11^{\frac{1}{3}} = .48$ per relaxation sweep.

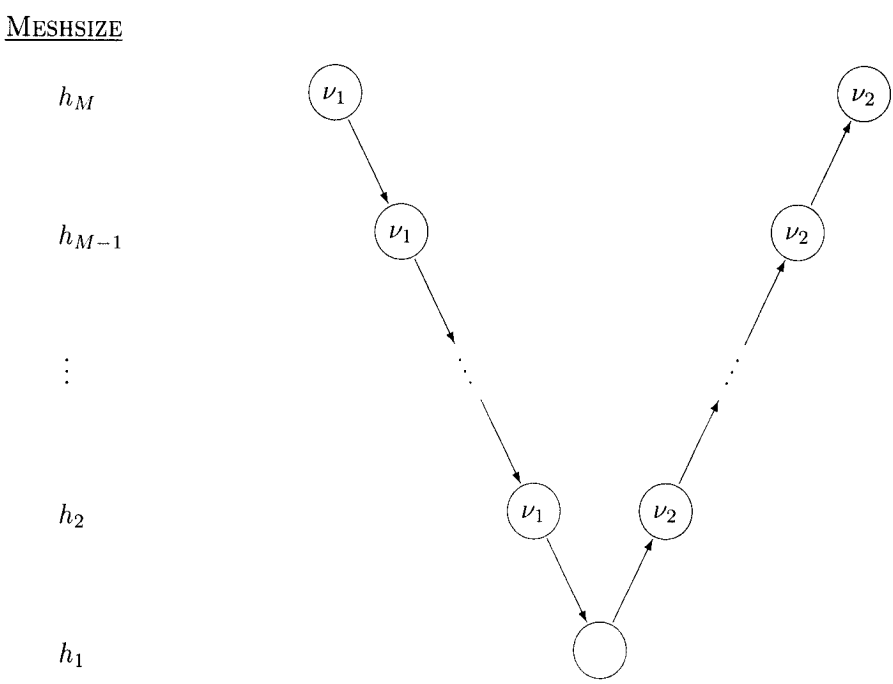


Figure 1.1. ***Multigrid cycle*** $V(\nu_1, \nu_2)$.

(ν_i) *stands for* ν_i *relaxation sweeps at the meshsize shown to the left. (At the coarsest grid :* $\nu_1 + \nu_2$ *relaxation sweeps are usually performed, or the equations are solved directly.)*

↘ *is the fine-to-coarse* $(k+1$ *to* $k)$ *transfer.* f^k *is defined by (1.17) and* u^k *is trivially initialized* $(u^k \leftarrow 0)$.

↗ *is the coarse-to-fine* $(k-1$ *to* $k)$ *interpolation of correction (1.19).*

The same algorithm attains the exact same efficiency on general non-rectangu-lar domains: `TestCycle`'s flexible design is extensible, although the implementation of its components naturally becomes more complex. Collections of multigrid programs with varying degrees of simplicity vs. generality are available, e.g. [Hym77, Dou05].

We strongly recommend the reader to experiment with the program and tweak it in various ways (vary the input options, modify the equations, etc.) to acquire a deep understanding of the conceptual and technical aspects of multigrid.

```
              TestCycle Output
              ----------------
>> TestCycle.run;
##################### CYCLE #1 ######################
LEVEL ACTION                    ERROR NORM    WORK
5     Initial                   1.211e+003     0.00
5     Relaxation sweep 1        3.512e+002     1.00
5     Relaxation sweep 2        1.312e+002     2.00
4     Initial                   5.661e+001     2.00
4     Relaxation sweep 1        2.673e+001     2.25
4     Relaxation sweep 2        1.752e+001     2.50
3     Initial                   1.353e+001     2.50
3     Relaxation sweep 1        7.609e+000     2.56
3     Relaxation sweep 2        5.408e+000     2.63
2     Initial                   4.126e+000     2.63
2     Relaxation sweep 1        2.296e+000     2.64
2     Relaxation sweep 2        1.362e+000     2.66
1     Initial                   7.402e-001     2.66
1     Relaxation sweep 400      0.000e+000     4.22
2     Coarse-grid correction    9.729e-001     4.22
2     Relaxation sweep 1        3.406e-001     4.23
3     Coarse-grid correction    3.632e+000     4.23
3     Relaxation sweep 1        1.007e+000     4.30
4     Coarse-grid correction    1.313e+001     4.30
4     Relaxation sweep 1        3.423e+000     4.55
5     Coarse-grid correction    1.179e+002     4.55
5     Relaxation sweep 1        3.971e+001     5.55
CYCLE  1   CONVERGENCE FACTOR = 0.033
##################### CYCLE #2 ######################
LEVEL ACTION                    ERROR NORM    WORK
5     Initial                   3.971e+001     5.55
5     Relaxation sweep 1        1.584e+001     6.55
5     Relaxation sweep 2        7.647e+000     7.55
4     Initial                   4.535e+000     7.55
4     Relaxation sweep 1        2.360e+000     7.80
4     Relaxation sweep 2        1.635e+000     8.05
3     Initial                   1.291e+000     8.05
3     Relaxation sweep 1        7.766e-001     8.11
3     Relaxation sweep 2        5.736e-001     8.17
2     Initial                   4.439e-001     8.17
2     Relaxation sweep 1        2.515e-001     8.19
2     Relaxation sweep 2        1.479e-001     8.20
1     Initial                   7.935e-002     8.20
1     Relaxation sweep 400      0.000e+000     9.77
```

```
2     Coarse-grid correction    1.079e-001     9.77
2     Relaxation sweep 1        3.743e-002     9.78
3     Coarse-grid correction    3.899e-001     9.78
3     Relaxation sweep 1        1.060e-001     9.84
4     Coarse-grid correction    1.228e+000     9.84
4     Relaxation sweep 1        3.180e-001    10.09
5     Coarse-grid correction    6.290e+000    10.09
5     Relaxation sweep 1        2.175e+000    11.09
CYCLE  2   CONVERGENCE FACTOR = 0.055
###################### CYCLE #3 ######################
LEVEL ACTION                    ERROR NORM    WORK
5     Initial                   2.175e+000    11.09
5     Relaxation sweep 1        1.014e+000    12.09
5     Relaxation sweep 2        5.740e-001    13.09
4     Initial                   3.827e-001    13.09
4     Relaxation sweep 1        2.189e-001    13.34
4     Relaxation sweep 2        1.583e-001    13.59
3     Initial                   1.268e-001    13.59
3     Relaxation sweep 1        8.092e-002    13.66
3     Relaxation sweep 2        6.139e-002    13.72
2     Initial                   4.785e-002    13.72
2     Relaxation sweep 1        2.725e-002    13.73
2     Relaxation sweep 2        1.583e-002    13.75
1     Initial                   8.372e-003    13.75
1     Relaxation sweep 400      0.000e+000    15.31
2     Coarse-grid correction    1.180e-002    15.31
2     Relaxation sweep 1        4.049e-003    15.33
3     Coarse-grid correction    4.234e-002    15.33
3     Relaxation sweep 1        1.135e-002    15.39
4     Coarse-grid correction    1.201e-001    15.39
4     Relaxation sweep 1        3.090e-002    15.64
5     Coarse-grid correction    4.576e-001    15.64
5     Relaxation sweep 1        1.572e-001    16.64
CYCLE  3   CONVERGENCE FACTOR = 0.072
CYCLE  4   CONVERGENCE FACTOR = 0.084
CYCLE  5   CONVERGENCE FACTOR = 0.093
CYCLE  6   CONVERGENCE FACTOR = 0.098
CYCLE  7   CONVERGENCE FACTOR = 0.103
CYCLE  8   CONVERGENCE FACTOR = 0.105
CYCLE  9   CONVERGENCE FACTOR = 0.103
CYCLE 10   CONVERGENCE FACTOR = 0.109
CYCLE 11   CONVERGENCE FACTOR = 0.111
CYCLE 12   CONVERGENCE FACTOR = 0.106
```

1.6 Full Multigrid (FMG) algorithm

The multigrid cycles described above can be applied to any first approximation given at the finest grid. In a full multigrid (FMG) algorithm, the

first approximation is obtained by interpolation from a solution at the next coarser grid, which has previously been calculated by a similar FMG algorithm. With such a first approximation, and provided that the interpolation correctly corresponds to the error norm used, one multigrid cycle should suffice to solve the fine-grid equations to the level of truncation errors (see §7). A typical FMG algorithm, with one V cycle per refinement, is shown in Fig. 1.2. FMG algorithms are less sensitive than the multigrid cycles. That is, in many irregular cases the asymptotic convergence factor of the cycles is not good, but an FMG algorithm with one cycle per refinement still guarantees solution to the level of truncation errors. In fact, even this guarantee is not necessary: from differences between the final solutions at different meshsizes (e.g., differences between the solutions at the doubly-circled stages in Fig. 1.2), one can directly calculate the rate of convergence to the *differential* solution, which is all that really matters.

1.7 General warnings. Boundary conditions. Nonlinearity

Attempts to extend existing multigrid software often fail. For example, almost everyone who tries to extend the program of §1.5 from Dirichlet to Neumann boundary conditions, first obtains a much slower solver. Some would then hastily announce that the method is inherently slower for general non-Dirichlet boundary conditions. Others, realizing the conceptual generality of the basic multigrid approach, would ask themselves what caused the slowness and how to correct it. They will eventually discover that various additional processes should be done in case of non-Dirichlet conditions, such as relaxation sweeps over these conditions and transfers of their residuals to coarser grids, concurrently with the corresponding interior processes. It will take some more effort to realize that the best relaxation of boundary conditions in multigrid solvers is often quite markedly different from the scheme one would use in relaxation solvers. Eventually, when everything is done correctly, the algorithm will regain the efficiency it showed in the Dirichlet case. Indeed, with proper treatment, the multigrid efficiency should never depend on boundary conditions, only on the interior equations (and in fact, only on the interior smoothing rates, hence only on the *factors* of the subprincipal *determinant* of the interior operator – see §3.7).

Imagine now someone trying to extend the simplest program and write a multigrid solver for the steady-state compressible Navier-Stokes or Euler equations. Here he has a *multitude* of new features: non-Dirichlet boundary conditions is just one of them, and by no means the most difficult one. Others are: non-symmetry; non-linearity; anisotropy; non-ellipticity, in fact a challenging mix of PDE types; boundary singularities (e.g., trailing edges); discontinuities (boundary layers, shocks); global conditions (Kutta

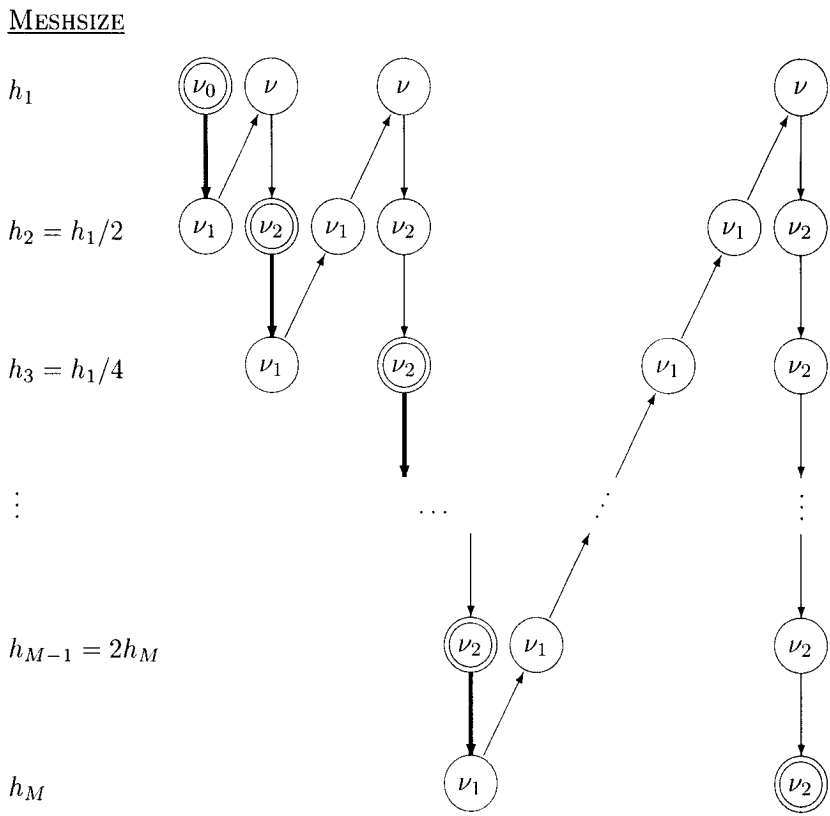


Figure 1.2. ***FMG Algorithm with one*** $\mathrm{V}(\nu_1, \nu_2)$ ***cycle per level.*** *The grids are pictured upside down relative to Fig. 1.1 (both ways are common in the multigrid literature).*

is the solution interpolation to a new grid.

is the coarse-to-fine ($k-1$ to k) interpolation of correction (1.19).

is the fine-to-coarse ($k+1$ to k) transfer. f^k is defined by (1.17) and u^k is set to 0.

stands for ν_i relaxation sweeps. At the coarsest grid $\nu = \nu_1 + \nu_2$ or somewhat larger ν_0 are usually used, or the equations are solved directly.

shows the stage in the algorithm where the final solution is obtained for the corresponding meshsize.

condition, total mass, etc.); unbounded domains; complex geometries; three dimensions; not to mention physical instabilities and turbulence.

Each and every one of these features requires a thorough understanding as to what it implies in terms of each multigrid process. Mistreating just one of them may cause the solution time to increase very significantly, sometimes by orders of magnitude. Because convergence will often still be obtained, due to the corrective nature of the fine-grid relaxation, one may be misled to believe that nothing is wrong. Even if he suspects errors, he is unlikely to find them all, because they confusingly interact with each other, an impossible network of conceptual mistakes with programming bugs.

As stated in the Introduction, we believe that every problem should be solvable in just few work units. But only a systematic development is likely to produce this top performance. Each feature should first be separately studied in the framework of as simple a problem as possible. The solver should be then constructed step by step, adding one feature at a time, making sure that full efficiency is always maintained. This Guide may help in the process.

It may be useful to add here a preliminary remark about nonlinearity. The multigrid processes are *not* inherently linear. The basic idea described in §1.1, namely, the relations between slow convergence, smallness of residuals, and error smoothness, has nothing to do with linearity. Multigrid can thus be applied *directly to nonlinear problems*, as efficiently as to the corresponding linearized problems (see §8). Hence, repeated linearizations are neither required nor advised; they are especially wasteful in case the nonlinear problem is autonomous (as is usual in fluid dynamics), because the linearized problems are not autonomous (see more detailed arguments in §8.3). Moreover, the multigrid version developed for nonlinear cases, called FAS, is useful in many other ways. In particular, it gives a convenient way to create non-uniform adaptable discretization patterns, based on the interaction between the levels and therefore very flexible, allowing fast local refinements and local coordinate transformations, with equations being still solved in the usual multigrid efficiency (see §9). Using FAS, one can integrate into a single FMG process various other processes, such as continuation, design and optimization, solution to eigenvalue problems and to inverse problems, grid adaptation, etc.

Part I

Stages in Developing Fast Solvers

The intention of this part is to organize existing multigrid approaches in an order which corresponds to actual stages in developing fast multigrid solvers. Each section (§2 through 7) represents a separate stage. To get an overview of these stages, the reader may first go through the opening remarks of all sections, skipping the subsections. The actual sequence of development may correspond to the actual order of the sections; but §5, 6, and 7.4 represent three independent stages, which can be taken in any order following §4. In fact, an increasing current tendency is to replace the usual two-level mode analysis (§5) by the two-level FMG mode analysis (§7.4). Generally, one can skip a stage, risking a lesser control over potential mistakes. Even when one does, the information and advice contained in the corresponding subsections are still important.

This part emphasizes the *linear* solver: relaxation of nonlinear equations is described, but the Full Approximation Scheme (FAS) used in intergrid transfers of nonlinear solvers is deferred to the next part (§8).

Chapter 2

Stable Discretization

The formulation of good discretization schemes is of course the first step in any numerical solution of continuous equations. For multigrid solutions, some additional considerations enter. First, discrete equations should be written for general meshsizes h, including large ones (to be used on coarse grids). Also, the multigrid processes offer several simplifications of the discretization procedures.

(A) Only *uniform-grid discretization* is often needed. Non-uniform discretization can be effected through multigrid interactions between uniform grids (see §9). In fact, if the basic grids are *non*-uniform, various structural and algorithmic complications are introduced in multigrid, as well as and even more than, in unigrid processes. The uniform discretization can be made either with finite element or finite difference formulations. For finite elements, it is preferable to use either piecewise uniform partitions as in [Ban81] and [Bra77a, §7.3], or uniform partitions modified at special parts (e.g., at boundaries) as in [Bra79a, Fig. 2], and produce local refinements by the multigrid process (§9). When general non-uniform partitions must be used, *algebraic* multigrid solvers (see §13.1) are recommended; their set-up processing (coarse-levels assembly) is much more costly, but it is still small compared with the very expensive processing required for assembling the (finest-level) equations.

(B) Only *low-order (first or second order)* discretization need to be developed. Higher-order schemes can later be superposed for little extra programming effort and computer time (§10). The low order makes it easier to write stable equations, easier to devise and analyze relaxation schemes, and less expensive to operate those schemes. The superposed higher-order schemes need not be stable on their own.

(C) Designing stable discretization with efficient relaxation is only required in terms of the *(sub)principal part of the operator* (see §2.1).

(D) An easier implementation and more flexible control of *global constraints* is achieved by effecting them only at the coarsest levels of the multigrid processing (with suitable inter-grid transfers. See §5.6). This makes it possible to free local differencing from complicated forms aimed at precise conservation of global quantities (such as total kinetic energy and square vorticity, as in [Ara66]), and to add suitable controls to ill-posed problems, etc.

Low order finite elements on uniform grids yield in fact different kinds of difference equations. The description below will therefore be in terms of finite difference formulations only. It should be emphasized, however, that variational formulations, where appropriate, automatically yield good prescriptions for the main multigrid processes [Bra77a, App. A.5], [Nic77], [Bra79a]. This is especially useful in some complicated situations, as in [ABDP81]. We revisit this issue in §§4.5, 4.6 and 11.

For a boundary-value problem to be solvable by a fast multigrid algorithm, its discretization should be suitably stable. More precisely, the type of stability determines the type of multigrid algorithm that can be efficient. *The simplest difference equations of regular elliptic PDEs are stable in every respect, so the reader interested only in such problems can skip the rest of this chapter.* But remember:

Unstable (or very inaccurate) discretization must lead to slow multigrid convergence (unless special techniques are adopted to the case, or algebraic multigrid is used). It is indeed an important *advantage* of multigrid solvers that bad discretization cannot be passed unnoticed; it must show up as slow algebraic convergence.

2.1 Interior stability measures: h-ellipticity

Numerical stability is a *local* property, i.e., a property significant only for non-smooth components of the solution (components that change significantly over a meshsize), whereas the smooth-component stability depends, by consistency, on the *differential* system, not on its discretization. Indeed, in multigrid solvers, stability of the discrete operator is needed only in the *local* process of relaxation (cf. §10.2). Moreover, what really counts is not the stability of the static difference operator itself, but the overall efficiency with which the dynamic process of relaxation smoothes the differential error (cf. §12); numerical stability of the operator is just a necessary condition for achieving that smoothing.

Because of the local character of the required stability (corresponding to the local task of relaxation), it is easy to get a very good quantitative idea about it, for any given difference operator L^h, by **local mode analysis**, analogous to the Von-Neumann stability analysis for time-dependent

problems. It turns out, however, that for steady-state problems, especially non-elliptic or singular-perturbation ones, the distinction between stable and unstable discrete operators is not enough. More important is the *measure* of stability. When that measure, for a given meshsize, is small, the scheme is still formally stable, but its actual behavior can be intolerably bad (see example in [Bra81a, §3]).

Briefly, the basic relevant measure of stability of an interior (not at boundaries) linear difference operator L^h with constant coefficients is its **h-ellipticity measure** $E^h(L^h)$, defined for example by

$$E^h(L^h) := \frac{\min_{\hat{\rho}\pi \le |\underline{\theta}| \le \pi} \left| \tilde{L}^h(\underline{\theta}) \right|}{|L^h|}, \tag{2.1}$$

where the complex function $\tilde{L}^h(\underline{\theta})$ is the "symbol" of L^h, i.e.,

$$L^h e^{i\underline{\theta}\cdot\underline{x}/\underline{h}} = \tilde{L}^h(\underline{\theta}) e^{i\underline{\theta}\cdot\underline{x}/\underline{h}};$$

$\underline{\theta} := (\theta_1, \ldots, \theta_d)$; $\underline{\theta}\cdot\underline{x}/\underline{h} := \theta_1 x_1/h_1 + \cdots + \theta_d x_d/h_d$, $|\underline{\theta}| := \max\{|\theta_1|, \ldots, |\theta_d|\}$; d is the dimension; h_j is the meshsize in direction x_j; and $|L^h|$ is any measure of the size of L^h, e.g., $|L^h| = \max_{\underline{\theta}} |\tilde{L}^h(\underline{\theta})|$. The constant $0 < \hat{\rho} < 1$ is in fact arbitrary, but for convenient multigrid applications a natural choice is the meshsize ratio, hence usually $\hat{\rho} = \frac{1}{2}$. The range $\hat{\rho}\pi \le |\underline{\theta}| \le \pi$ is then the range of "high frequency" components on grid h, i.e., components $e^{i\underline{\theta}\cdot\underline{x}/\underline{h}}$ which on the next coarser grid, with meshsize $h/\hat{\rho}$, coincide (alias) with lower components.

For systems of equations, L^h and $\tilde{L}^h(\underline{\theta})$ are matrices and $|\tilde{L}^h(\underline{\theta})|$ should then be understood as a measure of the non-singularity of $\tilde{L}^h(\underline{\theta})$ (e.g., its smallest eigenvalue, or its determinant). See more details and explanations in [Bra80b, §3], [Bra81a, §3.1]. In fact, good discretization schemes can generally be arrived at by requiring $\det(L^h)$ to be a good discretization for $\det(L)$, the determinant of the given differential operator: see the examples in §17.2, 18.2, 19.2, 20.2. We will use the notation $\tilde{\det} L^h(\underline{\theta})$ to denote the symbol of the operator $\det(L^h)$.

In case the differential operator L, and hence also L^h, have **variable coefficients**, L^h is called h-elliptic if $E^h(L^h) = O(1)$ for each combination of coefficients appearing in the domain. If L is **nonlinear**, L^h is called h-elliptic if its linearizations around all approximate solutions encountered in the calculations are h-elliptic.

A major simplification in selecting the discretization scheme for complicated systems is the fact that, being interested in local properties only, we can confine our considerations to those terms which are locally important. In the discretized and linearized operator L^h, the locally important terms, called the **h-principal terms**, are simply those with large coefficients (relative to other coefficients, if any). In case of a system, the h-principal terms are those contributing to the h-principal term of $\det(L^h)$. Other terms are

not important in relaxation; namely, they need not satisfy any stability conditions, they can actually be transferred to the right hand-side of the relaxed equations, and they need not even be updated each sweep (only each multigrid cycle; cf. §10.3).

The h-principal terms all normally come from discretizing **subprincipal terms** of L. These are defined as the principal terms (the terms contributing to the highest order derivatives in the determinant of the linearized operator) plus the principal terms of the reduced operator (the operator without singular perturbation terms). Thus, *in discretizing any differential operator L, we can confine our attention to its subprincipal part.* See the examples in §19.1 and §20.1. Note however that on very coarse grids, terms corresponding to lower-order derivatives may become h-principal.

Regular discretizations of elliptic systems should, and usually do have good (i.e., O(1)) h-ellipticity measures (but see a counterexample in §17.2). Singular perturbation or non-elliptic systems can also have such good measures, e.g., by using artificial viscosity or by upwind (upstream) differencing (note that a regular elliptic system with lower-order terms may be a singular-perturbation problem on a sufficiently coarse grid).

If, however, characteristic or *sub*characteristic directions (i.e., characteristic directions of the reduced equations, in the case of singular perturbation problems) coincide with grid directions, upwind differencing schemes are only semi h-elliptic. That is, they have a bad h-ellipticity measure E^h, yet they still have a good *semi* h-ellipticity measure in the characteristic direction, defined as follows.

Let $S \subseteq \{1, \ldots, d\}$ be a subset of grid directions. The measure of **semi h-ellipticity in directions** S, or briefly **S-h-ellipticity**, of a difference operator L^h, is

$$E^h_S(L^h) = \frac{\min_{\hat{\rho}\pi \le |\underline{\theta}|_S \le \pi} \left| \tilde{L}^h(\underline{\theta}) \right|}{L^h}, \tag{2.2}$$

where $|\underline{\theta}|_S := \max_{j \in S} |\theta_j|$. Full h-ellipticity is the special case $S = \{1, \ldots, d\}$. If $S_2 \subset S_1$, then clearly $E^h_{S_1} \le E^h_{S_2}$, hence S_1-h-ellipticity entails S_2-h-ellipticity.

In case (sub)characteristics are aligned with grid directions, full h-ellipticity is not needed for stability. The corresponding S-h-ellipticity is enough; it allows large local oscillations perpendicular to the characteristics, but those oscillations are also allowed by the differential equations.

Fully h-elliptic approximations can be constructed even for non-elliptic or semi-elliptic differential equations, by using isotropic artificial viscosity. In various cases, however, semi h-elliptic approximations are preferable, because they entail much less cross-stream smearing. These are mainly cases of **strong alignment**, that is, cases where (sub)characteristic lines are non-locally (i.e., for a length of many meshsizes) aligned with a grid-line, and where this non-local alignment occurs either for many gridlines, or even for one gridline, if that line is adjacent to a boundary layer or a

similar layer of sharp change in the solution (for a method to obtain strong alignments and thus avoid smearing – see §9.3).

A convenient way of **constructing** h-elliptic and semi h-elliptic operators is by term-by-term R-elliptic or semi R-elliptic approximations [Bra79b, §5.2], [BD79, §3.6]. Another, more physical, way is to regard the given boundary-value problem as a limit of an elliptic problem (usually this is physically so anyway), and enlarge the elliptic singular perturbation to serve as artificial-viscosity terms [Bra82c], [Bra81a]. When solving a steady-state problem of originally time-dependent equations, the artificial elliptic terms should conform to the original time-dependent problem, i.e., with those terms that problem should still be well posed. This requirement often determines the sign of the artificial terms. Such physical artificial viscosity terms ensure that computed solutions will exhibit (as $h \to 0$) only those discontinuities allowed physically (hence, with this approach, explicit entropy conditions are not needed). A proper amount of anisotropic artificial viscosity gives the correct upstream differencing whenever desired.

The desirable amount of artificial viscosity (either isotropic or anisotropic) is mainly determined not by stability considerations, but by the smoothing properties of relaxation. Below a certain level of viscosity, more costly, distributive relaxation will have to be used. Increasing the artificial viscosity slightly larger beyond the minimum required for convergence of the simplest scheme makes the relaxation ordering-free (see §3.6 and [Bra81a, §5.7], [Bra80b, §4.2]), which is desirable, except perhaps near discontinuities. Considerably larger artificial viscosity makes the algebraic smoothing faster, but impedes the differential smoothing (cf. §12).

Interior difference equations that are not even semi h-elliptic should be used with care. Their solutions may show large numerical oscillations (giving nice solutions only on the average), and their fast multigrid solvers must have more complicated fine-to-coarse interactions (see for example §4.2.2). Some **quasi-elliptic** equations, i.e., cases where $\tilde{L}^h(\underline{\theta})/|L^h|$ does vanish for some $|\underline{\theta}| = \pi$, but not for other $|\underline{\theta}| \neq 0$, can be solved without much trouble. All that is needed is to average out the bad components (see for example §§18.6, 19.5).

2.2 Boundaries, discontinuities

We have so far discussed the stability conditions related to the *interior* difference equations, away from boundaries. To gain overall stability, some additional conditions should be placed at the boundaries. These can be analyzed by mode analysis in case the boundaries are parallel to grid directions (cf. §7.5). More general boundaries are however difficult to analyze.

Usually, however, h-elliptic approximations consistent with a well-posed problem and employing low-order approximations to boundary conditions, are stable. The order can be then raised in a stable way by one of the methods of §10. At any rate, the boundary stability is not related to

the stages of developing the main (interior) multigrid processes.

More critical than discretization near boundaries is the treatment of *discontinuities*, whether at boundaries (e.g., boundary layers) or in the interior (e.g., shocks). The basic rule, in multigrid as in unigrid processes, is to try not to straddle the discontinuity by any difference operator (during relaxation as well as in residual transfers. The rule also applies to the interpolation operators). More precisely, the rule is not to difference any quantity which is discontinuous in the interval of differencing. This can be fully achieved only in cases where the location of the discontinuity is known or traced (at such discontinuities the above rule overrides upstream differencing if they happen to conflict), or when the discontinuity is more or less parallel to grid directions (so that upstream differencing will automatically satisfy the rule). Captured discontinuities that are not in grid directions must perhaps be smeared; a multigrid way to get high accuracy then is by local refinements (see §9). Generally, multigrid procedures for discontinuities are now under active investigation (see [Bra81a, §4] and a remark in §8.5 below).

Chapter 3

Interior Relaxation and Smoothing Factors

The crucial step in developing multigrid solvers is the design of interior relaxation schemes with high error-smoothing rates. Namely, the crucial question is how to reduce non-smooth error components for as little computational work as possible, neglecting interactions with boundaries. This is the crucial question, from the point of view of solution efficiency, because reducing *smooth* error components will require less computational work (being done at coarser grids), and because reducing non-smooth error components *near boundaries* will require much less work as it involves local work only near the boundary (which is a lower-dimensional manifold). Also, relaxation is the most problem-dependent part of the algorithm – other parts are usually quite standard (the relaxation of boundary conditions is discussed in §5.3).

3.1 Local analysis of smoothing

To reduce non-smooth error components is basically a *local* task; it can be done in a certain neighborhood independently of other parts of the domain. This is why it can be efficiently performed through relaxation, which is basically a local process (the information propagates just few meshsizes per sweep). Hence also, the efficiency of this process can accurately be measured by local mode analysis.

That is, one can assume the problem to be in an unbounded domain, with constant (frozen) coefficients, in which case the algebraic error $u^h - \tilde{u}^h$ (where u^h is the exact solution to the discrete equations and $\tilde{u}^h$ is the computed approximation) is a combination of Fourier components $e^{i\underline{\theta}\cdot\underline{x}/\underline{h}}$. For each such Fourier component and any proposed relaxation scheme, one can easily calculate the amplification factor $\mu(\underline{\theta})$, defined as the factor by which the amplitude of that component is multiplied as a result of a

relaxation sweep (see simple examples in §1.2 above and in [ST82, §3.2]). The **smoothing factor** $\overline{\mu}$ of the relaxation scheme, defined by

$$\overline{\mu} = \max_{\frac{\pi}{2} \le |\underline{\theta}| \le \pi} |\mu(\underline{\theta})| \tag{3.1}$$

can then be easily computed, usually by a standard computer program (see for example [Wei01]). This is indeed the measure we need: $\overline{\mu}$ is the worst-case (largest) factor by which all high-frequency error components are reduced per sweep, where we define the frequency to be high if the component is not visible (aliases with a lower component) at the next coarser grid (grid $2h$).

In case of a **system** of q grid equations in q unknown grid functions (i.e., q unknowns and q algebraic equations are defined per mesh cell), each Fourier amplitude is a q-vector, hence $\mu(\underline{\theta})$ is a $q \times q$ amplification *matrix*. $\overline{\mu}$ is still defined as in (3.1), except that $|\mu(\underline{\theta})|$ is replaced by $\rho(\mu(\underline{\theta}))$, where $\rho(\mu)$ is the spectral radius of μ.

For L^h with **non-constant coefficients**, $\overline{\mu}$ defined by (3.1) depends on the location. In case of a **nonlinear** L^h, the analysis is made for the linearized operator, hence $\overline{\mu}$ also depends on the solution around which linearization is made. The quality of relaxation is then determined by the worst $\overline{\mu}$, i.e., the maximum $\overline{\mu}$ over all possible coefficients of L^h for any solution which may evolve in the calculations (one may disregard $\overline{\mu}$ over small regions: see §3.3).

A major simplification in calculating $\overline{\mu}$ for complicated systems is to look at **subprincipal terms only** (see §2.1 and 3.4, and examples in §19.2 and 20.2).

Some relaxation schemes do not transform each Fourier component of the error to a multiple of itself. Instead, they couple several (l, say) Fourier components at a time (even for an infinite domain). For example, if relaxation is performed in red-black (checker-board) ordering (cf. §3.6), the $\underline{\theta}$ component is coupled to the $\underline{\theta} + (\pi, \ldots, \pi)$ component. Instead of the $q \times q$ amplification matrix $\mu(\underline{\theta})$, we then have the $(ql) \times (ql)$ matrix $\mu(\underline{\theta}^1, \ldots, \underline{\theta}^l)$, describing the transformation of the l q-vector amplitudes corresponding to the coupled components $(\underline{\theta}^1, \ldots, \underline{\theta}^l)$. Definition (3.1) is extended to such cases by *defining*

$$\overline{\mu}_\nu := \max \left[\rho \left(C \left(\underline{\theta}^1, \ldots, \underline{\theta}^l \right) \mu \left(\underline{\theta}^1, \ldots, \underline{\theta}^l \right)^\nu \right) \right]^{\frac{1}{\nu}}, \tag{3.2}$$

where the max is taken over all coupled l-tuples $(\underline{\theta}^1, \ldots, \underline{\theta}^l)$, C is an $l \times l$ matrix of $q \times q$ blocks C_{ij}, such that $C_{ij} = 0$ for $i \neq j$, $C_{ii} = I_q$ (the $q \times q$ identity matrix) if $|\underline{\theta}^i| \geq \frac{\pi}{2}$, and $C_{ii} = 0$ otherwise. ν is the number of sweeps performed at the finest grid per multigrid cycle; only in the simple case ($l = 1$), $\overline{\mu}$ does not depend on ν. For examples with $l > 1$, see [Bra81b, §3.3], [Lin81, §2.3.1].

The smoothing factor is the first and simplest quantitative predictor of the *obtainable* multigrid efficiency: $\overline{\mu}^\nu$ (or $\overline{\mu}_\nu^\nu$) is an approximation to the asymptotic convergence factor obtainable per multigrid cycle. Usually this prediction is more accurate than needed. There are still more accurate predictors (see §4.1). But *the main importance of $\overline{\mu}$ is that it separates the design of the interior relaxation from all other algorithmic questions. Moreover, it sets an ideal figure against which the performance of the full algorithm can later be assessed* (see §§4, 5).

The analysis of relaxation within multigrid is thus much easier than its analysis as an independent iterative solver. The latter is not a local process, and its speed depends on smooth components badly approximated by mode analysis due to boundaries and variable coefficients. For multigrid purposes, however, wherever the equations (or their linearized version) do not change too much within few meshsizes, the smoothing factor can be used as a standard measure of performance. A general computer program for calculating $\overline{\mu}$ is described in [Wei01].

3.2 Work, robustness and other considerations

In comparing several candidate relaxation schemes we should of course take into account not only their smoothing factors, but also the amount of work per sweep. The aim is generally to have the best high-frequency convergence rate per operation, i.e., the largest $w_0^{-1}\log(1/\overline{\mu})$, where w_0 is the number of operations per gridpoint per sweep. But other considerations should enter as well: The rate should be robust, that is, $\overline{\mu}$ should not depend too sensitively on problem parameters or on a precise choice of various relaxation parameters. Also, between two schemes with similar values of $w_0^{-1}\log(1/\overline{\mu})$ but with very different w_0, the simpler scheme (where w_0 is smaller) should be preferred, because very small factors $\overline{\mu}$ cannot fully be obtained in practice (owing to the inability of the coarse-grid correction to obtain such small factors for the smooth components, and owing to interactions with boundaries). Moreover, large values of w_0 leave us with less flexibility as to the amount of relaxation work to be performed per cycle. Very small $\overline{\mu}$ may in fact be below what we need in the Full Multigrid (FMG) algorithm (see §7).

An important consideration, sometimes overlooked, is that each relaxation sweep should of course be **stable**. The most familiar schemes *are* stable, but distributive schemes (§3.4) for example, can be unstable exactly in cases showing the best $\overline{\mu}$. A trivial example: satisfy each difference equation in its turn by changing its *latest* unknown (in the sweeping ordering) instead of its usual corresponding unknown. $\overline{\mu}$ will then vanish, but the process will be unstable. Stability analysis can in each case be performed as Von-Neumann analysis for time dependent problems, taking the main relaxation marching direction as the timelike direction.

Also, let us not forget that relaxation has a certain **effect on smooth**

(low frequency) components, too. Usually this effect is slow: $\mu(\underline{\theta})$ is close to 1 for small $|\underline{\theta}|$. But sometimes schemes which show spectacularly small values of $\overline{\mu}$ also show either bad divergence ($|\mu(\underline{\theta})| \gg 1$) or fast convergence ($|\mu(\underline{\theta})| \ll 1$) for *low* frequencies. This for example may happen in relaxing hyperbolic (relative to some time-like direction) equations using upstream differencing and marching with the stream (the time-like) direction. Schemes with bad divergence should clearly be rejected (see an example in §20.3.4, the super-fast smoothing case). Those with fast convergence may also have some disadvantage (in case high-order corrections, as in §10.2, are desired; see for example [Bra81a, §2.2]).

It is therefore advisable to add to the program of calculating $\overline{\mu}$ also a routine for checking the stability of the scheme examined, and to calculate, together with (3.1), also the value of $\max_{|\underline{\theta}|\leq\pi} |\mu(\underline{\theta})|$. It is also useful to calculate weighted mean squares of $\mu(\underline{\theta})$ for high-frequency $\underline{\theta}$'s. Such quantities predict the error decrease in a given number of multigridded relaxation sweeps for a given initial error [BD79, §4.5]. Some of these measures are listed in [Wei01].

The value of local mode analysis becomes dubious at places of strong discontinuities, e.g., where the coefficients of the differential equation change their order of magnitude discontinuously (or within few meshsizes). This usually happens along manifolds of lower dimensionality, therefore more computational work per gridpoint can there be afforded, hence an accurate measure of efficiency is not so needed. But some basic rules, outlined below, must still be followed. One can also employ local R–E analysis (cf. §1.1 and more details in [Bra86]), which yields good quantitative information even in strongly discontinuous cases.

3.3 Block relaxation rule. Semi smoothing

The most basic rule in devising relaxation schemes is that *a locally strongly coupled block of unknowns which is locally decoupled from (or weakly coupled with) the coarser-grid variables, should be simultaneously relaxed.* The reason is that a point-by-point relaxation smoothes only along the strongest couplings, whereas block relaxation also smoothes along second-strongest couplings (provided the strongest ones are included in the blocks. See for example the case $a \ll c$ in §1.2, and a more general analysis in [Bra86, §3.5, 4.6].

This rule is of course important whether or not the equations are continuous. In the case of **persistent S_1-h-ellipticity** (i.e., a difference operator with good S_1-h-ellipticity throughout a substantial subdomain, but without uniformly good S_2-h-ellipticity measure for any $S_2 \not\subset S_1$) , the rule implies either the use of block relaxation in suitable directions (line relaxation, plane relaxation, etc.), or the use of suitable "semi coarsening" (see below), or both.

Generally, for any set S of grid directions (usually $S \subset \{1, 2, \ldots, d\}$

, but sometimes including a special bisecting direction could be advantageous), a relaxation scheme is called an **S-block relaxation** if it relaxes simultaneously all (or many contiguous) equations defined on the same S-subspace. (Two points $(x_1, \ldots, x_d)$ and $(y_1, \ldots, y_d)$ are in the same S-subspace if $x_j = y_j$ for all $j \notin S$.) For example, the line relaxation in §1.2 is a y-block (vertical line) relaxation.

Semi coarsening, or more specifically **S-coarsening**, means that $H_j = 2h_j$ for $j \in S$, and $H_j = h_j$ otherwise, where H_j and h_j are the mesh-sizes of the coarse grid and the fine grid, respectively, in the x_j direction $(j = 1, \ldots, d)$. In such a coarsening, we need to smooth the error only in directions S. The definition of the smoothing factor should accordingly be modified. We generalize (3.1) to any coarsening situation by defining

$$\overline{\mu} := \max \left\{ \rho(\mu(\underline{\theta})) : \ |\underline{\theta}| \le \pi, \max_{1 \le j \le d} \frac{|\theta_j| H_j}{h_j} \ge \pi \right\}. \tag{3.3}$$

Similarly we generalize (3.2) by defining $C_{ii} = I_q$ if $\max |\theta_j^i| H_j / h_j \ge \pi$, and $C_{ii} = 0$ otherwise. The **S-smoothing factor** is defined as (3.3), or the generalized (3.2), for S-coarsening. (An improved $\overline{\mu}$ definition that allows the constant π in (3.3) to be replaced by a smaller positive number, is explained in §12.)

If *point* (not block) relaxation is to be used, then S-h-ellipticity defined in §2.1 is a necessary and sufficient condition for the existence of relaxation schemes with good (i.e., bounded away from 1) S-smoothing factors. This is an easy generalization of a theorem proved in [Bra80b, §4.2]. The more general situation, with *block* relaxation, is summarized by the following theorem:

Theorem 3.1. *Let S and S' be two sets of directions: $S, S' \subset \{1, \ldots, d\}$. A necessary and sufficient condition for the existence of an S-block relaxation scheme with good S'-smoothing rates is that the discrete operator L^h is uniformly coupled in S' modulo S; that is,*

$$E^h_{S',S}(L^h) := \min_{\substack{\frac{\pi}{2} \le |\underline{\theta}|_{S'} \le \pi \\ \theta'_j = \theta_j \text{ for } j \in S}} \frac{\tilde{L}^h(\underline{\theta})}{\tilde{L}^h(\underline{\theta}')} = O(1). \tag{3.4}$$

$E^h_{S',S}(L^h)$ is called the measure of uniform coupling in S' modulo S. The theorem states, in other words, that the S'-smoothing factors, produced from L^h by any *suitable* S-block relaxation, are bounded away from 1 by a quantity which depends only on $E^h_{S',S}(L^h)$.

In this context, the role of block relaxation can sometimes be played by simple point relaxation. This is when the relaxation marching direction conforms with the downstream time-like direction of a hyperbolic-like system, so that a relaxation sweep nearly *solves* the equations (especially

when upstream differencing is used). The role of block relaxation can also be played, more automatically and in more situations, by ILU smoothers (see §3.8).

Variable coefficients. When the coefficients of L^h (or of its linearization, in case it is nonlinear) are not constant, a *perfect smoother* is a relaxation scheme whose formal $\overline{\mu}$ (calculated at each point by assuming the coefficients there to extend as constant coefficients throughout) is good at all points. Such a perfect smoother can sometimes require quite costly block relaxation (e.g., plane relaxation) in several directions, because of varying semi-h-ellipticity directions. It is therefore important to realize that *such perfect smoothers are not really needed* because *accidental* semi h-ellipticity need not be taken into account. Explanation:

First, the above block-relaxation rule itself suggests that $\overline{\mu}$ may be allowed to be bad (close to 1) at some isolated points.

The multigrid convergence rates will still be good. More importantly, however, consider the common case of a (nearly) non-elliptic differential operator whose (sub)characteristic directions continuously vary over the domain, so that in some particular small region they happen to approach some grid directions. As a result, in that particular region the discretization may be semi h-elliptic, smoothing there will be bad, hence the asymptotic multigrid convergence will slow down. Notice, however, that the components slow to converge are very special ones: They are necessarily high-frequency characteristic components in that particular region; i.e., components smooth in the (sub)characteristic directions but not smooth in all directions. Such components exist on the grid *only* when (sub)characteristic directions approach grid directions. Elsewhere, such components are not represented at all by the finite-difference solution; they are truncated. Hence, if one is interested in solving the discrete equations only to the level of truncation errors (and hence using an FMG algorithm – cf. §7), such components can be ignored: Their slow algebraic convergence in regions of *accidental* semi h-ellipticity does not matter, because similar components are not approximated at all in other regions.

Only in cases of **strong alignment** (see §2.1) the corresponding block relaxation must be used. But it may be confined to the region of strong alignment. If, for example, the strong alignment is due to grid alignment of boundary layers, it is enough to perform line (or plane) relaxation only at the very lines adjacent to such boundaries (and sometimes not even there – see [Bra81a, §3.3]), with just point relaxation elsewhere. If the alignment is strong because it occurs throughout a major subdomain, line (or plane) relaxation of only that special direction is needed there. Alternating-direction block schemes may be needed only if errors far below truncation errors are for some reason desired.

A general way to avoid any need for block relaxation, even when solving far below truncation errors, is to use semi coarsening, as mentioned above. In the case of variable coefficients, causing variable coarsening di-

rections, this leads to AMG processes (see §13.1).

3.4 Distributive, weighted, collective and box Gauss-Seidel. Principal linearization

To obtain efficient smoothing, a selection should be made from an abundance of available relaxation schemes. The choice depends on experience and on some physical insight, with $\overline{\mu}$ calculations serving for final quantitative judgement. We list here some important types of schemes. Each of those can be operated pointwise or blockwise (see §3.3) and in different orderings (§3.6). Some simple schemes are described in more detail in [ST82]. We first describe successive displacement schemes, then we mention their simultaneous-displacement counterparts (§3.5). In the case of equations with many or complicated linear or nonlinear lower-order terms, it may pay to apply any of these schemes with the scaled principal terms only (see §10.3). In the case of complicated *systems*, see the general approach in §3.7.

The most basic scheme is the **Gauss-Seidel** (GS) scheme , in which all the discrete equations are scanned one by one in some prescribed order. Each of them in its turn is satisfied by changing the value of one corresponding discrete unknown. This is easy to do if the problem is linear and if there is a natural one-to-one correspondence between equations and unknowns (i.e., if the matrix of coefficients is definite or is approximately definite; see §6.3). If the problem is **nonlinear**, each discrete equation may be a nonlinear equation in terms of the corresponding unknown. It is then usually best to make just one Newton step toward solving each equation in its turn. This **Gauss-Seidel-Newton** (GSN) scheme is not related to any global linearization of the system of equations, it just linearizes one discrete equation in terms of one discrete unknown, yielding usually a very simple scheme that does not require any storage other than the storage of the (approximate) solution.

Principal linearization. Moreover, it is actually enough to relax an equation through an approximate linearization of its h-principal terms, corresponding to the (sub)principal terms of the differential operator (§2.1). Thus, for example, if the equation is $\mu\Delta u + uu_x + \cdots$ and the current approximation just before relaxing at some point is $\tilde{u}$, then relaxation at that point can simply be the same as if relaxing the equation $\mu\Delta u + \tilde{u}u_x + \cdots$. A full linearization would in addition include the term $(u - \tilde{u})\tilde{u}_x$, but on the scale of the grid, hence in relaxation, that term is negligible. This "principal linearization" is *as good as full linearization for purposes of relaxation*, at least as long as differences of $\tilde{u}$ at adjacent gridpoints are small compared with $\tilde{u}$ itself. Note that for quasi-linear equations, which include almost all practical equations, the principal linearization involves *no linearization at all, just trivially retarding the lower-order derivatives in each term*, as in the example.

When relaxation is used as the prime solver, much may be gained by Successive Over Relaxation (SOR), in which the GS correction calculated for each unknown is multiplied by a **relaxation parameter** ω. The situation is different when relaxation is used only as a smoother in multigrid solvers. The best smoothing (lowest $\overline{\mu}$) is usually obtained for the natural value $\omega = 1$, so that GS is not only cheaper (per sweep), but also at least as effective (per sweep) as SOR. Lower $\overline{\mu}$ may be obtained by other parametrizations (e.g., the distributive GS described below), but for regular second-order elliptic equations this gain hardly justifies the extra work involved: Simple GS is probably the best known smoother (especially with red-black ordering – see §3.6).

If block relaxation is required (cf. §3.3), **block GS** can be used. This means that blocks are scanned one-by-one; the equations of each block are simultaneously satisfied by updating the corresponding block of unknowns. In the two-dimensional plane (x, y), if the blocks are lines parallel to x (constant-y lines), the relaxation is called x-Line GS (xLGS). yLGS is similarly defined (see example in §1.2).

When there is no natural one-to-one correspondence between discrete equations and unknowns (the matrix is not approximately definite; e.g., non-elliptic and singular perturbation equations, or elliptic *systems* which are not strongly elliptic [BD79, §3.6]), simple GS should be replaced either by **Distributive Gauss-Seidel** (DGS) or by Weighted Gauss-Seidel schemes. In DGS, with each discrete equation we associate a "ghost" unknown, with some prescription being selected for the dependence of regular unknowns on ghost unknowns. Usually, each regular unknown is written as a prescribed linear combination of neighboring ghost unknowns. Then, as in GS, the equations are scanned one by one, each being satisfied by changing the corresponding ghost unknown. This means in practice that a certain pattern of changes is distributed to *several* neighboring regular unknowns (hence the denomination "distributive" GS); the ghost unknowns do not explicitly appear, nor stored in any way, they just serve for the description of DGS. (In fact their values are never known – only changes in their values are calculated to induce changes in the regular unknowns.) In the case of block (e.g., line) DGS relaxation, a block of ghost unknowns is simultaneously changed to simultaneously satisfy the corresponding block of equations. In two dimensions we thus have xLDGS and yLDGS schemes. A special case of DGS is the Kaczmarz relaxation (see §1.1). Other examples are described in detail in §17.3, 18.3, 19.3 and 20.3. The smoothing analysis of DGS schemes is best executed in terms of the ghost unknowns; see §3.7.

In **Weighted GS** (WGS) schemes, with each discrete unknown we associate a ghost equation, which is a preassigned linear combination of neighboring equations, and we perform GS in terms of the ghost equations. Taking work into account, WGS is usually inferior to DGS, since each equation is calculated several times per sweep, unless the ghost equations

explicitly replace the original equations – which is just a linear transformation of the discrete system of equations. It pays to transform the system (as against performing DGS) only if the resulting system is not more complicated than the original, which is seldom the case: A transformation that yields a simpler system could usually be done in terms of the *differential* equations, giving a simpler differential system. (Exceptions are cases where the simplifying transformation gives a worse system for discretization; e.g., a system not in conservation form as in [Bra82c, §2.1]. To relax in conservation form in that case, a combination of WGS and the DGS scheme of [Bra82c, §4.1] is indeed needed.)

For systems of equations ($q > 1$), simple GS is appropriate only in case the system is strongly elliptic [BD79, §3.6]. Otherwise collective GS or DGS schemes should be employed. **Collective Gauss-Seidel** (CGS) is performed when the grid is not staggered, i.e., all the q grid equations and q unknown functions are defined on the same gridpoints: The grid points are scanned one by one, at each point we change simultaneously ("collectively") its q unknowns so as to simultaneously satisfy its q equations. In case of a staggered grid, one can divide the domain into (usually overlapping) small boxes. The boxes are scanned, for each one we change simultaneously all unknowns interior to it so as to simultaneously satisfy all equations interior to it. This is called **Box GS** (BGS). DGS schemes are generally more efficient for staggered grids than BGS (except sometimes in very coarse grids; cf. §6.3), because they do not couple the equations (see §3.7).

For *nonlinear* equations, all these methods can be used, but instead of fully satisfying an equation (or a collective of q equations, or a box of equations), only one Newton step (or just principal linearization) is made in terms of the corresponding (regular or ghost) unknown (or collective of q unknowns, or box of unknowns). For semi h-elliptic cases, block CGS (e.g., line CGS, meaning simultaneous solution of all equations on a line through changing all that line's unknowns), or block DGS, or block BGS, may be performed (after principal linearization, if needed) .

Higher-order equations are sometimes most efficiently relaxed by writing them as systems of lower order equations. For example, the biharmonic can be written as a pair of Poisson equations. Relaxing this system involves less work (per complete sweep) and yields better smoothing (per sweep) than relaxing the biharmonic. But special care should be taken in relaxing the boundary conditions for this system (see §5.3).

3.5 Simultaneous displacement (Jacobi) schemes

The GS schemes described above are *successive-displacement* schemes: The new value of an unknown (or block of unknowns) replaces the old one as soon as it is calculated, and is immediately used in relaxing the next

equations. In *simultaneous displacement* schemes new values replace old ones only at the end of the sweep, after all of them have been calculated; hence each of them is calculated explicitly in terms of old values only. Corresponding to each of the schemes above we have a simultaneous-displacement scheme, called: Jacobi-relaxation, Jacobi-Newton, distributive Jacobi, weighted Jacobi, collective Jacobi, box Jacobi, line Jacobi, line distributive Jacobi, etc. – corresponding to GS, GSN, DGS, WGS, CGS, BGS, line GS, line DGS, etc., respectively.

Unlike GS, Jacobi schemes often require **under-relaxation** ($\omega < 1$) in order to provide good smoothing. But with relaxation as a smoother (not an independent solver), good and optimal values of w are independent of the domain, and can easily be calculated by local mode analysis.

Distributive and weighted Jacobi (under-)relaxation amounts actually to the same thing. An example of an optimized weighted Jacobi scheme is analyzed in [Bra77a, §3.3].

Experience so far shows that Jacobi schemes are inferior to the corresponding GS schemes. They not only require more work (for operating the relaxation parameter) and more storage (for storing the new values separately), but their smoothing factors are in fact worse. For the 5-point Poisson equation, for example, Jacobi under-relaxation ($\omega_{\text{optimal}} = .8$) yields $\overline{\mu} = .6$, while GS gives $\overline{\mu} = .5$ and .25 for lexicographic and red-black orderings, respectively. The situation is similar in all cases so far examined. The advantage of simultaneous displacement schemes is in their being more amenable to certain rigorous analyses (but there usually seems to be little practical value to this – see §14) and their vectorizability and parallelizability (but red-black GS and similar schemes are also fully parallelizable – see §3.6).

3.6 Relaxation ordering. Vector and parallel processing

For successive-displacement schemes, the order in which the equations (or blocks of equations) are relaxed has an important effect on the smoothing factors. The main orderings used are the usual **lexicographic** (LEX) order (in which the equation at grid point $(i_1, \ldots, i_d)$ is relaxed before $(j_1, \ldots, j_d)$ if $i_k = j_k$ for $1 \leq k < l$ and $i_l < j_l$), and related orders (LEX order for some permutation of the coordinates, some of them possibly reversed); **symmetric** relaxation (lexicographic sweep followed by a sweep in the reversed order); **Red-Black** (RB) ordering (in which all "red" gridpoints are relaxed before all "black" ones, where the coloring is similar to that of a checkerboard, namely a point $(i_1, \ldots, i_d)$ is red if $i_1 + \cdots + i_d$ is odd, and black if it is even); and more general **pattern relaxation** (similar to RB, but with different coloring and possibly more colors). For difference equations involving more than nearest neighbors, RB schemes still depend

on the ordering of points within each color. If such points are displaced simultaneously, the scheme is called Jacobi-RB; similarly LEX-RB, etc. The performance (i.e., smoothing per operation) of Jacobi-RB is more like GS schemes than like Jacobi, and like them it does not generally require underrelaxation.

Each of these orderings has its block-relaxation versions. xLGS (or xLDGS) can be done lexicographically forward (increasing y) or backward (decreasing y), or symmetrically (forward alternating with backward). Or, corresponding to RB, we can first relax the even lines, then the odd lines. This is called **zebra** xLGS (or x-zebra) relaxation. Similarly, yLGS (or yLDGS) can be done upward, downward, symmetrically or zebra. Particularly robust schemes are the Alternating-Direction Zebra (ADZ = x-zebra alternating with y-zebra) and Alternating-Direction Symmetric LGS (ADS = symmetric xLGS alternating with symmetric yLGS). Many more block GS schemes are similarly defined in higher dimensions. The choice of blocks is governed by the rule in §3.3. Concerning the choice of ordering we have the following remarks.

It has been found that GS with RB ordering is the best for the 5-point Poisson equation [FST81]. Similarly, DGS with RB ordering within each of its passes (called briefly Distributive RB, or DRB) is the best for many *systems*, such as Cauchy-Riemann and compressible and incompressible Navier-Stokes equations (see §17.3, 18.3, 19.3 and 20.3). For 5-point Poisson, RB-GS provides $\overline{\mu}_1 = .25, \overline{\mu}_2 = .25$ and $\overline{\mu}_3 = .32$ (cf. (3.2)), as opposed to $\overline{\mu} = .5$ for LEX-GS. Moreover, RB-GS can be executed with only four operations per grid point, whereas lexicographic GS requires five. Similar comparisons hold for the more complicated elliptic systems.

In addition, the mentioned RB schemes (more precisely Jacobi-RB) and similar pattern relaxation schemes are fully **vectorizable and parallelizable**: All the equations of the same color can be relaxed in parallel, thus taking full advantage of computers having vector or parallel processing capabilities. The zebra schemes are similarly parallelizable. (See more about parallelization of all multigrid processes in [Bra81b].)

For non-elliptic equations or for elliptic equations with large non-isotropic lower-order terms (singular perturbation problems, in particular), the first approach [Bra76], [SB77], [Bra77a], [Bra79b] was to employ **"downstream" ordering**, in which the equation at a point A is relaxed before (or simultaneously with) that at point B if the solution at B depends more heavily on the solution at A than vice-versa (e.g., if the fluid flows, or the convection transports, from A to B). This provides very good smoothing factors (better than those for regular elliptic problems). If different "downstream" directions exist at different parts of the domain, this may require a sequence of several relaxation sweeps in several directions. If for example line relaxation is also required, ADS relaxation may be needed, i.e., four passes over the domain. Each pass may be effective in only part of the domain, but the combined sweep will give excellent smoothing everywhere,

for any combination of semi h-elliptic approximations in two dimensions (and also in three dimensions, if the grid is coarsened in only two directions (cf. §4.2.1. In some particular cases (when the reduced equation is hyperbolic in some time-like direction, and upstream differencing is employed) such schemes yield not only great smoothing but also great convergence, making coarse-grid corrections superfluous.

Our preference today, however, is away from these downstream marching schemes. First, because they are not so good for vector and parallel processing. Also, because in the cases where several downstream directions are required, the programming is complicated and the multi-direction procedure is not fully efficient, since it requires several passes over the grid where one or two (efficient) passes is all that would be needed at each multigrid stage. Hence **ordering-free** schemes were developed, with which good smoothing is obtained for any ordering, including RB and/or zebra (the block-relaxation rules should still be kept). Such ordering-free schemes are obtained either by distributive relaxation [Bra79b, §6], or by using slightly more artificial viscosity than that required for upstream-differencing [Bra80b, §4.3], [Bra81a, §5.7, 6.3, 7.2]. Actually, even these devices (distributive relaxation and/or increased artificial viscosity) are not usually needed, unless one wants a "perfect smoother" in order to reduce algebraic errors far below truncation errors: If all that matters is the fast approximation of the *differential* (not the discrete) solution and an FMG algorithm is employed (see §7), then the simplest direction-free schemes, such as RB, can do [Bra81a]. Moreover, the tendency of downstream marching schemes to yield fast *convergence* (not just fast smoothing) may sometimes be disadvantageous (see §3.2).

3.7 Principle of relaxing general PDE systems

To obtain a good smoother (i.e., a good combination of discretization and relaxation, yielding good *differential* smoothing, as defined in §12) for a given *system* of q partial differential equations $Lu = f$, it is important to understand in advance what smoothing rates are obtainable. The guiding principle here is the following.

The smoothing rate for a given PDE operator L can be as good as the smoothing rates obtainable for the factors of the subprincipal part of $\det(L)$.

Many examples are given in Part III of this Guide (e.g., study §17.3 before proceeding to the general case that follows). To explain this generally, we first show how a smoother for L can be constructed in terms of a smoother for the scalar operator $\det(L)$. One way to do it is through distributive relaxation (cf. §3.4). Such a relaxation is defined by considering the vector of unknown functions u (or their discrete counterparts) to affinely depend on a "ghost" vector of functions w, i.e., $u = Mw + u^0$, where M is a $q \times q$ matrix of differential (or finite-difference) operators, and the vector u^0 is immaterial (since we are interested in *changes* in w,

through which *changes* in u are defined; w itself is not explicitly used). In terms of w we then devise a suitable relaxation for the product operator LM. It is easy to see that the smoothing rate of this relaxation in w will automatically be taken over by the resulting distributive relaxation in u.

(Note that a variable coefficient and a derivative do not generally commute. However, interchanging their order does not change the principal part of the operator. So in fact, it is only in terms of principal or subprincipal parts that the determinant of a matrix operator and its factorization are well defined.)

One particular choice is to take M to be the transposed matrix of cofactors of L, in which case LM equals $\det(L)$ times the $q \times q$ identity operator. One can thus devise for each component of w any relaxation suitable for $\det(L)$; the corresponding distributive relaxation for u will have the same smoothing rate.

As mentioned earlier (§2.1), the only part of L which really participates in devising the smoother is the *subprincipal* part of the linearized operator; the smoothing rates obtained for L are the same as for that part. Thus, for the discussion here, we may think of L as having subprincipal terms only. In that case we can often factor $\det(L)$ into simpler factors. Typically, in many physical problems, the factors are either the Laplacian Δ or the convection-diffusion operator $\Delta + \underline{a} \cdot \underline{\partial}$. Smoothing rates for the latter are discussed for example in [Bra81a], [Ket82] and [ST82].

One general way to devise the relaxation of L in terms of the factorization of $\det(L)$ is to correspondingly factorize L itself, using the following theorem.

Theorem 3.2. *If* $\det(L) = l_1 l_2$, *where each* l_i *is a (scalar) differential operator, then one can factorize the* $q \times q$ *operator* L *into* $L = L_1 L_2$, *where* L_i *are* $q \times q$ *matrix operators such that* $\det(L_i) = l_i$.

The proof, for which we thank Anthony Joseph, is based on Theorem 5 on page 393 in [Lan65]. A nice example is the factorization of the elasticity operator

$$\begin{pmatrix} \mu\Delta + \lambda\partial_{xx} & \lambda\partial_{xy} \\ \lambda\partial_{xy} & \mu\Delta + \lambda\partial_{xy} \end{pmatrix} = \begin{pmatrix} \partial_x & \partial_y \\ \partial_y & -\partial_x \end{pmatrix} \begin{pmatrix} \lambda + \mu & 0 \\ 0 & \mu \end{pmatrix} \begin{pmatrix} \partial_x & \partial_y \\ \partial_y & -\partial_x \end{pmatrix}$$

that is indeed useful for its relaxation (through the scheme of §17.3), especially in case $\mu \ll \lambda$, where simpler schemes fail.

To relax the factorized system $L_1 L_2 u = f$, one can simply introduce the auxiliary vector of unknown functions $v = L_2 u$ and alternating relax the two systems: $L_1 v = f$ and $L_2 u = v$. The combined smoothing rate is asymptotically no worse than the worst of the rates of the two systems. (If these two rates are very different, the system with the slower rate can be relaxed more times [TOS00, App. C], [LB04].) Special care should of course be exercised in relaxing near and on boundaries (see §5.3).

In many cases there is a simpler distributive relaxation which meets the goal of the above guiding principle. It is not necessary that LM be diagonal as in the general approach above; it is enough to get LM to be triangular. Moreover, if the operators on one of the columns of M all have a common divisor, that divisor can be omitted. In this way one can often have each term on the diagonal of LM to be just one separate factor of $\det(L)$ (some factors possibly appearing in more than one diagonal term), in which case no auxiliary functions (such as v above) are needed. The relaxation schemes in §17–20 are all of this kind. Also, instead of distributive relaxation one can obtain the same goal by weighted relaxation schemes. The important upshot in any case is that thanks to the above guiding principle, *the goal is known in advance, so do not settle for any substantially slower rates.* Note that the above smoothing discussion assumes frozen operators, hence may not apply to very coarse grids.

3.8 ILU smoothers

The above list of relaxation schemes, although including some of the most efficient smoothers, does not exhaust all possibilities. Of special interest is the use of incomplete LU decomposition (ILU), and related schemes, as smoothers. Such smoothers, first introduced in [WS80], have been shown to be very robustly efficient for a wide range of 5-point and 9-point difference equations. For an extensive treatment, see [Ket82] and the more recent [SWd85].

The basic ILU process can be described as a Gaussian elimination truncated so as to preserve a certain pattern of sparsity, simply by ignoring (i.e., replacing immediately by zero) any term produced by the elimination process at any matrix position designed to remain zero (e.g., any matrix position which is originally inherent zero). In case of nonlinear equations one can apply this process with the principal linearization (see §3.4).

The robustness of the ILU smoother can be explained by its ability in many cases to automatically produce an approximation to desired block relaxations in varying grid directions. If for example the system contains any sequence of unknowns each of which is strongly coupled only to its predecessor and successor in the sequence, and if the ILU ordering of unknowns conform with the ordering of that sequence (i.e., it gives that sequence upon omitting all other unknowns), then, ignoring weak couplings, the equations of the sequence appear to the ILU process as a separate tridiagonal system, which it automatically solve simultaneously (since Gaussian elimination for a tridiagonal system does not produce new non-zero terms). This is exactly what the block relaxation rule (§3.3) requires. Provided the weak couplings do not somehow accumulate and spoil this picture. There are special situations where that may happen. Indeed, for some special anisotropic equations there are some very special high-frequency error components which are even considerably *magnified* by the simple ILU process. More advanced

"block ILU" (see [Ket82, §3.2.3]) should then be used.

One can produce systems where various sequences of strong couplings are so ordered as to contradict any ordering chosen for the ILU process. But this rarely happens in practice; especially in two dimensions, such systems are artificial. Thus, using ILU we need to worry much less about all directions of relaxation required for a perfect smoother. Note however that such a perfect smoother is usually needed only for solving far below truncation errors (see §3.3). A careful comparison, in which the total amount of operations in an FMG algorithm is counted taking into account the ILU set-up operations shows ILU schemes to be quite comparable to suitable GS schemes [Tho83]. Moreover, in many cases ILU requires much more storage: One needs to store all the non-zero matrix elements, whereas the GS schemes often require storing just the approximate solution itself (e.g., when the problem, whether linear or not, is autonomous, as in all fluid dynamics cases). Also, GS schemes (in red-black ordering) are much faster on vector machines.

Both an advantage and a disadvantage is the fact that ILU is a "package deal", automatic prescription: It tells you exactly what to do in complicated situations, near boundaries for example; but it does not allow you to change the scheme to deal with special needs, such as local relaxation near reentrant corners and other singularities [BB87], other special local treatments, separate smoothing of coupled differential equations, etc. One can however combine ILU smoothers with sophisticated distributive schemes. For example, within the distributive relaxation described in §19.3 for the Navier-Stokes equations, one can use ILU for relaxing each set of momentum equations ((19.4b) for one j) in terms of the corresponding velocity function u_j^h.

Chapter 4

Interior Two-Level Cycles

Having computed the smoothing factor $\overline{\mu}$, one should expect the asymptotic convergence factor per multigrid cycle to approach $\overline{\mu}^{\nu}$, where ν is the number of relaxation sweeps (on the fine grid h) per cycle. This ideal figure does not take into account the exact nature of the inter-grid transfers. The next task then is to design those transfers so as to approach the ideal figure. To separate their design from questions related to boundary conditions (which are taken up at the next chapter), we still think in terms of fully-periodic or full-space problems; we still, that is, restrict our attention to *interior* processes, because it is there that most of the computational work is invested. Furthermore, we simplify the multigrid situation at this stage by restricting our attention to two grids only, the finest grid $h = (h_1, \ldots, h_d)$ and the next coarser grid $H = (H_1, \ldots, H_d)$, where usually $H = 2h$. That is, we assume in our analysis that the grid-H equations are solved (exactly) each time the algorithm gets to that grid, without analyzing how and how expensively that solution is obtained, hence without involving grids coarser than H in the analysis.

These assumptions indeed simplify our studies very much. First, the error can be expanded in a Fourier integral (or series) and the transformations of the amplitudes of different Fourier components by multigrid operations can be calculated. Indeed, for linear systems with constant coefficients only few Fourier components at a time are coupled to each other by these two-level interior processes, hence transformations of Fourier amplitudes are expressed as small matrices (§4.1). In case of nonconstant coefficients, we usually freeze them at some values (treated then as parameters of the analysis). In case of nonlinear equations, their Newton linearization is analyzed (although no such linearization is needed in the actual processes; see §8.3). The parameters of the analysis then depend on the solutions around which linearization is made.

This freezing of coefficients is reasonable as long as the real coefficients do not change too drastically over a meshsize. Where they do, we can sometimes model them as changing periodically, again making mode analysis with small matrices possible [BD79, §4.7]. See more about the rigor of this analysis in §14.

When mode analysis becomes too difficult or dubious, or if one simply wishes to skip it, **experiments with periodic boundary conditions** can be used instead. One can in fact do such experiments even when mode analysis *is* available, and *compare* the analysis with the experiments. This is an accurate debugging technique, completely separating away issues related to boundary conditions. Moreover, such periodic-boundary-condition program could serve as an excellent preliminary stage in developing your real multigrid program. For this preliminary program, the various advices given in §5.1 concerning the full program could be used, including in particular the use of *multi*grid program to simulate a *two*-grid algorithm, and the trick of near-linearization of non-linear equations.

Some warnings, however, concerning this use of periodic boundary conditions: First, relaxation should better be restricted to simultaneous-displacement (Jacobi, red-black Jacobi, zebra Jacobi, etc.) schemes, to avoid the special non-smoothness created along the starting line (or termination line) of the relaxation sweep. (This non-smoothness would be particularly bad if downstream ordering were used.) Second, periodic boundary conditions often require additional global conditions to be added so as to make the problem well posed. In some cases these extra conditions are easy to implement, involving for example just an adjustment of the solution average by adding a constant. (Multigrid treatment of global conditions is discussed in §5.6.) Other cases, especially nonlinear ones, are less straightforward. A full section could, and perhaps will, in fact be written about the art of using periodic problems. It is also important to realize that interior studies in general have more limited value for non-elliptic problems, where the interplay with boundaries is more essential (see end of §4.1).

One should also make sure that at this stage (whether mode analysis or periodic numerical experiments are used) both grid h and grid H are fine enough, and grid-H equations are solved accurately enough (without taking into account the work this accurate solution requires), in order to separate away questions related to coarser grids (see §6.3). Do not forget, however, in the process of optimizing the grid-H operator L^H, that this is a modeling for a multigrid solution, hence your model must be recursive: The H equations should have the same general form as the original h equations, with the same range of possible parameters.

In addition to the relaxation scheme, studied above, the main issues to be studied at this interior-two-level stage are when to switch (under what criteria, or after how many relaxation sweeps) from grid h to grid H; what should be the coarse-grid variables; and the type (in the interior) of three multigrid operators: The fine-to-coarse transfer of residuals I_h^H, the coarse

grid operator L^H, and the coarse-to-fine interpolation of corrections I_H^h. These issues are one-by-one discussed in the subsections below. They are later reviewed again, from a more general perspective, in §11. Relevant to these issues are also §8.5 and §10.2 (nonlinear problems and higher-order techniques).

4.1 Two-level cycling analysis. Switching criteria

Details of the two-level mode analysis are described in [BD79, §4.6–4.8] and in [ST82, §§3.3–3.5, 7, 8, 9]. The former also discusses modifications of the analysis to account for the fact that in practice the grid-H equations are only approximately solved, modification for the case of equations with highly oscillatory coefficients, and ways to make precise comparisons between mode analysis and numerical experiments (for debugging purposes). The main things to know are the following.

On grid $2h$ the Fourier mode $\exp(i\underline{\theta} \cdot \underline{x}/\underline{h})$ aliases (coincides with) the mode $\exp(i\underline{\theta}' \cdot \underline{x}/\underline{h})$ if $|\theta_j - \theta'_j| = 0$ or π for $j = 1, \ldots, d$. Hence each set of so aliasing components usually includes 2^d components $\{\underline{\theta}^1, \ldots, \underline{\theta}^{2^d}\}$, called **harmonics** of each other. They are coupled to each other by the two-level processes. (The special sets with less than 2^d different components do not require special analysis, since they are limits of regular sets.)

We define the **two-level cycle** as follows: Make ν_1 relaxation sweeps on grid h, then transfer the residual problem to grid H and solve it there exactly, then interpolate that grid-H solution to grid h and add it as a correction to the former grid-h solution, then make ν_2 more relaxation sweeps on grid h. It is easy to see that in the infinite space, if L^h, L^H, I_h^H and I_H^h are all constant operators, and if the error in the solution before such a cycle has the form $\sum_j A_j \exp(i\underline{\theta}^j \cdot \underline{x}/\underline{h})$, where the sum is over a set of 2^d harmonics, then the error after the cycle will have a similar form, and the new A_j's will be linear combinations of the old ones. If we deal with a system of q grid equations then each amplitude A_j is a q-vector, hence the overall transformation of the 2^d amplitudes by the two-level cycle is a $(2^d q) \times (2^d q)$ matrix M, which can be denoted $M(\underline{\theta})$ where $\underline{\theta}$ is the lowest harmonic ($|\underline{\theta}| \leq \frac{\pi}{2}$).

This matrix $M(\underline{\theta})$ is called **the two-level amplification matrix**. The easiest and most modular program for calculating it is to write a different routine for the general matrix-element of each of the five involved processes: relaxation, L^h, I_h^H, L^H and I_H^h. Their respective matrices $\check{S}^h$, $\check{L}^h$, $\check{I}_h^H$, $\check{L}^H$ and $\check{I}_H^h$ have dimensions $(2^d q) \times (2^d q)$, $(2^d q) \times (2^d q)$, $q \times (2^d q)$, $q \times q$, and $(2^d q) \times q$, respectively, and each of their elements is a function of $\underline{\theta}$. Then program

$$M(\underline{\theta}) = \left(\check{S}^h\right)^{\nu_2} \left[I - \check{I}_H^h \left(\check{L}^H\right)^{-1} \check{I}_h^H \check{L}^h\right] \left(\check{S}^h\right)^{\nu_1}. \tag{4.1}$$

The main performance measure of the two-level cycle is the **two-level asymptotic convergence factor** (per cycle)

$$\overline{\lambda} := \max_{|\underline{\theta}| \leq \frac{\pi}{2}} \rho\left(M\left(\underline{\theta}\right)\right) , \tag{4.2}$$

where $\rho(M)$ is the spectral radius of M. Note that $\overline{\lambda}$ depends on the sum $\nu = \nu_1 + \nu_2$, but not on the separate values of ν_1 and ν_2. In fact, when many cycles are performed the separate values are immaterial. Various other performance measures can similarly be defined. (See [ST82, §3.4-3.5], where the notation M_h^{2h} and $\rho(M_h^{2h})$ is used for our M and $\overline{\lambda}$, respectively. Additional two-level measures will be discussed in §7.4.)

The two-level analysis is used to (roughly) optimize the involved processes; namely, the objective is to maximize $w^{-1}\log(1/\overline{\lambda})$, where $w = A(\nu w_0 + w_1 + w_2)$, w_0 is the work in one relaxation sweep, w_1 is the work of calculating and transferring the residuals, w_2 is the work of the I_H^h interpolation, and A is a factor through which the work on coarser grids is taken into account. For our objective here the value of A is really immaterial, but to have a definite value in later uses (see §6.2) we observe that for V cycles we can assume similar operations on each of the grids, hence $A = (1 - \hat{\rho}_1 \cdots \hat{\rho}_d)^{-1}$, where $\hat{\rho}_j = h_j/H_j$ (usually $\hat{\rho}_j = .5$) while for W cycles $A = (1 - 2\hat{\rho}_1 \cdots \hat{\rho}_d)^{-1}$. To avoid the laborious count of operations and the arbitrary assignment of proper weights to different arithmetic and non-arithmetic operations (which are really machine-dependent), one can use the work of a standard relaxation sweep as the work unit. In complicated problems, where calculating L^h outweighs interpolations, one can then neglect w_2 and take $w_1 = 1$ for full residual weighting and $w_1 = 2^{-d}$ for residual injection. The convergence factor per work unit is then denoted by $\mathring{\mu} = \overline{\lambda}^{1/w}$. As above (§3.2), in addition to the goal of minimizing $\mathring{\mu}$ we should take robustness and simplicity into account.

One can also partly separate the study of I_h^H, L^h and I_H^h from that of relaxation by the **Coarse-Grid Correction (CGC) mode analysis**, as in [Bra77a, §A.1]. But this is not simpler than the full two-level analysis, especially since relaxation schemes have already been selected in the previous stage. We use a CGC analysis in §4.3 below.

The ideal factor $\overline{\lambda} = \overline{\mu}^{\nu}$ is not always obtainable. If $\overline{\mu}^{\nu}$ is too small we will get $\overline{\lambda} > \overline{\mu}^{\nu}$, because of significant high-frequency amplitudes generated from low ones by interpolation or by RB-type relaxation (see §4.3). Even when obtainable, too small values of λ will require too precise interpolations, hence too much investment in w_1 and w_2, and will at a later stage be frustrated by other interactions (boundaries and non-constant coefficients). Also, such small $\overline{\lambda}$ will not usually be needed in the final FMG algorithm (see §§7.2-7.3). Hence, the optimal cycle always employs a small ν, typically $\nu \leq 3$.

In regular elliptic problems $\nu = 1$ is too small to be optimal (unless the sweep includes several passes, as in symmetric and alternating-direction

schemes), since the overhead of w_1 and w_2 weights too much against it. Hence usually the optimal number is $\nu = 2$ for very efficient smoothers ($\overline{\mu} \leq .3$ or so), and $\nu = 3$ otherwise. A small change in ν does not disturb the overall efficiency very much. Considerably larger ν are less efficient, because they bring the process into the range of larger feeding from low to high frequencies, while not much more is gained in reducing the overhead (already at $\nu = 3$, $w_1 + w_2$ is quite small compared with νw_0).

A possible approach is **accommodative**: do not fix ν in advance, but continue relaxation as long as it exhibits the fast convergence of high frequencies, e.g., as long as the convergence factor (some norm of the residuals divided by the same norm a sweep earlier) is smaller than the smoothing factor $\overline{\mu}$. For non-scalar ($q > 1$) systems, such a criterion can separately be applied to each equation, possibly resulting in more passes for part of the equations. Similarly it may separately be applied at different subdomains (since smoothing is a local process), possibly giving partial relaxation sweeps.

In the case of non-elliptic and singular perturbation problems there are some particular smooth error components (smooth characteristic components of the differential operator or the reduced differential operator) for which L^H is a bad approximation to L^h, hence $\overline{\lambda}$ cannot be much smaller than .5, no matter how small $\overline{\mu}^\nu$ is [Bra81a, §5.1], [Bör81]. But for exactly the same components and the same reason, L^h itself is not a good approximation to the differential operator L. Hence, exactly for these components, we do not *need* much *algebraic* convergence (convergence to the discrete solution), since the discrete solution itself is far from the differential solution. Hence, for such cases the asymptotic convergence factor $\overline{\lambda}$ is not really the measure we need.

The one we need is obtained by the two-level FMG analysis (see §7.4). Moreover, for non-elliptic or singular perturbation problems the usual assumption that high-frequency components are local does not hold. It is violated by high-frequency characteristic components in cases of strong alignment (§2.1). The interior mode analysis should then be supplemented with a half-space analysis (§7.5).

4.2 Choice of coarse grid

When the fine grid, with meshsize $h = (h_1, \ldots, h_d)$, is given, the choice of a coarse grid, with meshsize $H = (H_1, \ldots, H_d)$, is often straightforward: Take every other line (every other hyperplane, for $d > 2$) of the fine grid in each direction. The coarsening ratio $H_j/h_j = 2$ is usually optimal: it is the smallest recursively convenient number, and it is already big enough to make the coarser-grids work quite small relative to the fine-grid work; larger H_j/h_j will not save significantly more work, but will significantly degrade the smoothing factors (see (3.3)). The smaller ratio $H/h = 2^{\frac{1}{2}}$ may be as efficient (trading larger A for smaller ν), and it is recursively

convenient in some two dimensional problems with rotatable operators; see [ST82, §2.5], [RTW83].

When the fine-grid discretizations are done in terms of "cells" with the discrete variables defined at certain cell positions (e.g., cell centers, or centers of vertical cell boundaries, etc.), and especially when the grid is **staggered** (different grid functions are defined at different cell positions), it is more convenient to coarsen in terms of the cells: Take every 2^d fine cells as a coarse cell, and then place coarse-grid variables at coarse-cell positions analogous to their positioning in the fine cells. The coarse grid points then are not a subset of the fine grid points. See examples in §17.4, 18.4 and another approach in [Den82a].

In some cases the "fine-grid" is not a well-organized grid at all; e.g., a general finite-element triangulation, not based on any grid lines. Then one can still construct the coarse grid as a uniform grid, placed over the domain with no particular relation to the fine grid. Another approach is to base the choice of coarse-grid variables on purely algebraic considerations (§13.1). Mode analysis is of course not very suitable for analyzing such situations.

4.2.1 Semi coarsening

Semi coarsening, or more specifically S-coarsening, is the technique of using grid H which is coarser than h in only a partial set S of coordinates; i.e., $H_j = 2h_j$ for $j \in S$ and $H_j = h_j$ for $j \notin S$. This means some more work is done on coarse grids; but either this or block relaxation are needed in some cases – see the rule in §3.3. Semi coarsening is sometimes preferable to block relaxation. For example, in three-dimensional problems where there are two fixed coordinates with stronger couplings, full coarsening would require plane relaxation, which is inconvenient. (Solving these plane equations approximately by one multigrid cycle, if done simultaneously at all planes, will look very much like semi coarsening.) Also, exactly in those cases, semi coarsening involves relatively small work on coarser grids, since two coordinates are still coarsened, hence the total number of points on all coarser grids is at most one third the number of points on the finest grid.

Sometimes, a combination of block relaxation and semi coarsening may be the best. For example, the equation $aU_{xx} + bU_{yy} + cU_{zz}$ with $a \ll b \ll c$, discretized on a cubic grid ($h_x = h_y = h_z$), will best be solved by z-line relaxation and $y-z$ semi coarsening. Generally, rough calculations of S-smoothing factors (§3.3) immediately show what procedures can be taken.

In some cases block relaxation is of course preferable to semi coarsening. For example, when directions of strong alignment are different at different subdomains. To change accordingly the directions of semi coarsening would be much messier than changing block directions.

4.2.2 Modified and multiple coarse-grid functions

When a difference operator L^h is given which has no good h-ellipticity or semi-h-ellipticity measure, then no relaxation can be efficient in reducing all high-frequency error components. To reduce all components efficiently we can then often use modified coarse-grid correction functions.

Suppose for example that the slow components (i.e., the components for which relaxation is inefficient) are all clustered around some known modes $\phi_j(\underline{x})$, $(j = 1, \ldots, J)$. This means that the error $v^h = \tilde{u}^h - u^h$ can be written as $v^h(x) = \sum_j v_j^h(\underline{x})\phi_j(\underline{x})$, where v_j^h are smooth functions. It is then these smooth functions which we try to approximate by coarse-grid functions v_j^{2h}. See [Bra80c, §3.2]. Sometimes, each of these functions can most efficiently be approximated on a different (e.g., differently rotated) grid [BL97].

4.3 Orders of interpolations and residual transfers

The most important aspect of the coarse-to-fine correction interpolation I_H^h and the residual transfer I_h^H is their orders, defined as follows: The order of I_H^h is m if the interpolation of the low frequency Fourier component $\exp(i\underline{\theta} \cdot \underline{x}/\underline{h})$, with amplitude 1 on the coarse grid H, creates on the fine grid h high-frequency components (the harmonics of the low frequency) with amplitudes $O(|\underline{\theta}|^m)$. It also reproduces the $\underline{\theta}$ component itself on grid h with an amplitude $1 + O(|\underline{\theta}|^m)$. The order of the fine-to-coarse transfer I_h^H is said to be m, and its secondary order $\overline{m}$, if a high-frequency harmonic with amplitude 1 on grid h contributes $O(|\underline{\theta}|^m)$ to the amplitude of the corresponding low frequency $\underline{\theta}$ when transferred to grid H, while a low frequency with amplitude 1 on grid h contributes $1+O(|\underline{\theta}|^{\overline{m}})$ to its grid-H amplitude. Thus, linear and bilinear interpolations have order 2, while cubic interpolation is fourth order. Residual transfer by injection ($I_h^H \equiv 1$) has order 0 and infinite secondary order, whereas the usual full-weighting residual transfer ((4.6) below) is of order 2 and secondary order 2.

What orders should be used in the multigrid cycle? This depends on the orders of derivatives appearing in our equations. Suppose we have a system of q differential equations in q unknown functions, and let m_{ij} be the highest order of differentiation (or differencing) of the j-*th* unknown in the i-*th* equation, $(i, j = 1, \ldots, q)$. We assume, and this is usually the case, that the q unknown functions are interpolated independently of each other and that the residuals of each of the q grid equations are transferred separately from the others. Denote by m^j the order of I_H^h used in interpolating the j-*th* correction (correction to the j-*th* unknown function) and by m_i and $\overline{m}_i$ the order and secondary order, respectively, of the I_h^H used in transferring the i-*th* residual (residuals of the i-*th* equation).

What m^j, m_i and $\overline{m}_i$ $(i, j = 1, \ldots, q)$ should be used? Examining

orders of magnitude in the CGC mode-analysis operator (the operator in brackets in (4.1)), under the assumption that all $m^j > 0$, we find the following basic rules and observations:

(A) The high-frequency harmonics of the lowest frequencies (those with $|\theta| = O(h)$), are amplified by the CGC operator by a factor with a spectral radius $1 + O\left(\sum_{i,j} h^{m_i+m^j-m_{ij}}\right)$. Hence, to avoid large magnification of high-frequencies, we should have

$$m_i + m^j \geq m_{ij}, \tag{4.3}$$

preferably even $m_i + m^j > m_{ij}$. On the other hand, larger values ($m_i+m^j > m_{ij}+1$) would not significantly further reduce the spectral radius, hence they are *asymptotically* (when many cycles are made) not needed [Bra94].

(B) Every high-frequency harmonic (before the CGC cycle) contributes to the corresponding low-frequency (after the cycle) through a $q \times q$ transformation matrix $(L^H)^{-1}B$, where $B_{ij} = O(h^{m_i-m_{ij}})$. This is usually not important asymptotically (for many cycles), but *if only one cycle is performed* (as in FMG algorithms), that transformation may cause large errors unless

$$m_i \geq m_{ij}. \tag{4.4}$$

For relaxation schemes with interactions between high and low frequencies (e.g., RB schemes), this transformation may also cause *asymptotic* degradation unless $m_i > \sum_k (m_{ij} - r_{kj})$, where $O(h^{r_{kj}})$ is the size of the high-frequency errors in the *k-th* function generated by relaxation from an $O(1)$ low-frequency error in the *j-th* function. RB and zebra schemes for $q = 1$ give $r_{11} = m_{11}$, hence the rule requires $m_1 > 0$, i.e., full weighting (see §4.4). This requirement can however be slackened by a more precise look at the nature of these particular schemes (allowing the use of simpler transfers such as the "half injection" $I_h^H \equiv .5$ or "half weighting"; see [FST81, §2], [ST82, §8.1]).

(C) The low-frequency error components themselves are reduced by a factor $O(h^{\tilde{m}})$, where $\tilde{m} := \min\{\tilde{p}, \overline{m}_1, \ldots, \overline{m}_q, m^1, \ldots, m^q\}$ and $\tilde{p}$ is the lowest of the approximation orders on levels h and H. Hence $\tilde{m}$ must be positive, which is indeed the case for any consistent differencing and interpolation schemes. Larger values of $\tilde{m}$ may of course give better cycle performance. Our experience indicates that $\tilde{m} = 2$ gives considerably better $\overline{\lambda}$ than $\tilde{m} = 1$. Since this is a low-frequency matter, hence non-local, higher $\tilde{m}$ may be effective only if they are carefully matched by corresponding high-order approximations and

interpolations at *boundaries*. But one usually does not have to go into the trouble of $\tilde{m} > 2$. Rather, employ more cycles with $\nu \leq 3$ (see §4.1). As a result, the factor $O(h^{\tilde{m}})$ will usually be dominated by $\overline{\mu}^{\nu}$ in determining $\overline{\lambda}$.

(D) We also note that every low-frequency error component (before the CGC cycle) contributes to every one of its harmonics (after cycle) through a $q \times q$ transformation matrix D, where $D_{jj} = O(h^{m^j})$ and for $i \neq j$, D_{ij} has higher orders in h. This tells us something about the range where relaxation should be efficient (see §12).

4.4 Variable operators. Full weightings

The above mode-analysis rules are insufficient in case L^h is highly-varying, i.e., its coefficients substantially change between two neighboring grid-points. For such L^h the residuals after relaxation are also highly varying, hence to represent them correctly on grid H, *full residual weighting* should be used, i.e., I_h^H should satisfy, for any residual function r^h,

$$(H_1 \cdots H_d) \sum_{x^H} \left(I_h^H r^h\right)(x^H) = h_1 \cdots h_d \sum_{x^h} r^h(x^h), \tag{4.5}$$

where x^h are the fine-grid points and x^H are the coarse-grid points. In other words, full weighting "preserves integrals". (Throughout this discussion it is assumed that the difference equations on all grids are written in their *divided* form, analogous to the differential equations. If, however, they are multiplied through by factors which depend on the meshsize, then one should not forget to have those factors in (4.5), too.) One can regard full weighting as a scheme in which each residual $r^h(x^h)$ on the fine grid is being distributed to several coarse grid points, with weights whose sum is $\hat{\rho} = h_1 \cdots h_d/(H_1 \cdots H_d)$. Hence each residual r^h is a weighted average of its transferred values on grid H , times $\hat{\rho}$. This weighted average represents a certain interpolation, $\check{I}_H^h$ say. Thus every full weighting I_h^H is the *adjoint* (or, in matrix terminology, the *conjugate transpose*) of an interpolation $\check{I}_H^h$, times $\hat{\rho}$. $I_h^H = \hat{\rho}(\check{I}_H^h)^*$. For instance, the d-dimensional analogue of the standard 9-point symmetric full weighting, defined by

$$\left(I_h^{2h} r^h\right)(x^{2h}) = \sum_{\max_j |\nu_j| \leq 1} 2^{-d-\sum |\nu_j|} r^h \left(x^{2h} + (\nu_1 h_1, \ldots . \nu_d h_d)\right), \tag{4.6}$$

is the adjoint of d-linear multivariate interpolation (the tensor product of linear interpolations along one dimension at a time), times 2^{-d}.

The requirement (4.5) is equivalent, in terms of the Fourier analysis, to the requirement that I_h^H has a positive order (see §4.3). Such full weightings should perhaps be used in almost any case. Only in some particular cases non-full weighting operators happen to be asymptotically somewhat better.

An example is injection in case of the standard 5-point Poisson operator, which yields lower $\overline{\lambda}$ as well as lower w_1 than the full weighting (4.6) (see [BD79, §4.8]). But even in those cases, for the purpose of Full Multigrid (FMG) algorithms (see §7), full weightings may be preferable. (See rule (4.4) above and [ST82, §3.6]).

4.5 Coarse-grid operator. Variational and Galerkin coarsening

The coarse grid operator L^H should be a proper homogenization of the fine-grid operator L^h. In smooth problems this is easily obtained by good discretizations of both L^h and L^H. In nonlinear problems this is effectively obtained by a suitable FAS averaging of the fine-grid *solution* (see $\check{I}_h^H$ in §8.5). Sometimes one needs to derive L^H from L^h, not from the differential operator L, either because L is not available or because one wants an automatic program, for some *general* class of L^h, or without having to treat separately boundary conditions and whatever other features of the differential problem. That is, one wants to regard the fine grid equations simply as a matrix equation

$$\underline{L}^h \underline{u}^h = \underline{f}^h \tag{4.7}$$

where the underlines signify matrix notation: $\underline{L}^h$ is an $n^h \times n^h$ matrix, where n^h is the number of unknowns on grid h. The geometry of the grids is only used to construct the inter-grid transfers I_H^h and I_h^H. (A multigrid treatment without any geometrical structures is discussed in §13.1.).

In case $\underline{L}^h$ is **symmetric**, a general way to derive L^H is to regard (4.7) as the equivalent problem of finding $\underline{u}^h$ which minimizes the functional

$$\Phi^h(\underline{u}^h) := \frac{1}{2}(\underline{u}^h)^* \underline{L}^H \underline{u}^h - (\underline{f}^h)^* \underline{u}^h, \tag{4.8}$$

where stars stand for transposing. Given now an approximate solution $\tilde{\underline{u}}^h$, with a (smooth) error $\underline{v}^h = \underline{u}^h - \tilde{\underline{u}}^h$ which is to be approximated by the coarse-grid correction $I_H^h \underline{v}^H$, the-equations for $\underline{v}^H$ are fully specified by requiring it to yield a correction which reduces Φ^h as far as possible, i.e., $\underline{v}^H$ should minimize $\Phi^h(\tilde{\underline{u}}^h + I_H^h \underline{v}^H)$. This immediately gives the coarse grid equations

$$\underline{L}^H \underline{v}^H = I_h^H (\underline{f}^h - \underline{L}^h \underline{u}^h), \tag{4.9}$$

where

$$I_h^H = (I_H^h)^* \tag{4.10}$$

and

$$\underline{L}^H = I_h^H L^h I_H^h. \tag{4.11}$$

Equation (4.9) has the general form (1.14). The specific prescription (4.10)-(4.11) is called *variational coarsening*, since it results from the variational

formulation of the problem. It is automatically determined as soon as the interpolation operator I_H^h is selected. I_h^H given by (4.10) is automatically "full" in the sense of §4.4. (By (4.11), the coarse grid equations (4.9) are not basically changed if I_h^H is multiplied by any scalar, such as $\hat{\rho}$ in §4.4.)

In case $\underline{L}^h$ is not symmetric, (4.10) is not always advisable (see §4.6), but (4.11) can generally be used. This L^H is called the *Galerkin operator*, since it is equivalent to requiring the coarse-grid correction to be a *projection*, i.e., requiring the residuals of the corrected solution $\tilde{\underline{u}}^h + I_H^h \underline{v}^H$ to vanish when transferred back to the coarse grid:

$$I_h^H \left(f^h - L^h(\tilde{\underline{u}}^h + I_H^h \underline{v}^H) \right) = 0.$$

The reason Galerkin operators and variational coarsening are not always advisable is the amount of work involved in the construction (4.11), which could be considerably larger than the entire solution work (e.g., when solving by the algorithm in Fig. 1.2). Also, once constructed, the Galerkin operator is often much more complicated than the simpler L^H derived directly from L (e.g., 9-point instead of 5-point formulae), and requires much larger storage for storing all its coefficients, whereas the simpler L^H may require no storage (e.g., whenever L is autonomous, whether linear or not; this includes all fluid dynamic equations, if they are not linearized). See more about this issue in §11.

4.6 Strongly discontinuous, strongly asymmetric operators

As long as the fine-grid operator does not vary drastically, the above rules for I_H^h, L^H and I_h^H work fine. A more difficult case is that of a strong discontinuity in L^h, i.e., where its coefficients change their *order of magnitude* within a meshsize. Orders of interpolations are not so important then; rather, special forms should be used which take into account the particular nature of the discontinuity. The rule is first to analyze the behavior, near the discontinuity, of the error which is inefficiently reduced by relaxation. This error is approximately a solution to the homogeneous equations. (If it is not, then it has large residuals and therefore there locally exists a relaxation scheme for which it will be reduced efficiently; cf. §1.1). Hence its general behavior is like that of solutions to the homogeneous differential equations. The interpolation I_H^h of corrections should take this behavior into account. For example, if we have a diffusion problem $\nabla(a\nabla u) = F$, near a strong discontinuity of the diffusion coefficient $a(x)$ the derivatives of the solution to the homogeneous equation are not continuous. Instead, $a\nabla u$ is continuous there, and this can be used to design good interpolation schemes [ABDP81]. In the case of singular perturbation or non-elliptic problems, solutions to the homogeneous equations are continuous along (sub) characteristics, hence interpolation should be as much as possible in

the characteristic directions. This is possible exactly where it is most important, namely, cases of intended strong alignment (cf. §2.1). Thus, for example, avoid interpolating across a boundary layer which is intended to be sharply reproduced.

It is less clear how to generally design the residual transfers I_h^H and the coarse grid operators L^H near a strong discontinuity. In the symmetric case the variational rules (4.10)–(4.11) are most robust [ABDP81], even though expensive. For cases which are not essentially symmetric the Galerkin operator (4.11) can still be used, but instead of (4.10) one should take

$$I_h^H = \left({}^*\underline{I}_H^h\right)^* \tag{4.12}$$

where ${}^*\underline{I}_H^h$ is an interpolation appropriate in the above sense for $(L^h)^*$, the adjoint of L^h. See [Den82b].

For non-elliptic and singular perturbation problems, the considerations and experiments in [Bra81a] indicate that improved results are obtained by a full residual weighting in which residuals, on being transferred from a fine gridpoint to a different point (or points) on the coarse grid, are transferred roughly in the downstream direction. As for correction interpolation for such problems, however, it seems that the symmetric schemes are preferable to schemes with upstream bias. Coarse grid operators identical with the fine-grid ones (hence much cheaper than (4.11)) were used, with excellent FMG results (even where the asymptotic rates were slow).

A general perspective on these questions of coarsening a problem (designing I_h^H, L^H, I_H^h) is given in §11 below.

Chapter 5

Boundary Conditions and Two-Level Cycling

The theoretical two-level mode analysis described above (§4.1), and/or the numerical experiments with periodic boundary conditions (§4), give us the ideal convergence factor per cycle ($\overline{\lambda}$), or per work-unit ($\mathring{\mu}$) . These are the *interior convergence factors*, obtained in the absence of boundary interference. The next stage is to construct an actual multigrid program for an actual, bounded domain, and in particular to decide on the special treatment the various processes should take at points near or on boundaries. The goal is to attain or approach the interior convergence factors. For elliptic problems this is generally possible, since smoothing away from the boundary is decoupled from the boundary and since the boundary neighborhood itself is a lower-dimensional set of grid points, hence we can allow there more work (per point) than in the interior, without changing the total work by much [Bra94]. The comparison to the interior factors is a very important tool in debugging the program or finding conceptual mistakes, especially mistakes in treating boundary conditions or interior equations at points adjacent to boundaries. On the other hand, approaching the interior factors is not all-important; in fact, optimal performance of the full multigrid (FMG) algorithm may well be obtained without it, especially in non-elliptic or small-ellipticity problems (see end of §4.1).

In §5.2–5.5 below we mention some rules related to the multigrid processes near or on boundaries. The general remarks of §11 and the curved-boundary treatment in §9.3 are also relevant here.

In addition to boundary conditions, some problems have global conditions. These should also be incorporated at this stage. Their multigrid implementation is discussed in §5.6.

5.1 Simplifications and debugging

It is advisable to start with a program for **rectangular domains** whose boundaries coincide with grid lines at all levels. This will make the programming much easier (the program in §1.5 can serve as a model), and will separate away various difficulties related to more general domains. In fact, the first stage in constructing such a program could be the case of periodic boundary conditions (separating away boundary considerations altogether) discussed in §4.

Having made rectangular models work satisfactorily, one can then proceed to other domains. At this point one has to decide whether to write a general-domain or a specific domain program. Experience shows **general-domain** multigrid programs to be considerably less efficient (typically requiring twice the CPU time). One can model one's general domain program after MG01 [ST82, §10], or after MUGPACK, or actually use the MUGPACK or GRIDPACK software [MUG84]. But the efficiency of this software, too, is still considerably below the efficiency of specific domain programs (where the efficiency of rectangular domains can be approached). The reason is the many checks that should be made to distinguish between various possible positions of gridpoints with regard to the boundary, especially in interpolation and residual-transfer routines, where two grids are simultaneously involved.

It is advisable to start programming **cycling algorithms**, before proceeding to the additional questions related to the full multigrid (FMG) algorithm (taken up in §7). Cycling algorithms start with some arbitrary approximation on the finest grid and reduce its error by cycling between that grid and coarser grids. At this stage, one can avoid the question of what cycle to use: For debugging purposes it is best to start with comparing the theoretical two-level asymptotic convergence factor ($\overline{\lambda}$ - see §4.1) with the experimental one by an algorithm which **simulates a two-level algorithm**. This is done by returning from the next coarser grid H back to the finest grid h only when the H equations have been solved to a very good accuracy (e.g., by taking large γ or very small δ in the cycles of §6.2). In this way we still separate away questions particular to too-coarse grids or related to three or more levels (delaying them to §6).

Another major simplification is to experiment first with particularly **convenient known solutions**. Even for complicated nonlinear systems, one can engineer the tested problem so that it has a *known* solution u. This is done simply by planting suitable right-hand sides, both for the interior differential equations and for the boundary conditions. (Even if the original problem is homogeneous, the program should anyway be written for *general* right-hand sides, because non-vanishing right-hand sides on *coarser* grids are obtained from finer grids residuals.)

For many nonlinear problems it is especially useful to experiment with solutions of the form $u = u_0 + \eta u_1$, where u_0 is a constant (or a

constant vector, if u is a vector of functions), and $\eta \ll 1$ is employed in the first experiments. Taking u_0 as the first approximation for a cycling algorithm, the behavior of the solution process should almost identically (identically in the limit $\eta \to 0$) be the same as for a linear problem with constant coefficients. For such problems precise comparison can be made with mode analysis. The comparison can be pushed to be even more precise by choosing u_1 to be a particular Fourier component.

Afterwards, η can gradually be increased to study the effect of nonlinearity. This effect is in this way easily and precisely separated away from other effects, including effects which are often confused with nonlinearity because they appear in nonlinear terms; e.g., convection. whose relative magnitude in fluid problems depend on the solution itself. (See §8 for the algorithm used in nonlinear cases.)

Debugging of multigrid programs can generally benefit from relations between the levels. Most bugs and conceptual errors immediately show as irregular behavior in the standard multigrid output (listing the history of the dynamic residual norms for every relaxation sweep on every level, as in §1.5). A preliminary error-detection table, based on that output, is provided in [B+78, Lecture 18]. *Troubles related to the treatment of boundaries often show in the following way*: The first couple of cycles exhibit the expected (interior) convergence factor, since the relative weight of errors near the boundaries is small. Later, however, the errors near the boundaries start to dominate and the convergence factor degrades. The coarser is the basic (finest) grid, the sooner this degradation appears.

5.2 Interpolation near boundaries and singularities

The coarse-to-fine interpolation of corrections $I_H^h v^H$ should use the boundary conditions on v^H even when they are not explicitly shown on the grid (sometimes they are only implicit in the program). Otherwise extrapolation formulae would be needed for I_H^h, giving slower asymptotic convergence factors [Oph78]. Exception is the case of discontinuity on the boundary, such as a boundary layer thinner than the meshsize, in which case boundary data should *not* be used (see §4.6).

Near boundary singularities, such as reentrant corners, the interpolation can be improved by using the asymptotic behavior, whenever known. That is, if the correction v^h to be interpolated from the coarser grid is expected to be of the form $v^h = w^h \psi$, where ψ is a known singular function and w^h is smooth, then polynomial interpolation should be used to interpolate w^h, not v^h. But such improvements are hardly needed (see §5.7).

5.3 Relaxation on and near boundaries

Except for some simple Dirichlet problems, discrete boundary conditions should generally be relaxed and transferred to the coarser grid in the same way interior difference equations do. It is important to notice that the boundary relaxation may spoil very much the' smoothness of interior residuals near the boundary. Indeed, for a smooth error function, the interior residuals formed near the boundary by relaxing the boundary conditions are $O(h^{l-m})$ times the typical magnitude of other interior residuals, where m is the order of the interior differential equation and l is the order of the boundary condition (usually $l < m$).

One way around this difficulty is immediately realized by looking at the one-dimensional case. It is clear in that case that boundary conditions need not be relaxed at all. Their errors are not functions that can be smoothed out in any way; they are just isolated values, which can always very well be represented on the coarser grid.

Analogously in higher dimensional cases, the role of relaxation should not be to impose the boundary conditions, but only to smooth their error *along* the boundary. Instead of Gauss-Seidel-type relaxation for the boundary condition $Bu = g$, say, one can make a Gauss-Seidel relaxation of the $\Delta_s Bu = \Delta_s g$, where Δ_s is an approximation to the Laplace operator along the boundary; e.g., in two-dimensional problems, $\Delta_s = \partial/\partial s^2$, where s is the boundary arclength.

In practice this means that, instead of satisfying the given condition at each boundary point, we only change its error to be equal to an average of the errors at neighboring boundary points. This increases the above l by 2, making the perturbation to the interior smoothness negligible. In case the boundary smoothing factor is not as good as the interior one, a couple of boundary sweeps may be performed per each interior one.

Another way around the above difficulty is to ignore it and rely on more precise residual transfers (§5.4). This however is cumbersome, hence not used often. A general practical way to obtain sufficient smoothing on and near the boundary is to apply there a robust block relaxation scheme (box relaxation – §3.4) and to employ more sweeps near the boundary, according to the Local Relaxation Rule (§5.7).

5.4 Residual transfers near boundaries

Relaxation seldom leaves smooth residuals near the boundaries, where the normal sequence of relaxation steps breaks off (in lexicographic schemes, for example). This is especially the case when relaxation of boundary conditions is not done in the manner explained in §5.3. Thus, in many cases it is important that each individual fine grid residual is correctly represented on the coarse grid. This is what we called *full* residual weighting. The full weighting near boundaries, and also near interfaces, is consider-

ably more complicated than the interior full weighting (described in §4.4). This is because the influence of the residual on the solution depends on its distance from the boundary; e.g., in Dirichlet problems for m-order elliptic equations, the influence is proportional to the $(m/2)$-*th* power of the distance.

Thus the weight used in transferring a residual from a fine-grid point to a coarse-grid point depends on the distance of both points from the boundary. Near boundary corners the dependence is even more involved. Hence, near boundaries the interior full-weighting rule (4.5) is modified to the requirement that

$$\sum_{x^H} \left(I_h^H r^h\right)(x^H) W^H(x^H) G(x^H) = \sum_{x^h} r^h(x^h) W^h(r^h) G(x^h) \qquad (5.1)$$

is satisfied for any given $r^h(x)$, where $\sum f(x^H) W^H(x^H)$ and $\sum f(x^h) W^h(x^h)$ are discrete approximations, on grids H and h respectively, to the integral $\int f dx$ for any function f, and where $G(\xi)$ has the behavior of the Green function near the boundary. That is, for two neighboring ξ_1 and ξ_2, the ratio $G(\xi_1)/G(\xi_2)$ roughly gives the ratio between the solutions of $Lu(x) = \delta_{\xi_1}(x)$ and $Lu(x) = \delta_{\xi_2}(x)$, with homogeneous boundary conditions. Usually one can take $G(\xi) = d_\xi{}^\alpha$, where d_ξ is the distance of the point ξ from the boundary, and $\alpha = m - l - 1$, where l is the order of the highest normal derivative in the neighboring boundary condition.

Relation (5.1) need not of course be kept very precisely. Residual weighting I_h^H that deviate from it by 20% may still easily exhibit the same convergence rates. Another way of deriving residual weighting near boundaries is by variational rules, like (4.10) in essentially-symmetric cases. Still other ways exist. It may all seem complicated, but, as explained in §11, it is in principle no more complicated than discretizing the original differential equations near the boundaries.

5.5 Transfer of boundary residuals

Residuals are defined and are transferred (with some averaging) to the coarser grid H, not only with respect to the interior equations, but also with respect to the boundary conditions. Boundary residuals of grid h are averaged to form the right-hand side of the corresponding boundary conditions of grid H. In order to do it in the right scale, the *divided* form of the boundary conditions (the form analogous to the differential conditions, without multiplying through by a power of h) should be used to calculate residuals, average them and transfer. For this purpose a clear conceptual separation should be made between boundary conditions and neighboring interior equations. Incorporating the former into the latter is often convenient to do, but it may easily lead to wrong transfers. (To do it right, one should assume the given boundary condition is incorporated on the finest grid, while the corresponding homogeneous condition is incorporated

on all coarser grids. Even when correctly done, however, this is equivalent to *imposing* the boundary condition at relaxation, which, as explained in §5.3, will sometimes result in large neighboring residuals and hence slower convergence, unless more precise residual weighting is used.) In symmetric problems one can consistently use the variational relation (4.10) without ever distinguishing between interior equations and boundary conditions, provided good interpolation I_H^h is defined. For some classes of problems this interpolation may be based on the difference equations, interior and boundary alike [Den82a].

5.6 Treatment and use of global constraints

In addition to boundary conditions many problems also specify some global conditions, such as integral relations, etc. For example, the pressure $p(x)$ in incompressible Navier-Stokes equations is determined only up to an additive constant; for its unique determination one should add an integral condition like

$$\int p(\underline{x})d\underline{x} = 0 \tag{5.2}$$

(integrating over the entire flow field), or a pointwise condition such as

$$p(\underline{x}_0) = 0\,. \tag{5.3}$$

Both conditions are in fact "global" in the sense we like to consider here, even though (5.3) does not look so global: One should generally consider as global any single discrete condition which has a large global effect on the solution. Boundary conditions in one-dimensional problems are also of this type (cf. §5.3). The normalization condition $(u,u) = 1$ in eigenproblems is a nonlinear condition of this type. In that case, unlike (5.2) or (5.3), together with the global condition there also comes a global unknown, the eigenvalue. This is often the case. The one-dimensional boundary conditions are associated with unknown boundary values, which should be considered as global unknowns.

A very useful device is indeed to *add global constraints to a problem to make it better posed, freeing as many global parameters.* For example, a physical continuation parameter γ for solving a nonlinear problem (cf. §8.3.2) may be a bad parameter near some "limit points"; i.e., with fixed the problem is ill-posed (or nearly ill-posed, hence not approximable on coarse grids). By converting γ into an unknown and adding instead another global characterization of solutions (e.g., the arclength along the continuation path, as in [Kel77]), the problem is made well posed (and, in particular, the coarse grid can well approximate fine-grid errors). Or one can free some accuracy parameters (e.g., allow a small constant error to be added to all equations) and add some known global information (e.g., total mass, or energy, or vorticity, when solving a time-step problem as part of

a long-range evolution where such quantities must be strictly conserved). A particularly useful device is to free such accuracy parameters only on coarser grids, using global constraints only for coordinating solutions on those grids with the current fine-grid solution. In this way for example the above bad physical parameter γ can still be fixed on the finest grid. Still another example of an added constraint with an added parameter (α) appears in §5.7.

An important advantage of the multigrid processing is the easy and natural way with which such global conditions and global unknowns can be handled. To be sure, it is often done in a wrong way: For example, misguided by the practice in relaxation solvers, one would tend to treat (5.3) at the relaxation phase. Imposing such a pointwise global condition just by changing p at x_0 is really harmful to the multigrid solution, since it frustrates the error-smoothing processes near x_0.

Global conditions need not be treated at all on the fine grid. There can be no error-smoothing related to such single conditions. All one has to do is to transfer the residual of the condition to serve as the right-hand side for a similar condition on the next coarser grid. In case of a nonlinear condition, FAS should be used (§8.3). A condition like $(u^h, u^h) = b^h$, for example, will be transferred by FAS to the condition $(u^H, u^H) = b^H$, where

$$b^H = b^h + (I_h^H u^h, I_h^H u^h) - (u^h, u^h), \tag{5.4}$$

which is a special case of (8.5). The global nature of a condition like (5.3) becomes increasingly transparent as it is transferred to coarser grids by proper approximations.

The global condition must of course finally be enforced while solving the coarsest-grid problem (cf. §6.3). Likewise, global unknowns should usually be changed only on the coarsest grid (thereby matching the number of unknowns to the number of equations therein). Sometimes the global equation should be operated *on several* of the coarsest grids. For example, approximations to a condition like $(w, u) = b$, where w is a given weight function which changes signs in the domain, must perhaps be operated on a grid fine enough to resolve these sign changes (or at least crudely simulate them). Similarly, the condition $(u^H, u^H) = b^H$ should be operated on a grid fine enough to crudely resolve the sign changes in the solution u.

When a global condition *is* treated in relaxation (on a coarse but not the coarsest grid, for example) this should be done in a global way. For example, the condition should be satisfied (at the end of each sweep, say) by adding a constant (or a smooth function) to the entire solution, or by multiplying the entire solution by a constant (or a smooth function), so that the error-smoothing process is not frustrated.

There are sometimes conditions which are neither completely global nor quite local, but have some intermediate scale. For example, sometimes, when one global control would not do, several constraints are added, each controlling the solution on one subdomain. Any such condition should not

be treated in relaxation wherever the meshsize is small compared with the scale of the condition.

In some relaxation schemes, the global condition seems to be needed in the local relaxation. For example, in the BGS schemes (§3.4) one solves in small boxes little problems similar to the given boundary-value problem. For the solution in the box to be uniquely determined, a condition like the global condition is needed there. In solving discrete incompressible Navier-Stokes equations in a small local box, for example, a pressure condition similar to (5.2) or (5.3) is needed. The best then is to use in each box a "no. change" kind of condition. That is, to require, for example, that some discrete approximation to $\int p(x)dx$ (integration being over the small box) retains its value from before the relaxation step.

5.7 Structural singularities. Reentrant corners. Local relaxation

Structural singularities are singularities in the problem other than those caused simply by singular forcing terms (singular right-hand sides in either the differential equations or the boundary condition. These are discussed in §7.1.). Boundary reentrant corners and singularities in the (left-hand side) differential operators are structural singularities. Such singularities cause the asymptotic multigrid convergence factor to degrade (although it is still bounded away from 1 independently of the meshsize). The reason is that the error components slow to converge in relaxation, which are approximate solution to the homogeneous differential equations (cf. §4.6), have a singular behavior around the structural singularity, therefore they are not well approximated by a coarse-grid correction, unless this singular behavior is taken into account in formulating I_H^h, L^H and I_h^H (see §5.2 and §4.6).

It is easy to overcome this difficulty in several other, perhaps simpler, ways. First, the degradation of the asymptotic factors may not be a two-level feature at all, appearing only because errors of the above type accumulate by cascading through many levels. Indeed, the degradation almost disappeared (in cases of reentrant corners studied in [BB87]) when W cycles replaced V cycles. Secondly, even with V cycles, the degraded asymptotic factors little affect the ability of the FMG algorithm to solve the problem to within truncation errors by just one cycle per level. This is explained by the fact that the same singularity causing the degraded convergence also causes large truncation errors, in exactly the same components which are slow to converge. Thus, exactly these components need not converge much in order to be approximated below truncation error. Numerical experiment with reentrant corners [Oph78], [BB87] show this to be true, unless the number of levels is very large indeed; this usually occurs when local refinements are used.

Finally, a general way to deal with structural singularities, so that even the many-levels-V-cycle convergence factors are not degraded at all, is by *local relaxation sweeps*, i.e., special relaxation passes over a small number of points near the singularity. Experiments with reentrant corners [BB87] show that just one such pass over a small number (typically less than 1%) of points before each full sweep is enough to completely restore the interior (i.e., regular) convergence factors. In fact, there is no need to explicitly identify singularities if the following more general rule is adopted.

Local Relaxation Rule: *add local relaxation steps wherever, and as long as, exceptionally large normalized residuals are detected at a small number of points.* The normalized residual is the error in satisfying a discrete equation, divided by the l_1 norm of the equation coefficients. (This general rule is justified by the theory of §1.1, and by the unified exchange-rate approach for controlling multigrid processes discussed in §9.6.)

Reducing Truncation Errors. Another question of course is how to obtain small truncation errors (i.e., a good approximation to the differential solution) near structural (and other) singularities. Two general ways are local refinements and subtraction of the singularity. The former is a very general approach, discussed in §9; the latter can be used when the singular behavior of the solution u is known and simple. For example, near a reentrant corner $u = \alpha\psi + w$, where ψ is a known singular function (e.g., $\psi = r^\gamma \sin(\gamma\theta)$ in case of Poisson equation and a corner of π/γ radians), α is an unknown constant, and w is an unknown *non*-singular function. The procedure then is to rewrite the problem in terms of w. In the new problem, α serves as a global unknown, and a new constraint should be added to express the requirement that w is smooth at the singularity. (In the this example, an inward derivative near the reentrant corner may be required to vanish.) This constraint is "global", since it controls the size of α, so its treatment, and the determination of α, should follow the rules in §5.6. When the singularity is subtracted off like this, the degraded convergence factor discussed above should disappear.

Chapter 6

Many-Level Cycles

Having obtained satisfactorily performing two-level cycling algorithms, one needs next to turn on the complete sequence of grids, using now the two-level techniques in recursion. The new algorithmic questions which arise are discussed below. Some of them could theoretically be investigated by three-level mode analysis, but this trouble is neither needed nor normally taken.

6.1 Multigrid cycles. Initial and terminal relaxation

For any grid h, finest or intermediate, a multigrid h-**cycle** can recursively be defined as follows: make ν_1 relaxation sweeps on grid h; transfer the residual problem to the next coarser grid $H(=2h)$ and solve it there approximately, using γ H-cycles (unless H is the coarsest grid); interpolate the grid-H solution and add it as a correction to the grid-h solution; finally, make ν_2 more sweeps on grid h. On the coarsest grid the problem is solved either directly or by ν_0 relaxation sweeps (cf. §6.3).

In two-level cycles only the sum $\nu = \nu_1 + \nu_2$ matters. When h is an intermediate grid the separate values of ν_1 and ν_2 do make some difference, although not a big one. In regular elliptic solvers experience shows that $\nu_2 = [\nu/2]$ is probably the best prescription (see for example [ST82, Tables 3.3]). In double-discretization schemes (§10.2) it is important to use $\nu_2 = 0$. In "accommodative" algorithms (see §4.1, §6.2), the values of ν_1 and ν_2 vary and they are determined internally.

Note also that the several passes of a complex relaxation sweep (such as ADZ) can be divided between the initial and the terminal stages of the cycle [ST82, §8.2].

6.2 Switching criteria. Types of cycles

The criteria when to switch from a fine grid h to the next coarser grid $H = 2h$ were examined in a previous stage (§4.1). These same criteria can be used recursively, i.e., not only when h is the finest grid. We need in addition some criteria for switching from any grid H back to the next finer grid h. Two kinds of switches are used: Fixed and accommodative.

Fixed algorithms switch from H back to h after a preassigned number γ of H-cycles. The h-cycle is recursively defined as type $\mathrm{C}(\nu_1,\nu_2)^\gamma$ if all these H-cycles are of this same type; γ is then called the *cycle index*. The cycle is defined as type $\mathrm{F}(\nu_1,\nu_2)$ if $\gamma = 2$ and the first H-cycle is itself an $\mathrm{F}(\nu_1,\nu_2)$ cycle, while the second H-cycle is a $\mathrm{C}(\nu_1,\nu_2)^1$ cycle. See flowcharts and operation counts in [Bra81a, §6.1]. The cycle $\mathrm{C}(\nu_1,\nu_2)^1$ is also called a V cycle and denoted $\mathrm{V}(\nu_1,\nu_2)$; see Fig. 1.1. The cycle $\mathrm{C}(\nu_1,\nu_2)^2$ is also called a W cycle and denoted $\mathrm{W}(\nu_1,\nu_2)$.

V cycles require considerably less work than W cycles (1/3 in two-dimensional problems with full coarsening). F cycles are somewhat less expensive than W cycles in one-dimensional problems with many levels, and in higher dimensions when semi coarsening is used; but otherwise perform practically the same as W cycles.

Fractional γ. For extra flexibility, a $\mathrm{C}(\nu_1,\nu_2)^\gamma$ cycle can even be defined for non-integer γ. At each level h of the cycle, the number of visits N_c^h from the next-finer level to h is required to be *on average* as close as possible to γN_c^H. The larger γ, the more accurately solved is the H-grid problem, relative to the accuracy of the h-cycle it serves.

When should $\gamma > 1$ *be used?* Except for simulating two-level algorithms (§5.1), a large cycle index should only be used if the *smoothest component's two-level convergence factor* χ *is large.* Normally $\chi \to 0$ as the finest level meshsize $h \to 0$, and the V cycle convergence will be dominated by high-frequency components, leading to a convergence factor close to the two-level factor $\overline{\chi}$. On the other hand, when χ is constant in h a V cycle will not attain $\overline{\chi}$, because the coarse grid equations themselves will only be crudely solved, and the error will accumulate through the levels. In this case, $\gamma > 1$ is mandatory for obtaining good asymptotic factors. Indeed, the asymptotic convergence factor χ_m of an m-level $\mathrm{C}(\nu_1,\nu_2)^\gamma$ cycle with fixed γ is

$$\chi_m = 1 - (1-\chi)\left(1 - \chi_{m-1}^\gamma\right), \quad \chi_1 = 0. \tag{6.1}$$

For $\gamma(1-\chi) < 1$, $\chi_m \approx 1 - (\gamma(1-\chi))^{m-1}$ tends to 1 for $m \to \infty$, but has a finite limit $\overline{\chi}$ for $\gamma(1-\chi) \geq 1$ that satisfies

$$\frac{1-\overline{\chi}}{1-\overline{\chi}^\gamma} = 1 - \chi. \tag{6.2}$$

For a d-dimensional problem and full coarsening, the multilevel cycle work is linearly proportional to the finest grid size for $\gamma < 2^d$, therefore the smoothest component must be approximated to at least $\chi > 1 - 2^{-d}$. The

optimal γ can be determined from (6.2) by maximizing the accuracy per unit work (cf. §9.5).

Examples of large χ are (a) severe singularities (cf. §5.7); (b) intergrid transfers whose orders are too low (cf. §4.3). If for instance first-order transfers and Galerkin coarsening (§4.5) are employed for the d-dimensional Poisson equation, then $\chi = .5$. The optimal parameters are then $\gamma = 2.7, \overline{\chi} = .67$ for $d = 2$, $\gamma = 3.5, \overline{\chi} = .57$ for $d = 3$, and $\gamma \approx d, \overline{\chi} \approx .5$ for $d \gg 1$. (c) non-elliptic and singular perturbation problems. The artificial viscosity on grid kh is k times larger than on grid h, hence visiting grid kh only once per cycle would yield an asymptotic convergence factor no better than $1 - 1/k$ [Bra81a, §5.1]. Since $k = O(h^{-1})$ on the coarsest grid, the V cycle asymptotic factor will be $1 - O(h)$, which is very poor indeed. h-independent asymptotic convergence may be restored by using $\gamma > 1$; V cycles may however perform at a satisfactory level within FMG algorithms (see §7.4 and the numerical experiments in [Bra81a, §7.1]).

Accommodative algorithms switch from grid H back to grid h when a certain norm of the residuals on grid H drops below some factor η times the latest value of the corresponding norm on grid h. The parameter η is not sensitive one; a good general prescription seems to be $\eta = 1.1\overline{\lambda}$. If $\overline{\lambda}$ is not approximately known, use $\eta = 2^{-d}$, a value related to accuracy-to-work exchange-rate considerations (cf. §9.6).

Generally, accommodative algorithms may be troublesome at program development stages, since they may cause more complex interactions between the internal checks and the real questions one likes to examine. Their flexibility may prevent us from seeing some of the troubles, and they are not suitable for precise comparisons. In the production stages, accommodative algorithms have the disadvantage that they require the extra work of calculating the residual norms. On the other hand, accommodative algorithms can be more robust. Also, in complicated problems (which is where this robustness is needed), the residual norm calculation is inexpensive relatively to other calculations, assuming *dynamic* residuals (those calculated anyway in the relaxation process) are used.

6.3 Coarsest grids. Inhomogeneous and indefinite operators

When the multigrid h-cycle performs considerably poorer than expected, it is first important to distinguish between fine-grid and coarse-grid troubles. This distinction is easy to make, by simulating two-level algorithms (taking large γ or small η) and examining whether this improves the convergence factor (per h-cycle), and how much this improvement depends on the size of h. Also examine whether reasonable convergence is obtained on your coarsest grid. If not, or if the trouble is confined to coarse h, the following remarks may be relevant.

Inhomogeneous operators are the main source for the special troubles appearing only on sufficiently coarse grids. On such grids, lower order terms of the operator start to affect, or even dominate, the smoothing and convergence factors. If we have neglected them in designing the fine-grid relaxation, we should now take them into account. Generally, on every grid h, the important terms for designing relaxation are the h-principal terms (see §§2.1, 3.1, 3.4).

Another type of coarse level trouble is exhibited for example by the equation $-\Delta u + \sigma u = f$ with purely Neumann boundary conditions. If σ is positive but very small, the smoothing factor of a GS relaxation is essentially the same as for Poisson equation, but the convergence factor is roughly $4/(4 + h^2\sigma)$, which may be very slow even on the coarsest grid. Hence the coarsest-grid equations should be solved either directly (e.g., by elimination, which is inexpensive since the coarsest grid can contain just few points), or by relaxation, where after each sweep a suitable constant is subtracted from the approximate solution [ABDP81, §4]. If $\sigma = 0$ everywhere except in some small subdomain, that constant subtraction should be employed on all grids which are not fine enough to resolve that small subdomain.

Indefinite case. If σ is negative, the situation is much worse, whatever the boundary conditions: For the coarse grid to approximate the slowly converging fine-grid component, its meshsize must be fine enough: For large $|\sigma|$, the coarsest meshsize must satisfy $H \leq O(R^{-1/p}(-\sigma)^{-.5(p+1)/p})$, where R is the radius of the domain and p is the approximation order. In many cases this H is smaller than the *finest* affordable meshsize. To use coarser H, different intergrid transfers should be employed (cf. §4.2.2 and [BL97]).

In designing fine-level relaxation schemes for complex systems of equations, e.g., in fluid dynamics, we can take only subprincipal terms into account (§2.1, 3.1). On very coarse grids, however, this is no longer fully justified, and if the same relaxation schemes are still used, there the smoothing factors may deteriorate. We may then have to use either more sweeps (by increasing ν and/or γ, or by using accommodative algorithms), or more sophisticated relaxation. In solving Navier-Stokes equations, for example, improved results were obtained by using the high-speed DGS scheme (see §19.3) on all finer grids, while employing the most robust BGS (see §3.4) on the two coarsest grids.

Even for homogeneous operators, convergence of h-cycles can sometimes be slower on very coarse grids, because the convergence factor $\overline{\lambda}$ cannot be smaller than $O(h^{\tilde{m}})$; see (C) in §4.3. In such cases one can make more h-cycles, by increasing γ or switching accommodatively, which is inexpensive since h is coarse.

Sometimes troubles seen on coarse grids are only indications of bad procedures at some special, restricted regions, such as boundaries (see §5.1), or they may signal the need to operate some global conditions, which are not enforced on finer grids (see §5.6).

Of special concern is the **coarsest grid** itself. Relaxation there should be *converging*, not just smoothing as on other grids. various conditions not enforced on finer grids must be enforced on the coarsest one, calling for special procedures. If nothing better is known, one can always use either a direct solver or a slow but safe iterative process such as Kaczmarz relaxation (cf. §1.1); on the coarsest grid they are affordable. Finally note that the coarsest grid cannot efficiently contribute to convergence if all its points happen to lie too close (much closer than a meshsize) to boundaries with Dirichlet boundary conditions.

Chapter 7

Full Multi-Grid (FMG) Algorithms

The cycling algorithms developed in the previous stages are easily converted into full multigrid (FMG) programs. The main difference is that instead of starting with an arbitrary approximation (e.g., $u_0^h = 0$) on the finest grid, the first approximation u_0^h is obtained by an interpolation $\mathbb{I}_H^h$ from a coarse-grid (approximate) solution u^H. Namely, $u_0^h = \mathbb{I}_H^h u^H$, where $H = 2h$ and where u^H has been calculated by a similar FMG process with H as its finest level. The full algorithm can be either "fixed" (as for example in Fig. 1.2 above), or "accommodative" (as in [Bra77b, §1.3], [Bra80b, Fig. 1], [Bra79a, §3.6 and Fig. 1], [BD79, §2.2]). Both versions are available in the model program FMG1 [MUG84].

FMG algorithms are in a sense *easier* to program than cycling algorithms. Their main deriving routine is some lines longer, they include an additional interpolation routine ($\mathbb{I}_H^h$), and they involve several more algorithmic questions (dealt with in the following subsections) - but on the other hand they are much more *forgiving*. Their basic performance, which is to solve all problems to the level of truncation errors in just one or two cycles (see §7.3), is undisturbed by various little mistakes (conceptual mistakes or even programming bugs, especially in treating boundaries) which may degrade very much the *asymptotic* convergence of cycling algorithms. These mistakes may still be important in other situations, hence it is safer to detect them by perfecting the multigrid cycling (as in §4, 5 and 6) before turning FMG on. But it is important to understand those cases in which, for good reasons, the FMG results are absolutely satisfactory despite the necessarily bad asymptotic convergence factors. Examples are numerous; some are emphasized in §3.3, 5.7, 7.4, 7.5, 10.2 and 18.6. In many of these cases the analyses of FMG described in §7.4 and 7.5 can be useful.

More important than these apriori analyses, though, one should *always calculate in the FMG solver the rate of convergence to the differential*

solution, which is the real purpose of the solver; see the end of §7.2 and §12.

It is also worth mentioning at this point that the FMG algorithm can incorporate into it continuation processes (see §8.3.2), grid adaptation processes (see §9.6), and, generally speaking, any process aimed at solving original "outer" problems (see §13).

7.1 Order of the FMG interpolation

The FMG full-solution interpolation operator $\mathbb{I}_H^h$ is not necessarily the same as the correction interpolation operator I_H^h used in the multigrid correction cycles. Often the order of $\mathbb{I}_H^h$ should be higher than the order of I_H^h since the first approximation is smoother than the corrections: In the right-hand side of the latter (i.e., in the residuals) the amplitude of high-frequency components is usually comparable to that of low-frequency components. The optimal order of $\mathbb{I}_H^h$ depends on the purpose of calculations. If one desires ultimately to get the algebraic error (i.e., the errors in solving the *difference* equations) to be very small (far below truncation errors), then $\mathbb{I}_H^h$ should exploit all the smoothness of u^h in order not to produce unnecessary high-frequency algebraic errors. (High frequency errors are the most expensive to liquidate in the multigrid cycling, since they are processed on the finest grid.) In fact, in such a case the first few cycles should also employ a correction interpolation I_H^h of suitably high orders. The precise rules for scalar elliptic equations are given in [Bra81a, App. A.2]. Note that these rules assume that the order of smoothness is known in advance.

Usually, however, the smoothness order is not apriori known. More importantly, we are not interested in solving to arbitrarily small algebraic errors; we like them only to be smaller than the truncation errors. The optimal order depends then on the norm by which we measure errors. Suppose we solve a $q \times q$ system of differential equations, and assume our error norm includes difference-quotients up to order l_j in the j-*th* unknown function, $1 \le j \le q$. Then the order $\hat{m}^j$ of the first interpolation of that function should not be less than $p + l_j$, where p is the approximation order. Otherwise, the $O(h^{\hat{m}^j - l_j})$ high-frequency errors produced by interpolation would be much larger than the $O(h^p)$ (low-frequency) truncation errors.

In the case of equations with strongly discontinuous coefficients, the higher order interpolation $\mathbb{I}_H^h$ should be of a different form, taking into account the different sense of smoothness in the solutions (cf. §4.6. A higher-order interpolation of this sort is presented in [ABDP81, Eq. (5.12)]). The remarks of §5.2 apply here as well.

A general approach for equations with discontinuous coefficients or right-hand side is to use one-sided interpolation stencils that avoid straddling the discontinuity.

Whatever FMG interpolation $\mathbb{I}_H^h$ is used near a right-hand side discontinuity, it should be followed by special local relaxation steps around

the discontinuity. (This automatically follows from the general Local Relaxation Rule of §5.7).

In some programs, especially general-domain programs, the higher order interpolation I_H^h turned out to cost more CPU time than the rest of the algorithm [Oph78]. An interpolation of an order smaller than indicated above may then be more practical. In case of rotatable differential operators, simpler higher-order interpolations can be used, based on the equations themselves [Hym77], [FW81, §3].

7.2 Optimal switching to a new grid

In designing the FMG algorithm one should decide how well the equations on level $H = 2h$ should be solved before the solution is interpolated for the first time to grid h and the h-cycles start. The optimal point to switch is when the work of h-cycles becomes as efficient as the work of H-cycles in reducing the *differential* error (the difference between the differential solution u and our current computed solutions, $\tilde{u}^H$ or $u_0^h = I_H^h \tilde{u}^H$). This happens when the *algebraic* error on grid H, namely $e_*^H = \|u^H - \tilde{u}^H\|$, is about 2^{-d} times the algebraic error on grid h, $e_0^h = \|u^h - u_0^h\|$, where d is the dimension, u^H is the exact solution of the H-equations and u^h is the exact solution of the h-equations. This is because h-cycles are about 2^d times as expensive as H-cycles. The switching point $e_*^H \approx 2^{-d} e_0^h$ is roughly equivalent to

$$e_*^H \approx \beta E^H, \qquad \beta := \frac{1 - 2^{-p}}{2^d - 1}, \tag{7.1}$$

where $E^H = \|u^H - u\|$ is the truncation error on grid H and p is the order of approximation. This is because $e_*^H + E^H \approx e_0^h + 2^{-p} E^H$, both sides being estimates for $\|\tilde{u}^H - u\|$. In practice the values of e_*^H and E^H are of course not known, but we can derive from (7.1) the algebraic reduction needed on level H before switching. Namely, denoting by e_0^H the value of e^H when the H cycles are started and by e_*^H its value at the switching point (7.1), and assuming that the switching from the $2H$-cycles to the H-cycles has been made when a relation similar to (7.1) was reached on level $2H$, we find that $e_0^H \approx 2^d e_*^{2H} \approx 2^d \beta E^{2H}$ while $e_*^H = \beta E^H = \beta 2^{-p} E^{2H}$, consequently the algebraic reduction on grid H is roughly

$$\frac{e_*^H}{e_0^H} \approx 2^{-p-d}. \tag{7.2}$$

This can be obtained by about

$$\frac{p+d}{\log_2(1/\overline{\lambda})} \tag{7.3}$$

H-cycles, where $\overline{\lambda}$ is the convergence factor per cycle (§4.1), which can of course be measured. This number of required H-cycles usually turns out

to be 1 or 2. A more precise strategy for simultaneously optimizing the number of cycles at all levels is described in [TTKR10].

7.3 Total computational work. Termination criteria

Suppose that on the finest grid h we wish to obtain an algebraic error smaller than a specified factor α times the truncation error: $e^h \leq \alpha E^h$. Suppose also that the switch from level $H = 2h$ is made roughly when (7.1) is met; i.e., when $2^{-d}e_0^h \approx e_*^H \approx 2^p \beta E^h$. Then the algebraic error reduction required on grid h is roughly $\alpha_1 = e^h/e_0^h \approx \alpha(1-2^{-d})/(2^p-1)$. The number of work units to obtain such a reduction is about $\log(1/\alpha_1)/\log(1/\mathring{\mu})$, where $\mathring{\mu}$ is the interior convergence factor per work unit (see §4.1) and is usually just modestly larger than the interior smoothing factor $\overline{\mu}$. Counting also the work for the reduction (7.2) on coarser grids, we find that the total number of work units theoretically required by the Full MultiGrid (FMG) algorithm is about

$$\left\{\log\left(\frac{2^p-1}{\alpha(1-2^{-d})}\right) + \frac{p+d}{2^d-1}\log 2\right\} / \log\left(\frac{1}{\mathring{\mu}}\right). \tag{7.4}$$

The actual total number of work units is usually slightly larger than (7.4), because of the need to perform *integral* numbers of relaxation sweeps and coarse grid corrections. *In practice, one V or W cycle with $\nu = 2$ or 3 at each FMG level, yields an e^h that is significantly smaller E^h.*

The observation that only one cycle on each level is needed in FMG algorithms, and is also basically enough to reduce the algebraic errors to the level of truncation errors (even though sometimes two shorter cycles, each including less relaxation work, may be more efficient), can heuristically be understood as follows. The first approximation on grid h, obtained by interpolating the grid-$2h$ solution, necessarily contains two types of errors:

(A) High-frequency errors, i.e., errors having oscillations invisible and hence unapproximable on the coarser grid.

(B) Aliasing errors, i.e., smooth errors introduced by high-frequency data because on the coarse grid high-frequency data is mistaken for smooth data.

Relaxation on grid h can be efficient in removing the high-frequency errors (because of their local nature: At each point they are essentially determined by the neighboring residuals). Having removed the high-frequency errors we have also removed the high-frequency data from the *residual* problem, hence we can then go back to grid 2h with the residual problem to remove the aliasing errors (which are smooth errors, hence not determined by neighboring residuals, hence inefficiently treated by relaxation).

The algorithm may indeed be terminated after a fixed number of cycles on the finest grid h. This number is roughly $\log(1/\alpha_1)/\log(1/\bar{\lambda})$, and in practice it is one or two. Or else, especially if an estimate for $\bar{\lambda}$ is not known, termination can be done when a certain norm of the residuals on grid h becomes smaller than a corresponding norm of $\alpha\tau^h \approx \alpha(2^p-1)^{-1}\tau_h^{2h}$ (see §8.4).

One should of course check numerically, using a problem with a known solution or a solution computed on a finer grid, that with these termination procedures $e^h \leq \alpha E^h$ is indeed obtained. Better still (from the more advanced point of view of §13), one can observe the behavior of e^h as function of h. This can approximately be done in (the real, production) runs where the solution is *not* known (see §1.6 and the further discussion in §13). Quite often these checks also reveal mistakes in the *discretization* schemes, not just in the multigrid solver.

7.4 Two-level FMG mode analysis

Instead of developing full multigrid (FMG) programs from the cycling programs, including boundary conditions, one can first develop the FMG algorithm still within the framework of two-level mode analysis (immediately following the stage of §4). This may again serve to separate away questions related to boundary conditions (questions discussed in §5), and questions related to many levels and to very coarse grids (§6) from the particular questions of the FMG algorithm (§7.1–7.3). The latter can then be examined in the interior, without boundary interference (or also with ideal boundaries – see §7.5), and the performance figures so calculated can serve as ideals against which the actual program can be developed and debugged. Such an analysis is particularly useful in cases the usual two-level analysis (that of §4.1 above) is too pessimistic because of the existence of different components with markedly different convergence properties. For example, in case of nearly non-elliptic or nearly semi-elliptic problems there are smooth characteristic components which converge slower than others, since for such components L^H is not a very good approximation to L^h. But exactly for the same components and for the same reason, L^h itself is not a good approximation to L, hence these components do not need much algebraic convergence, and the fact that they have slow asymptotic rates does not matter [Bra81a, §5.1, 5.2]. What we need then is an analysis which does not tell us the worst *asymptotic* rate, but tells us, separately in each mode, how well we solve the problem by an FMG algorithm with a given number of prescribed cycles.

To analyze the FMG solution of $Lu = f$, where L is a $q \times q$ system and has constant coefficients (or frozen local values, in case the original L was variable), we first analyze a single component $u(\underline{\theta}) = exp(i\underline{\theta} \cdot \underline{x}/\underline{h})$. We calculate the corresponding f, and hence also the solution $u^H = u^H(\underline{\theta})$ to the coarse-grid equation $L^H u^H = f^H = I^H f$, where I^H is the local

averaging used in our discretization for transferring a continuum function to grid H. The interpolation of u to the fine grid gives an approximation $u_0^h = I_H^h u^H$ made up of 2^d Fourier components (the harmonics of $\underline{\theta}$, i.e. all the components $\underline{\theta}'$ such that $\underline{\theta}' = \underline{\theta} + (\nu_1, \ldots, \nu_d)\pi$, ν_j integer and $-\pi < \theta_j' \leq \pi$). To the set of 2^d amplitudes we then apply the usual (cycling) two-level mode analysis (§4.1); i.e., using (4.1) we calculate the $2^d q \times 2^d q$ matrix $M(\underline{\theta})$ describing the transformation of these 2^d amplitudes by one cycle. The result of applying k such cycles on grid h we denote by $u_k^h(\underline{\theta}) = M(\underline{\theta})^k u_0^h(\underline{\theta})$. Having calculated $u_k^h(\underline{\theta})$ we can then examine its qualities by several measures.

One measure is **how well below truncation errors** u_k^h is. This is measured for example by

$$\max_{|\underline{\theta}| \leq \pi} \frac{\|u_k^h(\underline{\theta}) - u(\underline{\theta})\|}{\|u(\underline{\theta}) - u^h(\underline{\theta})\|}, \tag{7.5}$$

where $u^h(\underline{\theta})$ is the exact solution of the grid h equations, and $\|\cdot\|$ is any norm under which we want to guarantee convergence. Note that u_k^h is made of 2^d components, while u^h and u are made of only one of those; the norms can be taken anyway.

Another, perhaps more direct and important measure, is **how well we have solved the differential equations**. That is, we directly measure $\|u_k^h - u\|$ thus evaluating not only the performance of our fast solver, but also the quality of our discretization scheme itself: We evaluate the total quality of our procedures in solving the differential equations at a given amount of work. In measuring the error $\|u_k^h - u\|$ we should of course give smaller weights to high-frequency $\underline{\theta}$'s; we cannot and need not solve for them as accurately as for low frequencies. Thus, if we aim at an approximation order p, good measures of performance are

$$\max_{|\underline{\theta}| \leq \pi} \frac{\|u_k^h(\underline{\theta}) - u(\underline{\theta})\|}{|\underline{\theta}|^p}, \tag{7.6}$$

or

$$\left\{ \int_{|\underline{\theta}| \leq \pi} |\underline{\theta}|^{-2p} \|u_k^h(\underline{\theta}) - u(\underline{\theta})\|^2 d\underline{\theta} \right\}^{\frac{1}{2}}, \tag{7.7}$$

etc. Several such measures can easily be produced, approximately, by the program that calculates $u_k^h(\underline{\theta})$. The program is an easy extension of the usual (cycling) two-level mode-analysis program.

All the issues examined by the two-level cycling analysis (relaxation, the number $\nu = \nu_1 + \nu_2$ of sweeps per cycle, and the interior operators I_h^H, L^H and I_H^h - see §4) can further (more accurately) be optimized by the FMG mode analysis. In addition we can examine by this analysis the effect (in the interior) of various I_H^h interpolation procedures, various values of ν_1, ν_2 and k, and, most importantly, various interior discretization procedures.

An important advantage is that this analysis can be used even in cases where no *algebraic* convergence is desired or obtained (cf. §10.2). Moreover, *the predictions of the FMG-mode analysis are more robustly held than those of the cycling mode analysis when real boundaries are added.* For example, take a problem with singularities in the boundary, such as reentrant corners. The effect of such singularities (similar to the effect of singular perturbations mentioned above) is to make the asymptotic convergence factors per cycle worse than predicted by the interior cycling mode analysis. But this occurs for particular components with large truncation errors (see §5.7), so that predictions like (7.5) are likely to be roughly held up.

A very simple example of two-level FMG mode analysis for a singular perturbation equation is given in [Bra81a, §5.2]. A similar analysis holds for semi-elliptic cases.

7.5 Half-space FMG mode analysis. First differential approximations

Another advantage of the two-level FMG mode analysis is the possibility to make it also near a piece of boundary, modeled by a grid hyperplane, so that the entire domain is modeled by a half space. This is particularly important in non-elliptic or singular perturbation problems, where the high-frequencies far in the interior can still strongly be affected by the boundary.

A simple example is given in [Bra81a, §5.3] for a singular perturbation equation. Its upshot is that both the algebraic error (after a one-cycle two-level FMG algorithm) and the truncation error increase as functions of the distance from the boundary, both eventually becoming as large as the solution itself; but at points where the truncation error is still small (compared with the solution), the algebraic error is smaller yet: the latter is at most a quarter the size of the former. (Again, a similar analysis can be made, with very similar results, for semi-elliptic equations.) *Interior* analysis could not of course describe this situation, and its relevance in such cases is therefore questionable.

Those examples in [Bra81a] illustrate another technique which can be used whenever one wants to focus one's analysis on *smooth* components. One can then simplify the analysis very much by using the **first-differential approximation** (the first terms in a Taylor expansion) of the difference operators, instead of the difference operators themselves. For example, the first differential approximation to the 5-point Laplacian is the differential operator $\partial_{xx} + \partial_{yy} + \frac{1}{12}h^2(\partial_{xxxx} + \partial_{yyyy})$. The analysis proceeds in the continuous domain, without mentioning grids except through the quantity h appearing in the operators.

Part II

Advanced Techniques and Insights

Having completed the above stages of developing the basic multigrid solver, one can start introducing various important improvements. Some possibilities are outlined in the following sections, followed by comments of more "philosophical" nature, which can however be readily useful to the practitioner. We then close with general remarks on multigrid applications to chains of problems and to evolution problems.

Chapter 8

Full Approximation Scheme (FAS) and Applications

The Full Approximation Scheme (or Full Approximation Storage - FAS) is a widely used version of multigrid inter-grid transfers, explained for example in [Bra76, §4.3.1],[Bra77a, §5],[Bra77b, §1.2], [Bra79b, §2.1],[Bra81b, §2.3]. It has mainly been used in solving nonlinear problems, but it has so many other applications that it should perhaps be used in most advanced programs. The scheme, its programming, and several of its applications are sketched below. Another, perhaps the most important application of FAS is described in §9, and yet another one in §15.

8.1 From CS to FAS

Consider first a **linear** problem $L^h u^h = f^h$ on a certain grid h, with some approximate solution $\tilde{u}^h$ obtained, say, after several relaxation sweeps. Now we like to use coarser grids in order to supply fast h approximations to smooth errors. Thus, it is the corrections $v^h = u^h - \tilde{u}^h$ which we try to approximate on the next coarser grid $H = 2h$. In the simpler multigrid programs, such as in §1.5, the coarse-grid unknown function is indeed v^H, intended to approximate the correction v^h. This multigrid version is therefore called the **Correction Scheme (CS)**. Since $L^h v^h = r^h$, where

$$r^h = f^h - L^h \tilde{u}^h \tag{8.1}$$

is the fine-grid residual, the CS coarse-grid equations are

$$L^H v^H = I_h^H r^h, \tag{8.2}$$

where L^H approximates L^h on the coarse grid. Once (8.2) has been approximately solved, its approximate solution $\tilde{v}^H$ is interpolated to the fine grid and serves as a correction to the fine-grid solution:

$$\tilde{u}^h_{\text{NEW}} = \tilde{u}^h + I_H^h \tilde{v}^H. \tag{8.3}$$

In the **Full Approximation Scheme (FAS)** we perform exactly the same steps, but in terms of another coarse grid variable. Instead of v^H we use

$$\hat{u}^H = \hat{I}_h^H \tilde{u}^h + v^H \tag{8.4}$$

as the coarse-grid unknown function, where $\hat{I}_h^H$ is some fine-to-coarse transfer which need not be similar to I_h^H in (8.2). (They are in principle defined on different spaces.) This coarse-grid variable $\hat{u}^H$ approximates $\hat{I}_h^H u^h$, the full intended solution represented on the coarse grid, hence the name "Full Approximation Scheme". The FAS coarse-grid equations, derived from (8.2) and (8.4), are

$$L^H \hat{u}^H = \hat{f}^H \tag{8.5a}$$

where

$$\hat{f}^H = L^H(\hat{I}_h^H \tilde{u}^h) + I_h^H r^h. \tag{8.5b}$$

Having obtained an approximate solution $\tilde{u}^H$ to (8.5), the approximate coarse-grid correction is of course $\underline{v}^H = \tilde{u}^H - \hat{I}_h^H \tilde{u}^h$, hence FAS interpolation back to the fine grid, equivalent to (8.3), is

$$\tilde{u}^h_{\text{NEW}} = \tilde{u}^h + I_H^h(\tilde{u}^H - \hat{I}_h^H \tilde{u}^h). \tag{8.6}$$

To use directly

$$\tilde{u}^h_{\text{NEW}} = I_H^h \tilde{u}^H. \tag{8.7}$$

would be worse, of course, since it would introduce the interpolation errors of the full solution u^H instead of the interpolation errors of only the correction v^H (but see end of §8.5). Notice that $\hat{I}_h^H \tilde{u}^h$ in (8.6) and in (8.5b) must be identically the same: A common programming mistake is to have a slight difference between the two; this difference may dominate the calculated correction function and hinder convergence. Also it is important to start with identically the same $\hat{I}_h^H \tilde{u}^h$ as the first approximation in solving (8.5).

Note: The FAS equations (8.5) are a model for each interior differential equation *or boundary condition* appearing in the problem: each of them is transferred, separately, by this prescription. Other side conditions, global constraints, etc., are transferred in exactly the same way. See for example (5.4). In case a double discretization is employed at all levels (see §10.2), two functions $\hat{f}^H$ should be calculated, one for each coarse-grid operator L^H [Bra81a, §2.1].

For linear problems, the FAS steps (8.5)–(8.6) are fully equivalent to the CS steps (8.2)–(8.3). Indeed, the safest way to construct a correct FAS program is to start with a linear subcase, write first a CS program, then convert it to FAS and check that, at each iteration, the FAS results on the finest grid are identically the same as the CS results, except for round-off errors. Common mistakes in implementing FAS, especially in treating boundary conditions, are thus easily detected. The conversion of a CS

program to FAS can be done by a trivial addition of three routines [B+78, Lecture 12]. It can be done either for a cycling program or for an FMG program. The simplest examples are the cycling program FASCC and the program FMG1 [MUG84].

8.2 FAS: dual point of view

To see why FAS is preferable to CS in many, even linear situations, we rewrite (8.5), using (8.1), in the form

$$L^H \hat{u}^H = f^H + \tau_h^H, \tag{8.8}$$

where

$$\tau_h^H := L^H(\hat{I}_h^H \tilde{u}^h) - I_h^H(L^h \tilde{u}^h) \tag{8.9}$$

and $f^H = I_h^H f^h$. Observe that (8.8) without the τ_h^H term is the original coarse-grid equation (with the particular discretization $f^H = I_h^H f^h$), and that $\hat{u}^H$ approximates $\hat{I}_h^H \tilde{u}_{\text{NEW}}^h$, and at convergence $\hat{u}^H = \hat{I}_h^H u^h$. Hence τ_h^H is the **fine-to-coarse defect correction**, a correction to the coarse-grid equation designed to make its solution coincide with the fine-grid solution.

We can now reverse our point of view of the entire multigrid process: Instead of regarding the coarse grid as a device for accelerating convergence on the fine grid, we can view the fine grid as a device for calculating the correction τ_h^H to the coarse-grid equations. Since this correction depends on the non-smooth components of the solution, we obtain it by interpolating the solution to the fine grid and correcting its non-smooth components by relaxation. Having obtained the correction τ_h^H by such a *"visit"* to grid h, we continue the solution process on grid H. Later we may *"revisit"* grid h, in order to update τ_h^H. In such a case the interpolation (8.6) should better be used if we do not want to lose the non-smooth components of the solution already obtained by relaxation in previous visits. This entire process will then yield a solution on grid h, which we can improve by inserting into it visits to grid $h/2$. Etc.

Because $\hat{u}^H$ is just an improvement to u^H, we can omit the $\hat{}$ symbol, and keep in mind that the meaning of u^H changes as soon as an approximation $\tilde{u}^h$ exists on the next finer grid h.

This point of view, and the fact that the full fine grid solution is represented on all coarser grids, open up many algorithmic possibilities, as we shall see below.

8.3 Nonlinear problems

The Correction Scheme is not applicable to nonlinear problems, since the correction equation $L^h v^h = r^h$ is valid only for linear L^h. In case L^h is

nonlinear, the correction equation can instead be rewritten using (8.1) in the form

$$L^h(\tilde{u}^h + v^h) - L^h(\tilde{u}^h) = r^h. \tag{8.10}$$

Transferring this equation to the coarse grid (replacing L^h by L^H, $\tilde{u}^h$ by $\hat{I}_h^H \tilde{u}^h$, v^h by v^H and r^h by $I_h^H r^h$), we get the FAS equations (8.5). Thus, one important advantage of FAS over CS is its direct applicability to nonlinear problems. (This is a general property of defect correction schemes – see for example [Lin76], [Ste78].)

The CS scheme can of course be applied to the Newton linearization of L^h around the current approximation $\tilde{u}^h$. But FAS is usually preferable, because:

(A) No global linearization is needed. The only linearization is the local one used in relaxation, and even this is seldom needed (see §3.4). Hence, no extra storage for coefficients of the linearized equation is required.

(B) The programming is very convenient since the FAS equations (8.5) are exactly the same as the original discrete equations, except for a different right-hand side. Hence the same relaxation routine, the same residual transfer routine, and the same boundary relaxation routine, can be reused at all levels of the program.

(C) In the linearized problems, where sight is lost of the original problem, it is much more difficult to employ the most efficient multigrid approaches: One tends to solve far below truncation errors and therefore tailor unnecessarily complicated and less vectorizable "perfect smoothers" (cf. §3.3), getting into unnecessary troubles connected with small ellipticity (cf. §7.4–7.5) or boundary singularity (§5.7). More importantly, one cannot then integrate the FMG algorithm with various processes outside the solver, such as continuation (see §8.3.2), grid adaptation (§9.6), and others (§13).

(D) The FAS multigrid rate of convergence is not constrained by the convergence rate of Newton iterations. It is still mainly determined by the interior smoothing rate. Solving the nonlinear problem is no more expensive than solving, just once, the corresponding linearized problem. (In many cases, though, Newton convergence rate is fast enough to impose no real constraint to an FMG algorithm.)

(E) FAS is useful in various other ways. Particularly important for nonlinear problems are its applications in solving chains of problems (§15), near discontinuities (§8.5) and in automatic local refinement procedures (§9).

On the other hand, Newton linearization may still be preferred in those cases where it is an essential part of the discretization process, as in some

finite-element formulations. This, however, may make such formulations less attractive in multigrid environments.

Although not employed in the FAS multigrid *processing*, we still use Newton linearizations in the local mode *analysis*, to estimate smoothing and convergence factors (§3, 4). See also §5.1 for a debugging technique related to nonlinear problems.

Examples: Various nonlinear problems have been solved by FAS, including transonic flow problems [SB77], [Jam79], [MR82]; steady-state compressible and incompressible Navier-Stokes equations (§19 and 20); the Bratu equation$\Delta u + \lambda e^u = 0$ (see §8.3.2); and complementary problems arising from free boundary problems. A simple example of the latter is to calculate the nonnegative function u which minimizes the Dirichlet integral $\int(\underline{\nabla} u \cdot \underline{\nabla} u + 2fu)dx$. without the $u \geq 0$ constraint the problem is of course equivalent to a problem with the Poisson equation $-\Delta u = f$. But the nonnegativity constraint introduces nonlinearity. Using a FAS-FMG algorithm this nonlinear problem is solved with essentially the same amount of work as Poisson problems [BC83].

8.3.1 Eigenvalue problems

Eigenvalue problems can simply be regarded as nonlinear problems. They are nonlinear since the unknown eigenvalue λ_j multiplies the corresponding unknown eigenfunction u_j. Also, to fix the eigenfunctions, nonlinear orthonormality conditions $(u_i, u_j) = \delta_{ij}$ are added as global constraints. The solution algorithm proceeds as a usual FAS-FMG multi grid, with the global constraints treated basically as in §5.6. The eigenvalues are updated once per cycle, together with a more precise determination of the individual eigenfunctions within the space spanned by them, by a Rayleigh-Ritz process. Experiments for model problems [BMR83] show that an FMG algorithm with one V(2, 1) cycle on each level gives a discrete eigenfunction with algebraic errors much smaller than truncation errors. Similar work is needed for each additional eigenfunction.

8.3.2 Continuation (embedding) techniques.

Nonlinear problems usually have many discrete solutions, and the desired one is obtained only if we start from a close enough first approximation. A common way to obtain a good first approximation, or generally to trace branches of solutions, is by *continuation* (also called "embedding" or "Davidenko method"): The problem, including its discretization and its approximate solution is written as depending on some parameter

$$\gamma_0 \leq \gamma \leq \gamma_*$$

such that for γ_0 the problem is easily solvable (e.g., it is linear), while for γ_* it is the problem we really need to solve. We advance γ from γ_0 to γ_* in

steps $\delta\gamma$ small enough to ensure that the solution to the γ problem can serve as a good first approximation in solving the $\gamma + \delta\gamma$ problem. Sometimes γ is a physical parameter; sometimes the solutions are better defined in terms of a non-physical parameter, such as the arclength of the solutions path [Kel77]; cf. §5.6.

As for the relation between multigrid and continuation, several situations arise. Sometimes, the FMG algorithm is a good continuation process by itself. In particular, in non-elliptic and singular perturbation problems where relaxation adds $O(h)$ artificial viscosity, the FMG algorithm starts from highly viscous solutions (since h is large) and gradually eliminates viscosity as it proceeds to finer grids. This is a natural continuation path since problems with large viscosity terms are well-defined and easier to solve. This continuation is carried even much further when the FMG algorithm is continued to include *local* refinements around thin viscous layers (see §9).

Very often, instead of solving the problem several times for a sequence of γ values, the gradual changes of γ can similarly be integrated into just one FMG solver, advancing γ each time the solver proceeds to a new finest level.

An explicit continuation over the problem path $\gamma_0 \leq \gamma \leq \gamma_*$ should be made because the intermediate problems are either interesting in themselves, or necessary for reaching or even defining the desired γ_* solution. When the intermediate problems are not of interest, they can of course be solved to a lower accuracy, using coarser grids only. The grids cannot all be too coarse, however; the meshsize $\hat{h}$ must participate in the continuation process if components of wavelengths comparable to $\hat{h}$ are needed to keep the solutions in the "attraction region" of the desired solution path.

Even when components with $O(\hat{h})$ wavelengths are needed in the continuation process, in each $\delta\gamma$ step they do not usually change much. We can therefore employ the "frozen τ" techniques described below (§15), and perform most of the $\delta\gamma$ steps using, on most parts of the domain, very coarse grids, with only few "visits" to grid $\hat{h}$: Such a continuation process will often require less computational work than the final step of solving the γ_* problem, at which a higher accuracy is sought.

A favorite example where these techniques are put to test is the Bratu problem $\Delta u + \lambda e^u = 0$ in the unit square, with $u = 0$ on the boundary. In collaboration with Klaus Stueben we have solved the problem using FAS, freeing λ to be unknown and adding the value $u(P)$, where P is the center of the square, as a global constraint, treated basically as in §5.6. In this formulation, the multigrid solver had no problem going around the "limit point" ("turning point") of the solution curve (e.g., the curve of $u(P)$ as function of λ, which is not a unique function). It gave solutions at the same efficiency as corresponding algorithms for Poisson equations. In fact, we could solve by one-cycle FMG algorithm to the level of truncation errors, even problems on the upper branch of the solution curve, and even without

continuation at all. The only region where more lengthy calculations were needed was a region where the discrete solution bifurcated, a phenomenon the algorithm was not designed to deal with efficiently. See more details in [ST82, §5.5.1]. A further work along these lines, also to other problems, is reported in [Sch83].

8.4 Estimating truncation errors. τ-extrapolation

As with other defect-correction schemes, the defect can serve also as an error estimator. That is, τ_h^H is an approximation to the local truncation error τ^H (on the coarse grid H), defined by

$$\tau^H = L^H\left(\hat{I}^H u\right) - I^H\left(Lu\right), \tag{8.11}$$

where u is the true differential solution and I^H and $\hat{I}^H$ are two continuum-to-H transfer operators, defined as follows. I^H is the operator used in our grid-H discretization for $f^H = I^H f$, and $\hat{I}^H$ represents the sense in which we want u^H to approximate u: We want u^H to actually approximate $\hat{I}^H u$. The injection $(\hat{I}^H u)(x^H) = u(x^H)$ is usually meant, but other local averaging are sensible too.

Note the analogy between (8.11) and (8.9). τ^H is the correction to grid-H right-hand side that would make the grid-H solution coincide with the true differential solution $\hat{I}^H u$, while τ_h^H is the correction that would make it, at convergence, coincide with the fine-grid solution $I_h^H u^h$. It is hence clear that at convergence

$$\tau^H \approx \tau^h + \tau_h^H, \tag{8.12}$$

where τ^h is the fine-grid local truncation error, defined with $\hat{I}^h$ such that $\hat{I}^H = \hat{I}_h^H \hat{I}^h$. The sign $\approx$ means an equality up to a higher order in h. Relation (8.12) means, more precisely, that if $\tau^h + \tau_h^H$ were used to correct the H-equations, then u^H would be a higher-order approximation to $\hat{I}^H u$; namely, $u^H - \hat{I}^H u$ would equal $w^H - \hat{I}^H w^h$, where $L^H w^H = I_h^H \tau^h$ and $L^h w^h = \tau^h$.

Relation (8.12) can be used to inexpensively raise the approximation order. If the local approximation order (order of consistency) at the point x is p, i.e., if $\tau^h(x) \approx c(x)h^p$, where $c(x)$ is independent of h, then $\tau^H(x) = 2^p c(x)h^p$, hence $\tau_h^H(x) \approx (2^p - 1)c(x)h^p$, and hence $\tau^H(x) \approx 2^p(2^p-1)^{-1}\tau_h^H(x)$. To raise the approximation order all we have then to do is to change the grid-H equations (8.5) by writing them as in (8.8) and multiplying τ_h^H by the fixed factor $2^p(2^p-1)^{-1}$ This operation is called τ **extrapolation**. It resembles Richardson extrapolation, but it can profitably be done even in cases the latter cannot (e.g., in cases $p = p(x)$ is not constant), because it extrapolates the equation, not the solution.

τ-extrapolation can be shown to be a special case of the higher-order techniques of §10.2 below, but it is especially simple and inexpensive. It costs only one multiplication on the *coarser* grid. It is probably best to use it in an FMG algorithm with $W(\nu, O)$ cycles, since a terminal relaxation with the lower order discretization would impair the approximation (even though its order would remain higher). An option for τ-extrapolation exists in the model program FMG1 [MUG84].

Because of the analogy to the local truncation errors, τ_h^H is also called the **relative local truncation error** – the local truncation error of grid H relative to grid h. It is a by-product of the FAS processing which can be used to estimate the true local truncation errors: $\tau^h \approx (2^p-1)^{-1}\tau_h^H$. Hence it can be used in FMG stopping criteria (see §7.3) and in grid adaptation criteria (§9.5, 15).

8.5 FAS interpolations and transfers

The consideration in determining the residual transfer I_h^H, the correction interpolation I_H^h and the FMG interpolation $\mathbb{I}_H^h$ in FAS are basically the same as in CS, but there are some additional possibilities and we should also specify now the FAS solution transfer $\hat{I}_h^H$.

For linear problems the choice of $\hat{I}_h^H$ does not matter of course; all choices will give identically the same results. The solution efficiency of many nonlinear problems is also insensitive to the exact choice of $\hat{I}_h^H$. The choice does matter only where the problem coefficients drastically vary over a meshsize. By the "problem coefficients" we mean those of the linearized problem. In practice, using FAS , we do not linearize the problem, but the transferred solution $\hat{I}_h^H u^h$ implicitly determines the problem coefficients on the coarse grid H – determining the coefficients may in fact be regarded as the *purpose* of this transfer (although, unlike the CS situation, the coefficients can *change* on grid H, with the changing approximation). When the problem coefficients are highly variable, each coarse grid coefficient should be a suitable average of neighboring values of the corresponding fine-grid coefficients. The coarse-grid problem, in other words, should be a proper "homogenization" of the fine-grid problem. Such homogenization is usually obtained by using full weighting for $\hat{I}_h^H$ (as for I_h^H in (4.5)–(4.6)).

In some, very special situations the dominant solution-dependent term in the coefficients may have the form $g(u)$, where g is a sensitive function; large changes in g are caused by more-or-less normal changes in u over a meshsize. In such a case the weighting $\hat{I}_h^H$ should have the special form

$$\hat{I}_h^H \tilde{u}^h = g^{-1} \bar{I}_h^H \left(g\left(\tilde{u}^h\right)\right), \tag{8.13}$$

where $\bar{I}_h^H$ is a normal full weighting, such as (4.6), $g(\tilde{u}^h)$ is a grid function such that $(g(\tilde{u}^h))(x^h) = g(\tilde{u}^h(x^h))$ for every fine-grid point x^h, and g^{-1} is the inverse function of g; that is, $g^{-1}(g(\alpha)) = \alpha$ for any value α. If several

sensitive functions such as g appear in the coefficients, several $\hat{I}_h^H$ may have to correspondingly be used. (So far we have not seen a practical problem where this was required.)

An important possibility offered by FAS is the usage of a special type of interpolation near a nonlinear interior discontinuity, such as a shock or an interface. The grid-H solution, introducing smooth changes to the grid-h solution, may change the **location** of such a discontinuity, moving it a meshsize or two. Near the discontinuity, the correction $\underline{v}^H = \tilde{u}^H - \hat{I}_h^H \tilde{u}^h$ will then be highly non-smooth; it will look like a pulse function. Interpolating it as a correction to the fine grid will introduce there unintended high oscillations. To avoid this, the FAS interpolation (8.6) should be replaced by (8.7) near the discontinuity. This is easy to implement, by adopting the following, more general rule.

> *Use (8.6) everywhere except near points where $\tilde{u}^H - \hat{I}_h^H \tilde{u}^h$ is comparable to $\tilde{u}^h$, where (8.7) should be used.*

8.6 Application to integral equations

When the integral equation

$$\int K(x,y)u(y)dy = f(x,u(x)) \tag{8.14}$$

is discretized in a usual way on a grid with $n = O(h^{-d})$ points, the unknowns are all connected to each other; the matrix of the (linearized) discrete system is full. A solution by elimination would require $O(n^3)$ operations. An FMG solution would require $O(n^2)$ operations, since each relaxation sweep costs $O(n^2)$ operations. In case (8.14) is nonlinear in u, FAS-FMG would be used. Using the FAS structure, even for linear problems, we can often reduce this operation count very much, by exploiting smoothness properties of K.

In most physical applications $K(x,y)$ has a singularity at $y = x$. So usually $K(x,y)$ becomes either smaller or smoother as the distance $|y-x| = (\sum_{j=1}^{d}(y_j - x_j)^2)^{\frac{1}{2}}$ increases. "Smaller" and "smoother" are in fact related: The former is obtained from the latter by differentiations of (8.14) with respect to $x = (x_1, \ldots, x_d)$ (possibly replacing the integral equation by an integro-differential equation). In either case, one can obtain practically the same accuracy in the numerical integration using meshsizes that increase with $|y - x|$, cutting enormously the work involved in relaxation. Usually u(y) is much less smooth than $K(x,y)$ for large $|y - x|$. The integration with increasing meshsizes cannot then use point values of $\tilde{u}^h$, but should use local averages of $\tilde{u}^h$, taken over boxes whose size increases with $|y - x|$. Exactly such averages are supplied by the sequence of coarser grids in the FAS structure. The FAS solution transfers $\hat{I}_h^H$ should of course represent full weighting. One can increase the accuracy of integration by using, in

addition to the full-weighting averages, higher local moments, represented on additional coarser grids. These techniques have been demonstrated in [BL90] and subsequent works.

8.7 Small storage algorithms

Various effective methods for vastly reducing the storage requirement of the discrete solution without using external storage, can be based on the Full Approximation Scheme. One simple method [Bra79b, §2.2] is to use the fact that a problem whose finest grid is h can satisfactorily be solved by an FMG algorithm with only one h-cycle (see §7.3). This means that only one visit is needed to grid h, including the FMG interpolation I_H^h, a couple of relaxation sweeps, and the residual and solution transfers I_h^H and $\hat{I}_h^H$ back to grid H. All these operations can be made "wave-like" by just one pass over grid h, requiring no more than few columns at a time kept in memory. (When the operations of one relaxation sweep have been completed up to a certain column, the operations of the next sweep can immediately be completed on the next column, etc.) This visit is enough to supply the corrected right-hand side $\hat{f}^H$ on grid H (cf. §8.2), hence enough to calculate $\tilde{u}^H$ without any storage allocated to grid h, except for the few mentioned continuously shifted columns.

$\tilde{u}^H$ is as precise as $\tilde{u}^h_{\text{NEW}}$. The usual terminal sweeps of the h cycle are only done if we need the solution on grid h, their role is to *smooth* the interpolation errors, not to *reduce* the error. Moreover, suppose that what we really need from our calculations is some functional of the solution, $\Phi(u)$ say, so we would like to calculate $\Phi^h(\tilde{u}^h_{\text{NEW}})$. All we have to do is to calculate, incidentally to transferring the solution u^h back to grid H, the values of both $\Phi^h(\tilde{u}^h)$ and $\Phi^H(\hat{I}_h^H \tilde{u}^h)$. Then, having later obtained $\tilde{u}^H$, we can calculate

$$\hat{\Phi}^H(\tilde{u}^H) = \Phi^H(\tilde{u}^H) + \Phi^h(\tilde{u}^h) - \Phi^H(I_h^H \tilde{u}^h), \tag{8.15}$$

which is practically as accurate as $\Phi^h(\tilde{u}^h_{\text{NEW}})$, because $\tilde{u}^H - \hat{I}_h^H \tilde{u}^h$ is small and smooth.

This simple procedure reduces the required storage typically by the factor $2^d - 1$, without increasing the computational work. Other procedures can reduce the storage much further by avoiding the storage of coarser grids too, except for a certain $(n_k h_k) \times (n_k h_k)$ box on each grid $h_k = 2^k h$. The h_k box is shifted within the h_{k-1} box to supply the $\tau_{h_k}^{h_{k-1}}$ corrections. The amount of work increases since on "revisiting" the h_k box we need to reproduce its own $\tau_{h_{k+1}}^{h_k}$ corrections. This can be done only in the interior of the box, distance $O(h_k|\log\varepsilon|)$ from the boundary of the box, where $\varepsilon = O(h^p)$ is the desired accuracy on the finest grid, because closer to that boundary the h_{k+1} high frequency errors are larger than the desired accuracy. Hence we must have $n_k \geq O(|\log\varepsilon|)$. The overall required storage can therefore

be reduced to $O(|\log \varepsilon|^d |\log h|)$ (not just $O(|\log h|)$, as mistakenly calculated in [Bra77a, §7.5]). Such procedures are called *segmental refinement techniques.*

Another small-storage multigrid algorithm, not based on FAS, is described in [Hac80]. It is a region dissection procedure, particularly suited for elongated domains.

Chapter 9

Local Refinements and Grid Adaptation

Non-uniform resolution is needed in many, perhaps most, practical problems. Increasingly finer grids are needed near singularities, near non-smooth boundaries, at boundary layers, around captured shocks, etc., etc. Increasingly coarser grids are needed for exterior problems on unbounded domains. The multi-level FAS approach gives a convenient way to create non-uniform adaptable structures which are very flexible, allowing fast local refinements and local coordinate transformation, and whose equations are still solved with the usual multigrid efficiency. Moreover, the grid adaptation can naturally be governed by quantities supplied by the FAS multigrid processing, and it can naturally be integrated with the FMG algorithm to give increasingly better approximations to the true *differential* solution, at a fast, nearly optimal rate. These techniques, outlined below, are described in more detail in [Bra77a, §7, 8, 9], [Bra77b, §2, 3, 4], [Bra79b, §3, 4]. An application to three-dimensional transonic flows is described in [Bro82].

Another highly flexible discretization using a multigrid solver [Ban81] is based on a finite element approach, which makes the program simpler – especially for complicated structures, but the execution is less efficient. The techniques outlined below are also applicable to finite element formulations as in [Bra79a], [B+78, Lecture 4].

9.1 Non-uniformity organized by uniform grids

Our non-uniform discretization grows from the simple observation that the various grids (levels) used in usual multigrid algorithms need not all extend over the same domain. The domain covered by any grid may be only a proper part of the domains covered by coarser grids. Each grid h can be extended only over those subdomains where the desired meshsize is roughly less than $2h$. In such a structure, the effective meshsize at each

neighborhood will be that of the finest grid covering it: see Fig. 9.1.

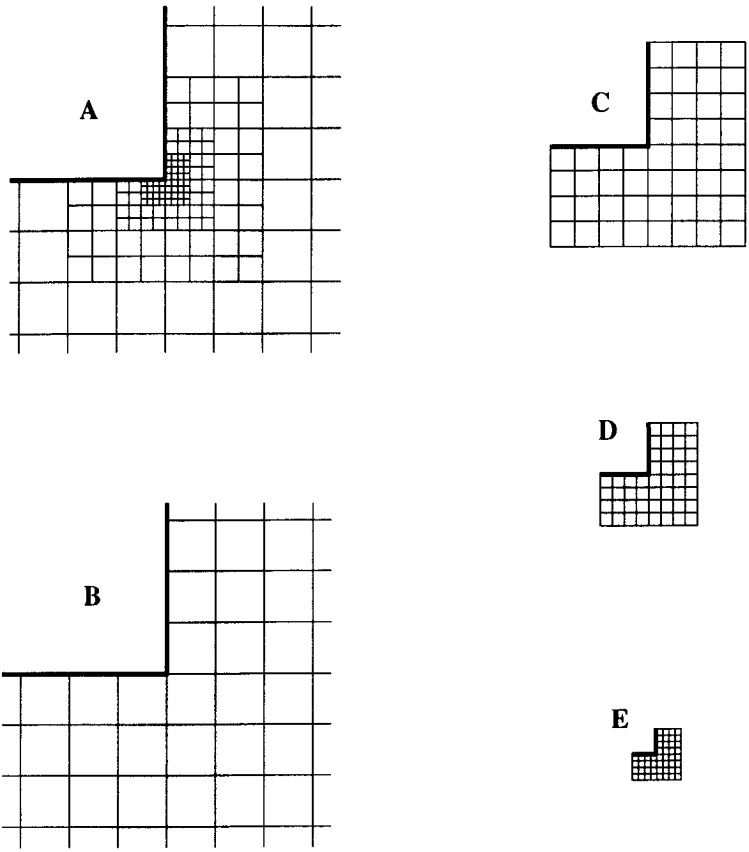


Figure 9.1. ***A piece of non-uniform grid (A) and the uniform levels it is made of (B,C,D,E).***
The heavy line shows a piece of the boundary, with a reentrant corner calling for the local refinements produced by the local patches (C,D,E).

This structure is very flexible, since local grid refinements (or coarsening) is done in terms of extending (or contracting) uniform grids, which is relatively easy and inexpensive to implement. A scheme named GRIDPACK [MUG84] has been developed for constructing, extending and contracting general uniform grids, together with many service routines for such grids, including efficient sweeping aids, interpolations, displays, treatment of boundaries and boundary data, etc. It is fully described in [BO83]. One of its advantages is the efficient storage: The amount of logical information (pointers) describing a uniform grid is proportional to the number of *strings* of points (contiguous sets of gridpoints on the same gridline), and is therefore usually small compared with the number of points on the grid.

Similarly, the amount of logical operations for sweeping over a grid is only proportional to the number of strings. Changing a grid is inexpensive too. One can easily add finer levels, or extend existing ones, thus effecting any desired refinements.

Moreover, this structure will at the same time provide a very efficient **solution process** to its difference equations, by using its levels also as in a multigrid solver. For this purpose the full approximation scheme must be used, because in parts of the domain not covered by the finer grid h, the coarser grid $H = 2h$ must certainly show the full solution, not just a correction. Indeed, the FAS approach naturally fit here: We use on grid H the equations $L^H u^H = \hat{f}^H$, where $\hat{f}^H$ is given by (8.5b) wherever $L^h u^h$ is well defined (i.e., in the interior of grid h), and $\hat{f}^H = f^H$ otherwise. In other words (cf. (8.8)-(8.9)), the fine-to-coarse correction τ_h^H is simply canceled wherever it is not defined. Applying an FMG algorithm with these structures and equations, we will get a solution that in each subdomain will satisfy its finest-grid equations, while at interior boundaries (of fine levels not covering the entire domain) the solution will automatically be as interpolated from the coarser grid. Note that the coarse-grid solution is influenced by the finer grid even at regions not covered by the latter, since the coarse grid *equations* are modified in the refined region.

In other words, a patch of the next finer level h can be thrown on any part of a given grid $H = 2h$, correcting there the latter's equations to the finer-grid accuracy. Moreover, several such patches may be thrown on the same grid. Some or all of the patches may later be discarded, but we can still retain their τ_h^H corrections in the grid H equations.

An important advantage is that difference equations are in this way defined on uniform grids only. Such difference equations on equidistant points are simple, inexpensive and standard, even for high-order approximations, whereas on general grids their weights would have to be calculated by lengthy calculations separately for each point. Relaxation sweeps are also made on uniform grids only. This simplifies the sweeping and is particularly useful for line relaxation schemes.

9.2 Anisotropic refinements

It is sometimes desired to have a grid which resolves a certain thin layer, such as a boundary layer. Very fine meshsizes are then needed in one direction, namely, across the layer, to resolve its thin width. Even when the required meshsize is extremely small, not many gridpoints are needed, since the layer is comparably thin, provided, of course, that fine meshsizes are used only in that one direction. We need therefore a structure for meshsizes which get finer in one direction only.

In case the thin layer is along coordinate hyperplane $\{x_j = \text{const.}\}$, this is easily done by **semi refinements**: Some levels H are refined by the next level h only in the j coordinate, $h_j = H_j/2$ whereas $h_i = H_i$ for

$i \neq j$. See Fig. 9.2. In fact, different patches may have different refinement directions. Thus, the set of all grids is arranged logically in a *tree*, each grid having a unique next-coarser grid, but possibly *several* next-finer grids, instead of the former linear ordering. All these grids are still uniform, and can still easily be handled by GRIDPACK.

Note that the next-finer grids of a given grid H may have some **overlap**. All that is needed in such cases is to get priority relations, to tell which correction τ_h^H applies at each point of grid H. Such priority relations are simply set by the order in which the corrections are transferred to grid H.

In case the thin layer is not along coordinate lines, the methods of the following sections could be used.

9.3 Local coordinate transformations

Another dimension of flexibility and versatility can be added to the above system by allowing each of the local patches to have its own set of local coordinates.

Near a boundary or an interface, for example, the most effective discretization is made in terms of coordinates in which the boundary (or interface) is a coordinate line. In such coordinates it is much easier to formulate high-order approximations near and on the boundary, and to introduce meshsizes which are much smaller across than along the boundary layer (§9.2); etc. In the interior, local patches of coordinates aligned with characteristic directions (along streamlines, for instance) can greatly reduce the cross-stream numerical viscosity (cf. §2.1), thus yield superior approximations to non-elliptic equations.

Each set of coordinates will generally be used with more than one grid, so that (a) local refinements, isotropic or anisotropic, in the manner described above, can be made within each set of coordinates; and (b) the multigrid processing retains its full efficiency by keeping the meshsize ratio between any grid and its next-coarser one properly bounded.

Since local refinement can be made within. each set of coordinates, the only purpose of the coordinate transformation is to provide the grid with the desired orientation, i.e., to have a given manifold (such as a piece of the boundary) coincide with a grid hyperplane. Since, furthermore, this needs to be done only locally, it can be obtained by a simple and standard transformation. For example, in two dimensional problems, let a curve be given in the general parametric form

$$x = x_0(s), \quad y = y_0(s), \quad (s_1 \leq s \leq s_2) \tag{9.1}$$

where s is the arclength, i.e.

$$x_0'(s)^2 + y_0'(s)^2 = 1. \tag{9.2}$$

To get a coordinate system (r, s) in which this curve coincides with the grid

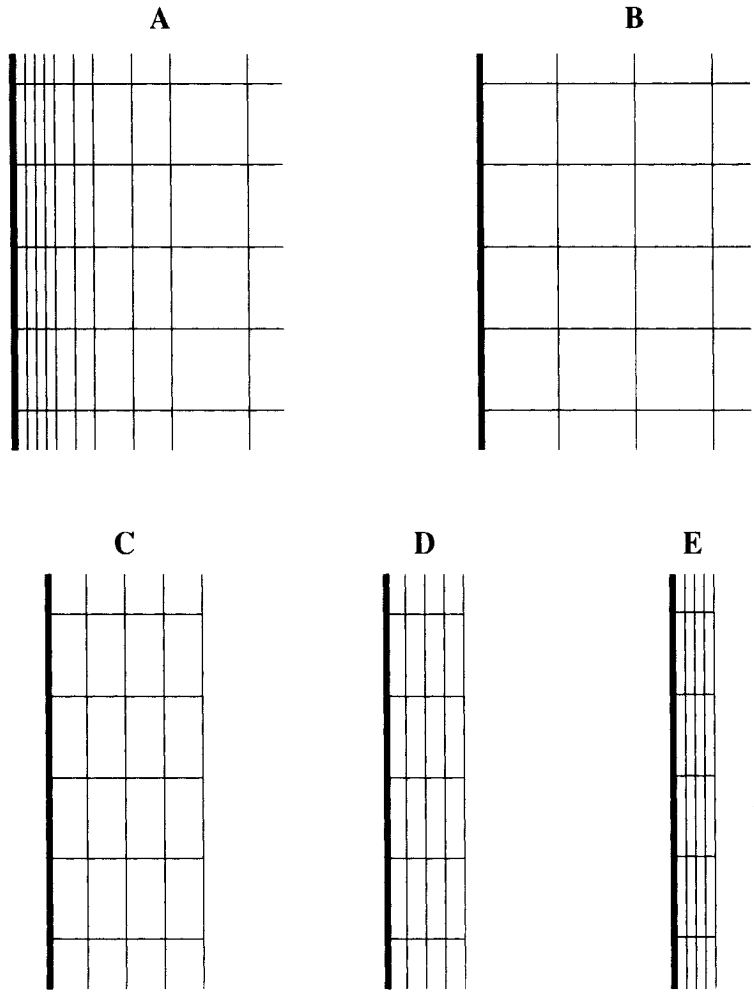


Figure 9.2. ***A piece of non-uniform, boundary-layer type grid (A) and the uniform rectangular subgrids it is made of (B,C,D,E).***

The meshsize in the local patches (C,D,E) is halved horizontally only.

line $\{r = 0\}$, we use the following transformation as standard:

$$x(r,s) = x_0(s) - ry_0'(s), \quad y(r,s) = y_0(s) + rx_0'(s). \tag{9.3}$$

Locally, near $r = 0$, this transformation is isometric (simple rotation).

The main advantage of this transformation is that it is fully characterized by the single-variable functions $x_0(s)$, $y_0(s)$. These functions, together with $x_0'(s)$, $y_0'(s)$ and $q(s) = x_0''/y_0' = -y_0''/x_0'$ can be stored as one-dimensional arrays, in terms of which efficient interpolation routines

from (x, y) grids to (r, s) grids, and vice versa, can be programmed once for all. The difference equations in (r, s) coordinates are also simple to write in terms of these stored arrays, since, by (9.2)–(9.3),

$$\frac{\partial}{\partial x} = -y_0' \frac{\partial}{\partial r} + \frac{x_0'}{1 + rq} \frac{\partial}{\partial s}, \quad \frac{\partial}{\partial y} = x_0' \frac{\partial}{\partial r} + \frac{y_0'}{1 + rq} \frac{\partial}{\partial s}. \tag{9.4}$$

A different kind of multi-level procedure using a combination of cartesian grids and grids curved along boundaries is described in [ST82, §11]. The main difference is that all levels, from coarsest to finest, are used there both for the cartesian and for the curved grids, and at each level the relaxation includes interpolations between the two types of grids, while the present approach is to regard the curved grids as a finer level which correct the finest cartesian grid near the boundary. The present approach is perhaps more economic and flexible, but it requires a (crude) approximation to the boundary conditions to be given on the cartesian grids, too.

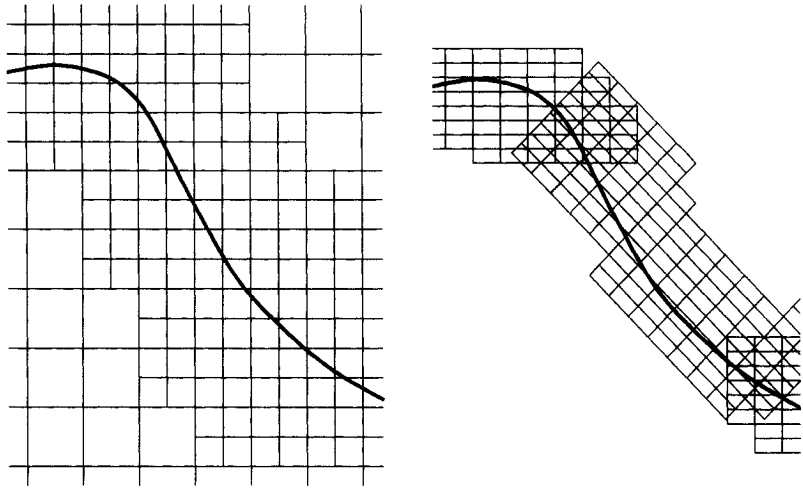
9.4 Sets of rotated cartesian grids

Another variant of this procedure is required in case the location of the thin layer (interface, shock, etc.) is not fully defined. For this purpose, each level will be a set of rotated cartesian grids, possibly overlapping. The finer the level, the finer (richer) is also the set of rotations. See Fig. 9.4. The self-adaptive criteria (see §9.5) can be employed to decide where to refine the set of rotations (together with refining the meshsize in one direction). Hence the scheme can capture discontinuities (thin layers), without defining them as such. The stronger the discontinuity, the better its resolution.

Since only rotated cartesian grids are needed in this scheme, the finite difference equations are as simple as ever. Hence this method is sometimes preferable even in cases where the location of the thin layer is known.

9.5 Self-adaptive techniques

The flexible organization and solution process, described above, facilitate the implementation of variable meshsize $h(x)$ and the employment of high and variable approximation order $p(x)$. How, then, are meshsizes and approximation orders to be chosen? Should boundary layers, for example, be resolved by the grid? What is their proper resolution? Should high-order approximations be used at such layers? How does one detect such layers automatically? In this section we survey a general multigrid framework for automatic selection of $h(x)$, $p(x)$ and other discretization parameters in a (nearly) optimal way. This system automatically resolves or avoids from resolving thin layers, depending on the goal of the computations, which can be stated through a simple function. (For more details see [Bra77a, §8], [Bra77b, §3]).



(A) (B)

Figure 9.3. ***Grid orientation around an interior thin layer.*** *The two coarsest levels (A) have the usual orientation* 0*. The next level (B) has four possible orientations:* $-\frac{\pi}{2}$, $-\frac{\pi}{4}$, 0 *and* $\frac{\pi}{4}$ *(only the last two are applied here). The next level (not shown) admits eight orientations,* $\frac{k\pi}{8}$, $-4 \leq k < 4$*; etc. The descendant refinements of a grid will always either have the same or one of the two nearest orientations (e.g., each descendant of the* $\frac{\pi}{4}$*-oriented grid at level B will either be* $\frac{\pi}{4}$*-,* $\frac{\pi}{8}$*-, or* $\frac{3\pi}{8}$*-oriented).*

As our directive for sensible discretization we consider the problem of minimizing a certain error estimator E subject to a given amount of solution work W (or minimizing W for a given E. Actually, the practical control quantity will often be neither E nor W, but their rate of exchange $\lambda = -dE/dW$, as shown below). This optimization problem should of course be taken quite loosely, since full optimization would require too much control work and would thus defeat its own purpose.

The error estimator E has generally the form

$$E = \int_{\Omega} G(x)\tau^h(x)dx, \tag{9.5}$$

where $\tau^h(x)$ is the local truncation error (cf. (8.11)) at x. $G(x) \geq 0$ is the error-weighting function. It should in principle be imposed by the user, thus defining his goal in solving the problem. In practice $G(x)$ serves as a convenient control. It is only the relative orders of magnitude of $G(x)$ at different points x that really matter, and therefore it can be chosen by some simple rules. For example, if it is desired to compute l-order derivatives of the solution up to the boundary then $G(x) \approx d_x^{m-1-l}$, where d is the

distance of x from the boundary, and m is the order of the differential equation.

The work functional W is roughly given by

$$W = \int_\Omega \frac{w(p(x))}{h(x)^d} dx, \tag{9.6}$$

where d is the dimension and h^{-d} is therefore the number of gridpoints per unit volume. (Replace h^d by $h_1 \cdots h_d$ in case of anisotropic grids.) $w = w(p)$ is the solution work per gridpoint. In multigrid processing, this work depends mainly on the approximation order (consistency order) $p(x)$. If the high-order techniques of §10 are used then usually $w(p) \approx w_0 p$, although sometimes $w(p) = O(p^3)$ for unusually high p (see §10.1).

Treating $h(x)$ as a continuous variable, the Euler equations of minimizing (9.5) for fixed (9.6) can be written as

$$G\frac{\partial \tau}{\partial h} - \lambda d w(p) h^{-d-1} = 0, \tag{9.7}$$

where λ is a constant (the Lagrange multiplier), representing the marginal rate of exchanging optimal accuracy for work: $\lambda = -dE/dW$.

In principle, once λ is specified, equation (9.7) determines, for each $x \in \Omega$, the local optimal values of $h(x)$, provided the truncation $\tau^h(x)$ as a function of h is fully known. In some problems the main behavior of $\tau^h(x)$ near singularities or in singular layers is known in advance by some asymptotic analysis so that approximate formulae for h(x) can a-priori be derived from (9.7). (Near source-type singularity (9.7) should be modified for that purpose, since $\tau^h(x)$ has an essential and singular sign reversal at the source [BB87].) More generally, however, equation (9.7) is coupled with, and should therefore be solved together with, the given differential equations. Except that (9.7) is solved to a cruder approximation. This is done in the following way:

In the FAS solution process we readily obtain the quantity τ_h^H. By (8.12) and (9.5), the quantity $-\Delta E(x) = G(x)\tau_h^H(x)$ can serve as an estimate for the decrease in E per unit volume owing to the refinement from H to h in the vicinity of x. By (9.6), this refinement requires the additional work (per unit volume) $\Delta W = w(p)h^{-d}(1 - 2^{-d})$. The local rate of exchanging accuracy for work is $Q = -\Delta E/\Delta W$. If Q is much larger than the control parameter λ, we say that the transition from H to h was highly profitable, and it pays to make a further such step, from h to $h/2$. So we will next establish grid $h/2$ in the neighborhood of x, as in any other neighborhood where there are points with $Q \gg \lambda$.

The computer work invested in the test is negligible compared with the solution work itself, since Q is calculated by just a couple of operations per point on the coarser grid H once per cycle. A similar test can be used to decide on changing the local approximation order $p(x)$ with τ_H^h being replaced by the p_1-to-p_0 defect correction (10.1) and correspondingly

$\Delta W = (w(p_1) - w(p_0))h^{-d}$. Or, if we treat p as a constant over the domain, but we like to optimize that constant, we can measure ΔE globally; i.e., measure directly the change in some quantity of interest (e.g., some functional of the solution we are most interested in), due to the transition from p_1 to p_0. Correspondingly, global ΔW will be used. Whether locally or globally, the order will be increased beyond p_1 if $-\Delta E/\Delta W \gg \lambda$. Other discretization parameters, such as the computational boundaries (when the physical domain is unbounded), or refinements in grid orientations (see §9.4), can be decided by similar tests all based on comparing some $-\Delta E/\Delta W$ to the exchange-rate parameter λ. How to control λ and coordinate it with the solution algorithm is discussed in the next section.

9.6 Exchange rate algorithms. λ-FMG

Near a severe singularity many levels of increasingly finer grids on increasingly narrower subdomains may be needed, and formed by the above criteria. If the usual FMG algorithm were applied to these levels, too much work would be spent, since too many passes on coarser grids would be made. Only when all grids cover the same domain is it true that the coarse-grid work is small compared to the next-finer grid work, since the latter deals with about 2^d as many points. This is no longer so when local refinements are used: Finer grids may include *less* points than some coarser grids. The amount of work in a usual FMG algorithm would therefore be much greater than proportional to the total number of gridpoints; (9.6) would not hold.

A better procedure is to decrease the accuracy-to-work exchange rate λ in a gradual sequence $\lambda_1 > \lambda_2 > \cdots$, and to use the solution obtained for λ_{j-1} as the first approximation to the solution on the grids formed for λ_j. In the absence of singularities, and for uniformly p-order approximations, this process with the ratio $\lambda_j/\lambda_{j-1} = 2^{-p-d}$ would yield the regular FMG algorithm, so generally it is called λ-FMG. It was tested [BB87] for the Poisson equation $\Delta u = f$, with severe singularity in f (e.g., $f = r^{-3.5}$, where r is the distance to a certain boundary point), or with 2π reentrant corner. The ratio $\lambda_j/\lambda_{j-1} = 1/16$ was used, for each λ_j the collection of local refinements was determined by (9.7), using the roughly known behavior of τ^h as function of the distance from the singularity. The 5-point Laplacian was employed, with red-black Gauss-Seidel relaxation. In the reentrant corner case, it was essential to use the local relaxation technique (§5.7). Results were invariably excellent: Algebraic errors $\|\tilde{u}^\lambda - u^\lambda\|$ smaller than the truncation errors $\|u^\lambda - u\|$ were obtained by one V(1,1) cycle per λ. More importantly, *the differential error* $E = \|\tilde{u}^\lambda - u\|$ *as function of the total work* W (measured as the total number of points traversed in all the relaxation sweeps) *behaves the same as in regular cases*: In regular problems with second-order approximations on two-dimensional uniform grids $E = O(h^2) = O(W^{-1})$, and the above experiments indeed clearly yield $E = O(W^{-1})$, for several orders of increase in W. This confirms

the validity not only of the λ-FMG algorithm, but also of the adaptation criterion (9.6). Had we used only uniform grids, the singularity would severely cripple $E(W)$; e.g., in the reentrant corner case it would yield $E = O(h) = O(W^{-\frac{1}{2}})$.

Switching criteria based on the exchange rate λ could also be used *in the multigrid cycles themselves.* Typically, the algorithm would continue relaxing on a given grid wherever $\hat{Q}(x) := -G(x)\Delta r(x)/\Delta W(x) > a_1\lambda$, where $-\Delta r(x)$ is the local decrease in the residual $|r(x)|$ per gridpoint per sweep and $\Delta W(x)$ is the corresponding work. Wherever uniformly over a substantial subdomain, $\hat{Q}(x) \lesssim a_1\lambda$ but $\overline{Q}(x) := -G(x)|r(x)|/\Delta W(x) > a_2\lambda$, a switch would be made to the next coarser grid. Wherever $\overline{Q}(x) < a_2\lambda$, a switch should in principle be made to the next finer level; if the current level is already locally the finest one, the processing for the current λ should terminate. Note that although for theoretical optimality different switches should thus in principle be required at different subdomains, for practical simplicity such a fragmentation of the algorithm should most often be avoided, except for the case of continuing relaxation locally (as mentioned in §5.7).

Ultimately, such exchange-rate criteria unify all the multigrid switching and adaptation criteria, integrating them into one algorithm, in which λ is gradually decreased. The process can be continued indefinitely, with increasingly finer levels created, globally and locally, in an almost optimal way. It can be terminated when either E or W or λ reach preassigned limits. The above techniques of anisotropic grids, local transformations and rotations, and adaptation of approximation orders can all be integrated into such an exchange-rate algorithm.

Chapter 10

Higher-Order Techniques

A sound way of constructing high-order approximations to a given differential problem $LU = F$, is first to construct a multigrid program with a low approximation order, and then convert it into a high-order program. The lower order is easier to develop and is also useful as a component in the higher-order program. Such programs are usually more efficient than programs which use high-order difference operators throughout. We mostly recommend the method of §10.2 below, especially for non-elliptic and singular perturbation problems.

10.1 Fine-grid defect corrections. Pseudo spectral methods

Given a program for solving the (linear or nonlinear) low-order (order p_0) discrete system $L_0^h u^h = f^h$, an obvious multigrid approach for raising the approximation order is by high-order "deferred" (or "defect") corrections introduced once per cycle on the currently-finest grid [Bra79b, §3.4]. That is, we add to f^h the correction

$$\sigma_{1,0}^h(x^h) := L_0^h \tilde{u}^h(x^h) - L_1^h \tilde{u}^h(x^h), \tag{10.1}$$

where L_1^h is the higher-order operator, its approximation order (consistency order) being $p_1 > p_0$, and $\tilde{u}^h$ is the current approximate solution. A similar correction is of course introduced to the discrete boundary conditions, too. To save h-cycles one should employ an FMG algorithm (§7), and use corrections like (10.1) at all the FMG stages (i.e., for every currently-finest grid). The total amount of work is then still basically given by (7.4).

Note that that work is proportional to the approximation order p_1. However, this count does not take into account the calculation of (10.1) once per cycle. For lower p_1 this extra work may be less than the other

work within the cycle (a couple of sweeps on each level), but for high p_1 it becomes dominant and makes the amount of work per cycle proportional to p_1 (assuming spectral-type methods cannot be used and the complexity of calculating L_1^h is thus proportional to p_1), hence the total work is in principle $O(p_1^2)$. Furthermore, for higher p_1 we have in principle to use higher computer precision, making the work of each arithmetic operation (in calculating (10.1)) again proportional to p_1, bringing the total work to $O(p_1^3)$. This can be reduced to just $O(p_1^2)$ by a method of *compound deferred corrections*, i.e., taking $p_0 = p_1/2$ and solving for L_0^h by deferred corrections to a system of order $p_0/2$, and so on recursively. In the normally used range of p_1, however, the work of even the uncompounded deferred correction is often still dominated by relaxation and hence still proportional to p_1.

This technique can in particular be applied to **pseudo-spectral approximations** L_1^h, i.e., approximations attaining very high order (proportional to $1/h$) through a discrete spectral (Fourier, Chebyshev, etc.) representation of the solution, using fast numerical transformers (e.g., FFT – the Fast Fourier transform) to obtain that representation and to calculate from it the approximate derivatives at gridpoints (cf. e.g., [GO77]). Using reasonably high order in L_0^h (e.g., $p_0 = 4$, itself calculated by deferred corrections to a second-order operator) one can attain the spectral approximation order with just few applications of the spectral operator. (Other spectral multigrid methods are described in [BFT83], [SZH83], [ZWH82], [ZWH83].)

The deferred correction technique (suggested by L. Fox) is a special case of the concept of defect corrections (see [Lin76], [Ste78], [AS82]). An important advantage of such a technique is that the higher-order operator L_1^h (and the corresponding higher order boundary conditions) need not be stable. This gives much freedom in the relatively difficult task of calculating L_1^h. This freedom is especially welcome in nonelliptic and singular perturbation cases, where convenient central approximations are unstable.

The reason L_1^h need not be stable is that the convergence of the defect correction iterations, to the solution corresponding to L_1^h, is fast only in the smooth components (for which L_0^h is a good approximation to L_1^h) and is very slow in the high-frequency components. Since instability is a property of high-frequencies, it can creep in only very slowly. The growth of unstable modes within the few solution cycles is negligible.

The whole purpose of defect corrections is in fact to correct lowfrequency components; only for such components higher-order approximation, such as L_1^h, are much better than lower-order approximations like L_0^h. Recognizing this and the fact that in multigrid processes low frequencies are converged via the coarse-grid corrections, we see that the main effect of the defect corrections can be obtained by applying them only at the stage of transferring residuals to the coarser grids. This would save about two work-units per cycle, and would give better approximations in case L_1^h is unstable. This idea, from a different point of view, is described in the next

section.

10.2 Double discretization: High-order residual transfers

On any given grid participating in multigrid interactions, discrete approximations to the continuous operator L are used in two different processes: in relaxation sweeps, and in calculating residuals transferred to coarser grids. The two discretization schemes need not be the same [BD79, §3.11]. The discretization L_0^h employed in the relaxation sweeps must be stable (see §12), but its accuracy may be lower than the one we wish to generate. The discretization L_1^h used in calculating the transferred residuals determines the accuracy of our numerical solution, but it need not be stable. This "double discretization" scheme is especially useful in dealing with non-elliptic and singular perturbation problems: One can use the most convenient (but sometimes unstable) central differencing for L_1^h, and add artificial viscosities (see §2.1) only to L_0^h. This will ensure stable solutions which still have the accuracy of the central differencing.

Note that such a multigrid process will not converge to zero residuals, since it uses two conflicting difference schemes. The very point is, indeed, that the solution produced may be a better approximation to the *differential* solution than can be produced by any of the two schemes. More importantly, during running a double discretization solver, instead of checking algebraic convergence, *one should directly check convergence to the differential solution* through the sequence of solutions produced at different stages of the FMG algorithm (see §1.6).

The lack of algebraic convergence makes the usual two-level mode analysis irrelevant for double discretization schemes. Instead they can be analyzed by the two-level FMG mode analysis (§7.4).

Double discretization schemes can of course similarly be applied to **boundary conditions**; e.g., to Neumann conditions: Simple first-order schemes can be used in relaxation, while second-order Neumann conditions (which are sometimes complicated and may sometimes be unstable) can be used to transfer boundary-condition residuals to coarser grids.

The double discretization scheme need not be confined to the currently finest level; it can **also be used on coarser levels**. This will give better coarse-grid corrections, and hence faster algebraic convergence. (In non-elliptic and singular perturbation cases the algebraic convergence is usually determined by the quality of the coarse-grid correction [Bra81a, §5.1].) It is also more convenient to program, since the same residual transfer routine, based on L_1^h, is used on all levels.

Moreover, if only L_0^h is used on coarser levels, the gain in approximation order per cycle cannot be more than p_0; hence the final approximation order cannot exceed $2p_0 + r_0$, where r_0 is the convergence order of relax-

ation [Bra81a, §2.2]. Such a restriction does not exist if L_1^h is used for residual transfers on all levels. The approximation order p_1 can then be attained, perhaps even in one cycle, no matter how high p_1 is. In particular, pseudo-spectral approximations can be used in L_1^h, yielding very high approximation orders in few cycles.

In order to obtain the high approximation orders several rules should be observed: Suitable interpolation orders and residual transfer orders should be employed. The right orders can be derived by crude mode analysis, as in §4.3, but with particular attention to boundary (see in particular rule (C) in §4.3). FMG algorithms with $W(\nu, 0)$ cycles should be used (see §6.2), to ensure accurate enough solution of the course-grid equations and to avoid degradation of the approximation by terminal relaxation. Also, when double discretization is used on all levels together with the Full Approximation Scheme (see §8), notice that two different right-hand sides should be used on coarser grids, one for relaxation and a different one for residual transfers [Bra81a, §2.1].

In case L_1^h is a better approximation than L_0^h not only for smooth components but also in the high-frequency range, the method of fine-grid defect corrections (§10.1) will eventually give smaller errors than the coarse-grid defect correction described here. But the gain will hardly justify the extra work involved in calculating (10.1) separately from the calculation of residuals. In problems where L_1^h is unstable, the present method is both faster and more accurate.

Double discretization schemes have already been used successfully in various cases, including fourth and sixth order approximations to Poisson equation [Sch82]; second-order approximations to simple singular perturbation problems [Bra77a, §7], [Bör81, §7]; and second-order approximations to incompressible Navier-Stokes equations with high Reynolds numbers. Also, the λ extrapolation (§8.4) can be viewed as a special case, where $L_1^h = (2^{p_0} L^h - L^{2h})/(2^{p_0} - 1)$.

10.3 Relaxation with only subprincipal terms

A particularly useful application of the above techniques is to employ a simple relaxation operator L_0^h where non-principal terms are neglected; more precisely, to employ the simplest stable L_0^h which approximate the subprincipal terms of the differential operator (see §2.1). Other terms need to be approximated only in L_1^h. For some fluid-dynamics systems this procedure can save a substantial amount of work. The techniques of either §10.1 or §10.2 can be used with this relaxation; more work is saved by the latter, but the former is safer. On very coarse grids, as the non-principal terms become more important, this type of relaxation may give worse performance. In such cases use more sweeps or reintroduce the neglected non-principal terms.

Chapter 11

Coarsening Guided By Discretization

The term "coarsening" is used here for the entire process of transferring a residual problem $L^h v^h = r^h$ from a fine grid h to the next coarser grid $H(= 2h)$. This includes the formulation of the coarsegrid problem $L^H v^H = I_h^H r^h$, where the coarse-grid operator L^H and the fine-to-coarse transfer I_h^H should be determined both in the interior and near boundaries, and similar equations should be transferred for the boundary conditions themselves, and for any other side conditions the problem may have. Any sufficiently general method of coarsening implies a discretization method, in the following sense: If the differential problem $Lu = f$ is discretized by any method, giving the problem $L^h u^h = f^h$ on grid h, and if this problem is then successively coarsened to $L^{2h} u^{2h} = f^{2h}$, $L^{4h} u^{4h} = f^{4h}$, etc. by successive applications of that same coarsening method, then in the limit (for a sufficiently coarse grid) we obtain a discretization of $Lu = F$ which does not depend on the original discretization $L^h u^h = f^h$, but only on the method of coarsening. The limit discretization, in this sense, is the **fixed point of the coarsening method**. (In practice the limit is almost fully established after just a couple of coarsening levels).

This rather trivial observation implies that coarsening is at least as rich and difficult as discretization. It also implies that controversies and competing techniques will emerge concerning coarsening techniques similar to the ones in the field of discretization. Indeed such competitions have already surfaced. For example, the competition between finite-difference and finite-element methods, a dispute which in fact consists of several separate issues: The variational derivation of discrete equations (or coarse-grid equations) vs. direct differencing; the interpolation issue (finite-elements insist on using the same coarse-to-fine interpolation - the same "element" - as used in deriving the discrete equations, while finite-differences allow more freedom in interpolation, sometimes gaining higher accuracy in some

error norms); the issue of general triangulation vs. uniform grids; and the issue of compactness of high-order approximations. These issues should not be confused with each other: variational derivation is possible and natural even without the use of elements [FW60, §20.5]. Uniform grids can be used with finite-element solutions, too, changing the elements only near boundaries, a structure offering high computational efficiency, especially in conjunction with multigrid methods [Bra79a]. High-order compact operators arise quite naturally in the finite-element method, but such operators can also be derived by finite-difference approaches, such as the operator compact implicit method [CLW78] and also the Hodie method [LR78].

All these and other issues arise as well with regard to coarsening, and the competing approaches are generally successful in coarsening wherever they are successful as discretization procedures – which is usually in problems.where they are more natural. Variational approaches [Nic75], [Bra77a, App. A.5], [Hac82] are natural for self-adjoint problems, and have provided the most robust and automatic coarsening procedures for such problems [ABDP81], [Den82a], although they can be replaced by much less expensive procedures (analogous to direct differencing) if the self-adjoint problem is not particularly complicated (cf. §4.5–4.6). In singular perturbation problems, such as those arising in fluid dynamics, discretization as well as coarsening are most successfully guided by physical understanding (artificial viscosity, upstream differencing, etc.); and so on.

The attempt to devise general fine-to-coarse transfers, good for all problems, is as hopeless (or as hopeful) as the attempt to have general, completely problem-independent discretization procedures.

Notwithstanding, while this argument tells us how complicated coarsening can be, it also elucidates a general way to handle this difficulty. Namely, the coarsening method can always be guided by the discretization scheme.

Indeed, conversely to the statement above (that every coarsening method implies a discretization scheme), one can say that every discretization scheme can be used to derive a coarsening procedure. This is done by imitation or analogy: Think about *discretizing* the problem $Lv = r$ on the coarse grid H; then replace the operations done on the continuous domain by analogous operations done on the fine-grid h; e.g., replace integrations by analogous summations (or by integrations by elements, in case v^h is given in terms of finite elements). Galerkin discretization schemes, for example, are easily translated in this way into analogous coarsening formulae of the type (4.11) [Nic75, §3].

A coarsening procedure analogous to the finest-grid discretization scheme is called **compatible coarsening**. It is not necessary to use compatible coarsening, but it usually makes a good sense to do so. In case the discretization scheme is a bad one, this would give a bad coarsening and hence slow asymptotic convergence rates of the multigrid cycling. But experience with several such cases (e.g., boundary singularities improperly

treated) show that, if compatible coarsening is used, this slowness does not matter, because the source of slowness (bad discretization) is also, and for the same solution components, a source for large truncation errors, hence an FMG algorithm (with the same discretization scheme on all currently-finest levels) still solves *below truncation errors* in the usual number of cycles (one or two, depending on the interior processes). Moreover, the slower *asymptotic* algebraic convergence rates can in this way serve as a detector for the bad discretization, which otherwise may be passed unnoticed.

Compatible coarsening makes sense also from the point of view of computer resources and programming effort. For example, if a great generality and simplicity of programming is obtained by a discretization scheme (e.g., finite elements) which on the other hand spends a lot of computer time and storage to assemble the discrete equations and store them, the coarsening procedure can do the same since the time and storage it spends will be smaller than those already spent on discretizing on the fine grid.

There are some special cases in which compatible discretizations are not quite available. These are cases where the discretization scheme is not general enough, because it specifically uses features of the finest grid not present on coarser ones. It uses for example a finest grid exactly laid so that its lines coincide with special lines of the problem, such as boundaries or lines of strong discontinuities (as in [ABDP81]). In such situations compatible discretization is not well defined. To define it we must think in terms of a more general discretization scheme. (Again, the coarsening process serves to detect a certain flaw in discretization: In this case the flaw is the lack of generality.)

When double discretization is used (§10.2), compatible coarsening means the use of such a double discretization on coarser levels, too (as indeed recommended in §10.2).

Chapter 12

True Role of Relaxation

The role of relaxation in multigrid processes has often been stated: It is to *smooth the error*; i.e., to reduce that part of the error (the "high-frequency" part) which cannot be well approximated on the next coarser grid. Some elaboration and clarification of this statement is important.

What is the "error" we want to smooth? It is usually thought of as the *algebraic* error, i.e., the difference $u^h - \tilde{u}^h$ between our calculated solution $\tilde{u}^h$ and the discrete solution u^h (the exact solution to the discrete equations). However, in view of the double discretization scheme (Sec §10.2), where u^h is not well-defined, it becomes clear that what relaxation should really do is to smooth the *differential* error, i.e., the difference $u - \tilde{u}^h$, where u is the solution to the given differential equations. In fact, this is the true role of relaxation even when double discretization is not used, if what we want to approximate is u, not u^h: It is the smoothness of $u - \tilde{u}^h$ which permits its efficient reduction via the coarser grids.

Thus; the important measure of relaxation efficiency is not the algebraic smoothing factor $\overline{\mu}$, but the differential smoothing factor, the factor by which the high-frequency part of $u - \tilde{u}^h$ is reduced per sweep. This is not usually recognized because the latter factor is not constant: It approximately equals $\overline{\mu}$ when the high-frequencies in $u - \tilde{u}^h$ are large compared with those in $u - u^h$ (where u^h is the local solution to the discrete equations employed in relaxation), but below this level $\overline{\mu}$ may mislead, and when $\tilde{u}^h$ is closer to u than to u^h in their non-smooth components, the differential factor may even be larger than 1. For example, in solving a singular perturbation problem with strong alignment (see §2.1), we can reduce the algebraic smoothing factors of point Gauss-Seidel relaxation by taking a larger artificial viscosity and, more importantly, by taking it *isotropically* instead of anisotropically. This would not however improve the overall performance of our double-discretization FMG algorithm (see the experiments

in [Bra81a, §7]), since it would not reduce the *differential* smoothing factors.

The differential smoothing is the purpose of relaxation not only on the finest grid h_*. On any grid h, its relaxation should reduce the scale-h high-frequency components of the error $u - \tilde{u}^{h_*}$ where we interpret changes in $\tilde{u}^h$ as changes in $\tilde{u}^{h_*}$ via the interpolation relations.

We can here also elaborate on what are those scale-h *"high-frequency components"* (of the differential error) that should be converged by relaxation on grid h. Generally speaking we say that these are the components "invisible" on the next coarser grid H, i.e.., Fourier components $\exp(i\underline{\theta}\cdot\underline{x}/\underline{h})$ which on grid H coincide with lower components, that is to say components with $\pi < \max_j |\theta_j| H_j/h_j$, $max_j|\theta_j| \leq \pi$ (cf. (3.3)).

More precisely, we should include in the "high-frequency" range all those components that are not efficiently reduced by relaxation on the other grids, which can for example be any range of the form

$$\left\{(\theta_1, \ldots, \theta_d) : \exists j \text{ s.t. } \frac{\alpha_j h_j}{H_j} \leq \theta_j \leq \alpha_j\right\}, \tag{12.1}$$

where each $0 < \alpha_j \leq \pi$ is fixed (assuming H_j/h_j is the same for all levels). In other words, we can allow some of the highest frequency components on any intermediate level not to converge efficiently by relaxation ($\alpha_j < \pi$), as long as those components efficiently converge by the next-finer-level relaxation. This may leave the highest frequencies on the finest grid uncontrolled, but they are unimportant and can be eliminated by averaging the final results. Examples where this further understanding of mode analysis is relevant are mentioned in §18.6 and in [Bra81a, §5.7].

The range of frequencies to be reduced by relaxation may also change by modified coarse-grid functions of the type mentioned in §4.2.2. In such cases relaxation may not reduce some high-frequency error components; but the unreduced components are very special ones, hence they are described by few parameters. This is a general property of relaxation (see §1.1). Very generally we can thus say that the role of relaxation is to reduce the *information content* of the error, so that it becomes approximable by a lower dimensional approximation space.

Another important point to clarify is that relaxation should be efficient only as long as the high-frequency error components have relatively large amplitudes: When the high-frequency errors are too small compared with the low-frequency ones, relaxation cannot usually be efficient because of certain *feeding from low to high components*. Such feeding is caused by interaction with boundaries, and by non-constant coefficients, and by the high-frequency harmonics generated when the low-frequency error is corrected via the coarse-grid cycle (see observation (D) in §4.3). Sometimes such feeding is even caused by the interior relaxation itself; e.g., red-black relaxation of an order-m differential equation produces $O(h^m)$ high-frequency errors from $O(1)$ low-frequency errors. When the size of

high-frequency amplitudes approaches the size fed from low frequencies, relaxation should be stopped; this is the point where the coarse-grid correction should be made. If relaxation is stopped in time, then the range of strong interactions with low-frequencies is not entered. It is also only then that the multigrid convergence rates can accurately be deduced from the smoothing-rate analyses.

Exchange-rate criteria for stopping relaxation in time are described in §9.6. In most cases, though, the practical approach is to stop relaxation as soon as its amount of computations becomes substantially larger than the amount invested in the coarse-grid correction, which simply means to limit the number of relaxation sweeps per cycle to be less than 3 or 4.

Finally, even though smoothing is the main role of relaxation, we should not forget its influence on other components. Some relaxation schemes with extremely good smoothing factors are either unstable or they cause large amplification of some low-frequency errors (see §3.2).

We can thus say in summary that *the role of relaxation is to reduce large amplitudes of certain components of the differential error (those components not efficiently reduced by relaxation at other levels), while avoiding from significantly amplifying its other components.*

Stability of the difference equations used in relaxation is only a tool in performing this role, not an end by itself.

Chapter 13

Dealgebraization of Multigrid

An interesting line in the development of multigrid can be viewed as a gradual "dealgebraization", a gradual liberation from algebraic concepts, and the development of methods that increasingly exploit the underlying *differential* nature of the problems. We'd like to briefly trace this line here, so as to bring out some concepts useful in practical implementations.

As the first step of dealgebraization we can regard the **replacement of "acceleration" by "smoothing"**. The early two-grid and multi-grid approach viewed coarse-grid corrections mainly as a tool for accelerating the basic iterative process - the fine-grid relaxation. Only later it became clear that the only role of relaxation is to smooth the error. (Cf. §12, where a further "dealgebraization" of the smoothing concept is described.) This slight shift in understanding revolutionized the multigrid practice: It made it clear that the fine-grid process is basically local, hence analyzable by local mode analysis. This understanding, together with that analysis, produced the truly efficient multigrid cycles, in which very few sweeps are made on each grid before switching to coarser ones, and in which the fine-to-coarse meshsize ratio assumes the (practically) optimal value of $1:2$.

The next dealgebraization steps are related to the trivial understanding that we are not primarily interested in solving the algebraic equations (obtaining u^h), but we are interested in approximating the differential solution u. First, this implies that we have to solve the algebraic equations only "to the level of truncation errors", i.e., only to the point that our calculated solution $\tilde{u}^h$ satisfies $\|\tilde{u}^h - u^h\| \approx \|u - u^h\|$; further reduction of $\tilde{u}^h - u^h$ is futile.

This implies that the asymptotic convergence rate of the multigrid cycle is not important by itself. What counts is the amount of work we need in an FMG algorithm in order to reduce the error from its original value on grid $2h$, which is approximately $\|I_{2h}^h u^{2h} - u\| \approx \|u^{2h} - u\|$, to the

desired level $\|u^h - u\|$. This is a reduction by a modest factor, which can usually be achieved in one cycle. (The fundamental reason for this is again non-algebraic: See §7.3.) Evidently it is then more relevant to think in terms of the FMG analysis (§7.4) than in terms of asymptotic rates.

Even the later viewpoint, that we want to reduce the errors to the level of truncation errors, is too algebraic-oriented. It is tied too much to one given discretization on one given grid. The optimal moment of switching from a certain currently-finest grid H to a new, finer grid $h = H/2$ is not necessarily when $\|\tilde{u}^H - u^H\| \approx \|u - u^H\|$. Rather, it is determined by comparing H-cycles to h-cycles in their efficiency at driving $\tilde{u}^h$ closer to u (see §7.2). what really counts is the behavior of the differential error $E = \|\tilde{u}^h - u\|$ as a function of the total accumulated computational work W.

We want $E(W)$ to be as fast-decreasing as possible.

From this as our objective we can derive correct switching criteria, i.e. decide when to establish a new finer grid. The next step is to realize that criteria based on $E(W)$ can be applied *locally*, to decide not only *when* to have a finer grid, but also *where* to have it. This naturally brings us to grid-adaptation (§9.5). Indeed one can integrate the switching and self-adaption criteria (discussed in §6.2, 7.2, 9.5) into a total **multi-level adaptive algorithm**, where switching between levels and creating new, or extending existing, levels are all governed by the same exchange-rate criteria (see §9.6).

Another step away from fixed algebraic concepts is to allow **variable discretization schemes**, i.e., schemes which can be changed throughout the algorithm to promote faster decreasing $E(W)$. This includes the use of higher-order, variable-order and adaptable-order schemes, governed again by E(W) criteria (see §9.5 and [Bra79b, §3.6, §4.3]). Using different discretization schemes in relaxation and in residual transfers (§10.2) is a further step in that direction.

By now we have gone quite far beyond the notion of multigrid as just a fast algebraic solver, toward viewing it as a **total treatment of the original problem**. This is proved to be a very beneficial general principle: Always think of multigrid in terms of as original a problem as possible: For example, instead of using Newton iterations, employing multigrid as a fast solver of the linearized problems, apply multigrid directly to the non-linear problem (§8.3). Instead of solving an eigenproblem by the inverse power method, with multigrid as the fast inverter, you can multigrid directly the original eigenvalue problem (§8.3.1). Instead of using multigrid for solving each step in some outer iterative process - be it a continuation process, a time-dependent evolution, a process of optimizing some parameters or solving an inverse problem, etc. - apply it directly to the originally given problem (cf. §8.3.2, 15, 16). Instead of a grid adaptation process where the discrete problem on each grid configuration is completely solved (by multigrid, say) and then used to decide on an improved grid configuration, the

whole adaptation process can be integrated into a multigrid solver (§9.6); and so on.

An illustration to this approach is the solution of **optimization problems**, where the parameter to be optimized is some continuum *function* on which the solution u depends. This "parametric function" may for example be the shape of the boundary (e.g., the shape of an airplane section which we want to optimize in some sense), or a certain coefficient of the differential equations (e.g., in inverse problems, where one tries to determine this coefficient throughout the domain so that the solution will best fit some observational data), etc. Multigridding the original problem means that we solve it by some FMG algorithm, where already at the coarser FMG stages we treat the given *optimization* problem, by optimizing a coarser representation of the parametric function. On the finer grids, incidentally to relaxing the equations, we optimize that function locally (when this makes sense), and then we introduce smooth corrections to the function during the coarse-grid correction cycles. Instead of using the multigrid solver many times, we may end up doing work only modestly larger than just *one* application of that solver.

13.1 Reverse trend: Algebraic multigrid

Contrary to the above line of dealgebraization, there is a recent trend to develop purely Algebraic Multi-Grid (AMG) algorithms. By this we mean a multi-level algorithm without any geometry, without grids. An algebraic (linear or nonlinear) system of equations is given. To solve it fast, a sequence of increasingly "coarser" levels is created. A coarser level in this context is a related, but much smaller, algebraic system. The choice of the coarse-level *variables* and of the coarse-to-fine interpolation I_H^h, is based not on geometric positions but on the algebraic equations themselves: The coarse variables are chosen so that each fine-level variable is strongly coupled to one or more of them, and each I_H^h coefficient can for example be chosen to be proportional to the corresponding coupling strength. The fine-to-coarse transfer and the coarse-level matrix are then constructed by prescriptions like (4.12) and (4.11), respectively. The theoretical background directing the various choices is developed in [Bra86].

Generally, the efficiency that can be achieved by such algebraic algorithms is below that of algorithms built to exploit the geometric information, let alone the further efficiency obtainable by further dealgebraization. On the other hand these algebraic solvers may be used as black boxes for larger classes of problems. They may especially be useful in cases where the geometrical information is too complicated, such as finite-element equations based on arbitrary partitions, or various problems which are not differential in their origin but still lend themselves for fast multi-level solutions. Also, there are cases of finite-difference equations on a uniform grid, in which the usual geometric choice of coarse-grid variables is not good, since too many

finegrid variables happen to depend too weakly on the coarse-grid variables (cf. e.g., [ABDP81, §8]). Algebraic multigrid can then perform better. Because of its sensitive coarsening, there is in AMG no need for special relaxation schemes, in varying block and marching directions (cf. §3.3); simple Gauss-Seidel is for example used for all definite problems. Experiments on a wide range of problems, including discretization of regular and degenerate second-order elliptic equations as well as problems with no continuous origin, show that the typical multigrid convergence rates are robustly obtained [BMR84], [Stü83]. The AMG set-up time is expensive, but still comparable to the set-up time required by any Galerkin coarsening (4.11). Work is underway to generalize AMG to other classes of matrices, such as those arising in discretizing *systems* of differential equations.

Chapter 14

Practical Role of Rigorous Analysis and Quantitative Predictions

14.1 Rigorous qualitative analysis

A good deal of the literature on multigrid consists of articles with *rigorous* analyses of the algebraic convergence. See for example the pioneering papers [Fed64a] and [Bak66b], the classical book [Hac85] and many articles in the proceedings of over thirty Copper Mountain and European conferences on multigrid methods. For a growing class of problems the basic multigrid assertion is rigorously proven, namely, that an FMG algorithm will solve the algebraic system of n equations (n unknowns on the finest grid) to the level of truncation errors in less than Cn computer operations; or at least, that $Cn \log \frac{1}{\varepsilon}$ operations are enough to reduce the l_2 norm of the error by any desired factor ε. The emphasis is on C being independent of n; it may depend on various parameters of the given differential problem. This is clearly the best one can do in terms of the order of dependence on n, hence the result is very satisfying.

The question discussed below is what role such rigorous analyses can have in the *practical* development of multigrid techniques and programs. It is an important question for the practitioner, who may wonder how much of those proofs he should try to understand. The rigorous proofs apply only to relatively simple problems and synthetic algorithms (often different from the best practical algorithms), but their main shortcoming is that they are usually unrealistic in predicting the true size of C. In most proofs C is not even estimated. This does not change the important fact that the best the proof could do in terms of C is very unrealistic: In most cases the provable constant is many orders of magnitude larger than the one obtainable in practice. In some typical cases the rigorous bound is $C \approx 10^8$, while the practical one is $C \approx 10^2$. Only in the very simplest situation (equations with constant coefficients in a rectangle) one can obtain realistic values of C,

by Fourier methods [Fre75], [Bra77a, App. C], [ST82, §8]. Recently, some analyses have been made which obtain reasonable (although still several times larger than the practical) values of C for more general problems [Bra82a],[Ver84a],[Bra86].

What can then be the practical value of the Cn results, especially those where C is unreasonably large? Usually in complexity analyses results with undetermined constants are sought in cases where the size of the constants is indeed less important. A typical result would for example be that a solution to some problem, depending on some parameter n, is obtained in $Cn!$ operations. Here C may be unimportant, since changing C by orders of magnitude will only slightly increase the range of n for which the problem is solvable. But this spirit of undetermined constants is clearly pushed way too far when the estimate is Cn, the typical constant is $C = 10^8$ and the typical value for n is 10^4 to 10^6. Here C becomes more important than n. In the practical range of n, the provable Cn result is then vastly inferior to results obtained by simpler algorithms (such as banded elimination with typically $4n^2$ operations; not to mention drastic improvements obtainable by modern sparse-matrix packages [Duf82]). Thus, the values of n for which the unrealistic rigorous result can compete with much simpler solution methods is very far out in the range of overkilling the problem. In a sense, one proves efficiency of an algebraic solution process by taking an extremely unreasonable algebraic problem.

The usual rigorous theory, being too concerned on making C independent of n, is often careless about its dependence on various problem parameters. This dependence can be hair-raising indeed, something like $\exp(\exp(\cdot))$, with as many compounded exponentials as there are stages in the proof. Hence, a very distorted picture is in fact supplied about the real complexity in solving the given differential problem.

The implied intention of "Cn" theorems with unspecified or unrealistic C is sometimes understood as follows: the rigorous analysis only tells us that a constant C *exists*, its actual value can then be determined empirically. That is, if we have calculated with $n = 10^3$ and solved the problem in 10^5 operations, say, then the rigorous proof guarantees that for n = 10^4 we would solve the problem in 10^6 operations. This understanding is wrong: The nature of the rigorous proofs is such that the information for $n = 10^3$ does not help the estimates for $n = 10^4$. The only rigorous estimate is still $C10^4$ operations, with the same unrealistic C. The guess that the number of operations for $n = 10^4$ will be 10^6 is purely non-rigorous. Even *heuristically* it does not follow in any way from the "Cn" theorem. Nothing in that theorem excludes, even heuristically, an operation count such as $Cn/(1 + 10^4C/n^2)$, for example with an astronomically large C. Thus, if one literally believed these rigorous bounds, one would not use the multigrid method in the first place. This indeed historically happened: The estimates in [Fed64b] are so bad (although only the simplest problem is considered; cf. [Bra77a, §10]), and those of [Bak66a] so much worse (even

though his constants are undetermined), that nobody was encouraged to use such methods. They were considered to be merely of asymptotic curiosity.

Several other cases from the multigrid history are known where wrong practical conclusions were derived from the asymptotic rigorous analysis. For example, non-smooth boundaries, reentrant corners in particular, gave troubles in the rigorous proofs. This led to the wrong conclusion that there are real troubles there. The practical fact is that such problems are solved to within truncation errors as easily as regular problems; even the asymptotic algebraic convergence rates in such cases can be made to attain the interior rates (see §5.7). The difficulties are purely difficulties of the proof, not of the computational process. The proof made us too pessimistic. In other cases similar proofs made people too optimistic, because their asymptotic relations did not show the real difficulties encountered in the real range. Some people did not realize, for example, the very real difficulties in solving degenerate and singular perturbation equations (in particular, indefinite problems such as $\varepsilon\Delta u + k^2 u = f$, where ε is positive but small), because these difficulties disappear for sufficiently small meshsizes. But such meshsizes are far too small to be used in practice. (The terrible growth of C as function of ε is not seen if all we are interested in is that C will not depend on n.) Fedorenko had a completely wrong idea about the *practical* meshsize ratios and the number of grids to be used. He writes: "The proposed method thus consists of a solution with the aid of an auxiliary net; if this latter is extremely large, the problem can also be solved on it by using a net of a particular type for the problem, and so on". Several similar historical examples could be given.

It is indeed not reasonable to expect unrealistic performance estimates to be of practical value. In practice we are interested in understanding the difference between one algorithm which solves the problem in few minutes CPU time and another algorithm which solves it in a few more minutes, or in hours. A rigorous result that tells us that the solution will surely be obtained within a few weeks (even years) of CPU time, cannot explain that difference. *The factors important in the proof may only remotely and non-quantitatively be related to those operating in practice.* Even in cases of much more reasonable C (such as [Bra82a]), the relative values of C in two competing approaches (e.g., V cycles vs. W cycles) does not point to their relative efficiency in practice. The rigorous proof tells us more about the efficiency of the proof than about that of the actual algorithm.

In sum, for all its pure-mathematical interest and intellectual challenge, much of the existing rigorous approach is *not a practical tool.* It has played no significant role in developing the various algorithms and insights described in this book. Its only role has generally been to *enhance our confidence in the method,* a psychological role that should not be slighted.

14.2 Quantitative predictors

On the other hand, it is strongly recommended not to restrict oneself to numerical experiments only, without *any* supporting theory. The experiments can be, and have been, quite misleading: they happen to show, for some particular cases, much better results than should generally be expected. More often, they show results much inferior to those that could be obtained, because of some conceptual mistakes and/or programming bugs. Experience has taught us that **careful incorporation of (usually non-rigorous) theoretical studies is necessary for producing reliable programs that fully realize the potential of the multigrid method.**

The purpose of the analysis should be borne in mind. We are not trying here to prove any central mathematical idea. We are engaged in a very practical problem, namely, how to solve the equations *fast.* This is in its nature as practical a problem as, say, building an airplane or understanding nuclear fission. (In fact the main purpose of the fast solvers is to aid solving such engineering and scientific problems.) One would not postpone building airplanes until rigorous proofs of their flight capabilities are furnished. Clinging to rigorous mathematics, like clinging to any secure images, may have wrong contexts. Moreover, in this business of fast solvers what one tries to a-priori estimate is nothing but the computer time (and other computer resources), which is after all exactly known in each particular case, even though *a-posteriori.* The main practical aims of theoretical understanding should therefore be:

(A) To give us realistic and **quantitative** insights to the important factors affecting the overall efficiency. The insight should be simple enough and still precise enough so that one can use it to improve our algorithms, and perhaps even to debug his programs.

(B) Even more importantly than quantitative performance prediction, one wants to know whether the performance (predicted or found empirically) is as good as one could *hope* to get (see the situation described in §0.2). Hence, the main theoretical task is to provide us with **ideal performance** figures, which the practical algorithm should then *attempt* to approach.

Local Mode Analysis (LMA) is an example of a theory constructed with these aims in mind. This is amply emphasized throughout Part I of the present book. The easiest and most practical LMA predictor is the smoothing factor (§3). A more elaborate predictor is obtained by a similar Fourier analysis of a several-level (most often two-level) multigrid cycle, thus analyzing both the relaxation and inter-grid transfers. (See §4.1 and detailed results and software for calculating such convergence factors in [Wei01]). LMA is also applicable to the FMG algorithm (§7.4), even in non-elliptic cases where the boundary plays an important role (§7.3). Although the em-

ployed Fourier analysis is rigorously valid only for equations with constant coefficients in an infinite or rectangular domains, in practice the predictions hold in a much wider class of problems, so they are routinely used in algorithm development and program debugging, even for complicated nonlinear systems.

Moreover, it was rigorously proved in [Bra91] and [Bra94] that in general uniformly-discretized linear elliptic PDE systems with piecewise smooth coefficients on general domains, the quantitatively sharp convergence factors predicted by LMA are indeed attained in the limit of small meshsizes, provided the multigrid cycle is supplemented with a proper processing at and near the boundaries. That processing was proved to cost negligible extra computer work.

Apart from mode analysis, a Coarse Grid Approximation condition has been introduced in [Bra91] and [Bra94] which is both necessary and sufficient for the multigrid algorithm to work properly. Various error norms and their relations to the orders of the inter-grid transfer operators are analyzed. Global mode analysis, required to supplement the local analysis in various border cases, is developed, and practical implications of the analysis, including practical ways for constructing and debugging multigrid solvers, are generally reviewed. A major emphasis is on the importance and practicality of adding partial (local) relaxation passes to the multigrid algorithm [Bra77a, App. A.9]: both theory and practice demonstrate that multigrid efficiency is greatly enhanced by adding special relaxation steps at any local neighborhood exhibiting unusually large residuals (cf. the adaptive relaxation rule in §5.7).

14.3 Direct numerical performance predictors

LMA played a key role in the initial understanding and the subsequent development of fully efficient multigrid algorithms. Its application has however proved increasingly cumbersome in advancing to variable-coefficient and nonlinear problems, in particular in analyzing unstructured or near-boundary processes. Much attention has therefore been given in recent years to more generally applicable and simpler-to-implement direct numerical tools for performance prediction and program debugging.

14.3.1 Compatible relaxation

Introduced in [Bra00], Compatible Relaxation (CR) is a tool to assess the potential multigrid efficiency of a given combination of a relaxation scheme and a set of coarse variables – prior to the actual construction of the inter-grid and coarse-grid operators. It is a special case of a general approach for constructing coarse-level descriptions of a given fine-level system, including non-stationary, highly nonlinear and also non-deterministic systems [Bra10]. The general coarsening criterion in all these systems states that *a*

set of coarse-level variables is considered adequate if, and to the extent that, a local processing is available to rapidly reconstruct any fine-level solution from its coarse-level values.

In the case of solving a system of local equations, such as discretized PDE, the local processing is *Compatible Relaxation* (CR), defined as a relaxation scheme that keeps the fine-level configuration compatible with the coarse one (i.e., coarsening the fine configuration yields the given coarse-level equations), at least asymptotically. For example, if coarsening is done by injection, i.e., if the coarse-level values consist of a subset of the set of fine-level value, then one kind of compatible relaxation can be a Gauss-Seidel relaxation that avoids relaxing the values of that subset. Another, more generally useful kind of CR is *Habituated Compatible Relaxation* (HCR) [Liv04], in which one alternates between relaxation sweeps and passes that reset (or asymptotically tend to reset) the coarse variables to their given values. Given a fine-level system of equations $L^h u^h = f^h$ and a corresponding relaxation scheme, together with a coarsening rule $u^H = \tilde{I}_h^H u^h$, HCR is applied as follows:

(1) Choose a particular problem that has a known solution, u_0^h, by setting $f^h = L^h u_0^h$. In the case of a linear system, simply choose $f^h = 0$, so the known solution is $u_0^h = 0$.

(2) Calculate the corresponding set of coarse values $u_0^H = \tilde{I}_h^H u_0^h$. The task of the next steps is then to rapidly reconstruct u_0^h by local processing, given f^h and u_0^H.

(3) Choose an arbitrary (e.g., random) initial approximation u^h. In the nonlinear case, u^h should be sufficiently close to u_0^h, but otherwise arbitrary.

(4) Modify u^h so that $\tilde{I}_h^H u^h$ gets much closer to u_0^H. If $\tilde{I}_h^H$ is an injection, simply introduce the values of u_0^H into u^h. If $\tilde{I}_h^H$ is such that each value of u_0^H is some local average (or any other linear combination) of values of u_0^h, make one or two passes of Kaczmarz relaxation (see §1.1) on the system of equations $\tilde{I}_h^H u^h = u_0^H$. (One sweep would suffice in the case of no overlap between these equations. In any case, the iterations should converge very fast. The important advantage here of the Kaczmarz relaxation is that it solves such a vastly-underdetermined system with minimal changes to u^h.)

(5) Relax the equation $L^h u^h = f^h$ by ν sweeps of the given relaxation scheme.

(6) Repeat steps 4 and 5, each repetition representing a cycle. Measure the rate of convergence per cycle of u^h to u_0^h.

Fast convergence (usually first tested with $\nu = 1$) implies that the set of coarse variables $\tilde{I}_h^H$, together with the given relaxation scheme, can produce

an efficient two-grid cycle. This test can therefore be a very effective tool in choosing the coarse set. It is extensively used for that purpose, particularly in constructing AMG solvers for problems on unstructured grids.

Moreover, suitable versions of HCR can be used to accurately predict the convergence rate obtainable for an actual two-grid cycle that includes the same number ν of relaxation sweeps. This in particular is achieved when $\tilde{I}_h^H$ is a Full Weighting operator (the adjoint of an interpolation operator; see §4.4). The prediction is especially accurate when $\tilde{I}_h^H = (I_H^h)^T$, the transposed of the interpolation operator used in the actual cycle, but even for other reasonable choices of $\tilde{I}_h^H$ the predictions are as accurate as those of the two-level LMA – when the latter is at all applicable. HCR offers several advantages over LMA:

(A) *Easy implementation.* Unlike LMA, which requires a substantial separate programming effort, HCR is simple to implement as soon as the relaxation routine has been constructed and a set $\tilde{I}_h^H$ of coarse variables has been proposed.

(B) *Idealized analysis.* Similar to the smoothing-rate LMA predictor (see §3.1), HCR does not really depend on the inter-grid transfers (I_H^h, I_h^H and $\tilde{I}_h^H$) or on the coarse-level operator (L^H). It thus predicts an ideal efficiency that can be attained once these operators are correctly built. HCR can therefore guide the actual construction of these operators by detecting wrong choices and implementation bugs.

(C) *Generality.* Unlike LMA, HCR is directly applicable for complex domains and/or disordered grids and/or disordered *coarse* grids and/or disordered relaxation schemes (including adaptable schemes, e.g., with extra steps near singularities) and/or irregular equations (possibly with strongly discontinuous coefficients) and/or nonlinear problems.

The experience and understanding is that the HCR analysis, with some quite obvious possible modifications, can generally predict *quantitatively well* the ideal efficiency of any normal two-grid cycle, where by "normal" we mean a cycle in which relaxation is used for reducing errors with "large residuals", while the coarse-grid correction is employed for reducing all other errors. For a (linearized) operator L^h, an error v^h is said to have "large residuals" if $\|L^h v^h\|$ is comparable with $\|L^h\| \cdot \|v^h\|$. the norms being the discrete l_2 norms. The HCR predictions should be quantitatively adequate also for the case that the cycle, together with these norms, operate only at some *subdomain*. (An example of cycling which is *not* normal is the one designed for solving non-uniformly elliptic problems, where the accuracy in approximating smooth component propagates a finite additional distance away from the boundary upon each coarse-grid correction. See §7.5.)

Similar predictions can also be extended to other types of cycles; for example, three-grid cycles. Most such predictions are not rigorous. But

their quantitative accuracy in predicting the ideal performance is no less reassuring, and certainly more directly useful, than non-quantitative rigorous analyses.

14.3.2 Other idealized cycles

In a similar spirit, a quite general numerical approach for isolating sources of inefficiency in an existing multigrid program has been developed in [BLE05]. Running that program on a particular case where the solution is known, so that the error function at each stage is also known, the performance of each part of the multigrid cycle is separately evaluated by replacing that part with an "ideal" part and comparing the (asymptotic) behavior of the original cycle with that of the idealized one. For example, as an idealized relaxation one can use an error averaging similar to that produced when relaxing the Poisson equation. Alternatively, one can apply to the error function the operator product $I_H^h I_h^H$, i.e. restriction followed by interpolation. As an idealized coarse-grid correction, one can multiply the error vector by the matrix $I - I_H^h I_h^H$, where I is the identity matrix; and so on. A collection of examples in [BLE05] shows the wide applicability and accuracy of the approach, successfully analyzing cases for which LMA is inapplicable.

Chapter 15

Chains of Problems. Frozen τ

We often need to solve not just one isolated problem but a sequence of similar problems depending on some parameter. For example, we may be studying the effect of changing some physical parameters on the "performance" of a system, where the performance is measured as some functional of the solution u to a differential problem. We may want to find for what physical parameters the performance is optimal. Or, in "inverse problems", we may desire to find the physical parameters for which the solution best fit some experimentally observed behavior. Or we may need to solve a sequence of problems in a continuation process (see §8.3.2). Or, the most familiar case, the parameter may be the time t, and each problem in the sequence may represent the implicit equations of one time step.

The key to a highly efficient multi-level treatment of such a sequence of problems is to understand the behavior of high-frequency components. Most often, the change in one step (i.e., the change from one problem in the sequence to the next) is a global change, governed by global parameters. In some problems, the relative changes in high-frequency components are therefore comparable to the relative changes in low ones. Hence, for such problems, the absolute high-frequency *changes* in each step are negligible – they are small compared to the high frequencies themselves, and therefore small compared with the discretization errors. In such cases one need not use the finer grids at each step; the finer the level the more rarely it should be activated. Often this is the situation in most parts of the domain, while in some particular parts, such as near boundaries, significant high-frequency changes do take place in every step, hence more refinement levels should more often be activated in those parts only.

The Full Approximation Scheme (FAS) gives us a convenient structure in which to see smooth changes in the solution without (locally) activating finer grids. The way to neglect *changes* in wavelengths smaller than $O(h)$,

without neglecting those components themselves, is to freeze τ_h^{2h} (see §8.2), i.e., to use on grid $2h$ the values of τ_h^{2h} calculated in a previous step, thus avoiding any visit to grid h during the present step. Once in several steps of visiting grid $2h$, a visit can be made to grid h, to update τ_h^{2h}. When visiting grid $2h$, changes in τ_{2h}^{4h} are made, and their cumulative values since the last visit to grid h can serve to decide when a new visit to grid h is needed, using exchange-rate criteria (see §9.5–9.6 and [Bra79a, §3.9]). Since these criteria can be applied locally, one can decide when and *where* to activate increasingly finer levels.

An obvious but important remark: Whether the above procedures are used or not, and whether FAS or CS is employed, in each step (i.e., for each problem in the chain) it is normally more economic to *work on the correction problem*, i.e., taking the previous-step solution as a first approximation. When FAS-FMG is used, this is easily done, even for nonlinear problems, as follows. First, the old values of τ_h^{2h} should be used in the $2h$-stage of the FMG algorithm (i.e., before grid h is ever visited in the present step). Secondly, the FMG interpolation (first interpolation to grid h in this step) should be a FAS-like interpolation, using the old values of $\tilde{u}^h$; i.e., like (8.6), but with possibly higher order $\mathbb{I}_H^h$ replacing I_H^h (cf. §7.1).

Solving the chain of problems we usually need to monitor certain **solution functionals**. In order to calculate such a functional Φ with finest-grid accuracy even at steps not visiting the finest grid, transfers as in (8.15) can be used.

Chapter 16

Time Dependent Problems

Several multigrid applications to evolution problems can be mentioned, including fast solvers to implicit equations, coarse-grid time steps, highly adaptable structures, high-order techniques and global conservation facilities.

One obvious application is to use fast multigrid solvers for solving the set of algebraic equations arising at each time step when implicit time differencing is employed. Such differencing is normally needed whenever a physical signal speed is considerably higher than the propagation speed of substantial changes in the solution. The latter speed determines the size of time steps δt we need for approximating the solution accurately, but with such δt and explicit differencing the numerical signal speed will be slower than the physical one, causing numerical instability. Using implicit equations and solving them by multigrid can be viewed as a way to inexpensively obtain high signal speeds by propagating information on coarse grids. Indeed, with a multigrid solver working on the correction problem (see §15), the cost of an implicit time step is comparable to that of an explicit one [BG91].

In many cases one can even do much better using techniques as in §15 above. For second-order parabolic problems, for example, significant changes in high-frequency components, whose wavelength is $O(h)$, occur only in very particular places such as

(A) Initially, for a short time interval, $0 \le t \le O(h^2)$;

(B) At distance $O(h)$ and time interval $O(h^2)$ from points where significant *changes* occur in boundary conditions or in forcing terms (source terms) of the equation.

At all other places significant high-frequency changes are induced by comparably significant low-frequency changes. Hence the frozen-τ technique,

with a special control for time-dependent problems [Bra79a, §3.9], can give us a solution with the fine-grid accuracy but where most of the time in most of the domain we use coarse grids only. The cost of an average step may then be far smaller than the cost of an explicit time-step.

For the heat equation in the infinite space and steady forcing terms, for example, one can show by Fourier analysis that marching from initial state to 90% steady-state, following the solution throughout with close to finest-grid accuracy, can in this way cost computational work equivalent to just 10 explicit time steps! The finest grid needs to be activated only in the first few time steps, and very rarely later [Gre92].

Notice that when we march (calculating smooth changes in the solution) on coarse grids we can also use large time steps with *explicit* differencing. When fully adapted grids are used there is no need for implicit differencing, because each range of components is in effect handled by a meshsize comparable to the wavelength and by a time-step corresponding to the propagation speed, so that no conflict arises between different characteristic speeds.

For problems with small parabolicity (e.g., parabolic singular perturbation to a "reduced" hyperbolic system), the above, technique can be superposed on an integrator of the reduced system (which may itself be based on a method of characteristics).

The multi-level techniques can also be applied to a parabolic *part* of the system, such as the implicit pressure equation in integrating Navier-Stokes equations [Bro82]. Here too, the techniques of §15 can further save a lot of fine-grid processing.

Whether the fast solver is used or not, the multi-level procedures can also give **highly flexible discretization structures**. Patches of finer grids with correspondingly finer time steps can be used in any part of the space-time domain, in a manner similar to §9.1. Anisotropic refinements, local coordinate transformations and rotated cartesian grids can be used as in §9.2, 9.3 and 9.4, all controlled by exchange-rate criteria (cf. 9.5, 9.6); but instead of criteria based on the τ_h^H of the approximate solution, criteria here will be based on recently accumulated changes in τ_h^H [Bra79a, §3.9].

In some problems, especially when integrating over long time periods, certain quantities, such as total mass, must strictly be conserved, otherwise the physics of the system would fundamentally change. Imposing such **global constraints**, with the corresponding freeing of some accuracy parameters in the difference equations, can easily be incorporated when a multigrid solver is used at each time step (see §5.7).

Finally, independently of the above techniques, one could use a multigrid procedure similar to §10.2 above to efficiently increase the approximation order of a stable discretization

$$L_0^h u_0^h(x,t) = 0, \qquad (t > 0) \tag{16.1}$$

of a time-dependent system, not necessarily linear, by using **coarse-grid**

defect corrections. Typically L_0^h is a simple low-order implicit operator, allowing simple integration. One wants to use a simple (e.g., central in time) higher-order operator L_1^h, which may be unstable, to raise the approximation order. This can be done by integrating the defect equation

$$L_0^H v^H = -I_h^H L_1^h u_0^h, \tag{16.2}$$

and then correcting

$$u_1^h = u_0^h + I_H^h v^H, \tag{16.3}$$

where H may either be coarser than h (coarse-grid defect correction) or $H = h$ (defect correction on the same grid). If the order of consistency of L_j^h is p_j, $(j = 0, 1)$, then for low-frequency components

$$|u_1^h - u| = O(h^{p_1} t + h^{p_0} H^{p_0} t^2), \tag{16.4}$$

where u is the differential solution and t is the time interval over which (16.1)–(16.3) is integrated from initial conditions $u_1^h(x,0) = u_0^h(x,0) = u(x,0)$.

The scheme (16.1)–(16.3) is always stable, since only the stable operators L_0^h and L_0^H are integrated. Notice that this is true only if u_0^h is integrated independently of u_1^h. The seemingly similar scheme, in which after each time step u_0^h is re-initialized by being replaced by the more accurate u_1^h, may well he unstable. On the other hand it does pay to re-initialize every $O(1)$ time interval, short enough to make the second term in (16.4) smaller than the first one.

The independent integration of u_0^h and v^h requires extra storage. By taking $H = 2h$ this extra storage becomes only a fraction of the basic storage (one time level of u_0^h), and the computational work is also just a fraction more than the work of integrating (16.1). These two-level schemes can be extended to more levels and more approximation orders. As in §10.2, such multigrid algorithms exploit the fact that the higher-order approximation L_1^h is desired only for sufficiently low frequencies; for the highest frequencies (where numerical instability occurs) L_0^h is in fact a better approximation than L_1^h.

Part III

Applications to Fluid Dynamics

Starting with [Bra73] and [SB77], an ever-increasing number of works have applied multigrid techniques to solve *steady-state* flow problems. Early investigations include works on transonic potential flows, such as [Arl78] , [Boe82], [BK83], [DH82], [MS83] , [Jam79], [SB77], [SZH83], [Bro82], [Cau83], [MR82] , [SC82] and [vdWvdVM83], the latter five treating 3-dimensional flows; works on the Stokes and incompressible Navier-Stokes equations, like [Bra73], [Bra80a], [BD79], [Din79], [Fuc82], [TF81], [Ver83] , [Ver84b], [WS80], [Wes77]; works on the Euler equations [Bra82c] , [Jam83], [Jes83], [Ni82] and on the compressible Navier-Stokes equations [Bra82c]. A survey of all this is not attempted here. Our purpose here is to trace a line of development which gradually leads from very simple equations to the most complicated ones, adding the difficulties step by step, but always maintaining the full multigrid efficiency; i.e., insisting on *solving every problem to $O(h^2)$ accuracy in just few work units*, where the work unit is the minimal amount of computer operations needed to express a discretization of the problem on a grid with meshsize h. and where the operations used can be fully parallelized (or vectorized) over the entire grid. *Minimal computer storage* is also maintained, i.e., a storage just a fraction more than needed to store the solution itself on grid h. Moreover, to show how these goals are achieved for the more complicated systems of equations, our emphasis here is on the treatment of *systems* of differential equations, although the line of development starts of course with simple scalar equations. In particular, the work on the scalar convection-diffusion problem [Bra81a] is a crucial step in that line, as will become clear in §19.3, not to mention the extensive work on the Poisson equation and on more general diffusion problems.

Most works mentioned above lag far behind the ideal performance, for various reasons (see discussion in §1.7). To achieve the goals stated above, many of the principles delineated in the previous two parts of this Guide are, and perhaps must be, used. Other principles described above have not yet been used, but they are available, ready to be added and enhance the power of the flow solvers presented in this part. This includes: methods of flexible local refinements and local coordinate curving (see §9); higher-order techniques (§10: the double discretization scheme of §10.2 is already used to obtain the mentioned O(h2) approximation in cases of inviscid or small-viscosity problems, but still higher orders are obtainable, if desired, for small extra work); procedures for further reducing computer storage (§8.7); the general approach of multigridding directly the real "outer" problem (e.g., the optimization or design problem for which the flow equations are solved; cf. §13); and the methods for efficiently treating sequences of many boundary-value problems and solving time-dependent problems (§15 and 16).

The work described in this part has been done in collaboration with Nathan Dinar and Ruth Golubev. Much of it has appeared before in [BD79] and [Bra82c].

Chapter 17

Cauchy-Riemann Equations

17.1 The differential problem

As a first simple exercise in multigridding a *system* of partial differential equations we have studied the elliptic system

$$u_x + v_y = F_1 \tag{17.1a}$$

$$u_y - v_x = F_2 \tag{17.1b}$$

in a domain Ω, where $u = u(x, y)$ and $v = v(x, y)$ are the unknown functions, the subscripts denote partial derivatives, and $F_i = F_i(x, y)$ are given functions. All functions are real. The homogeneous system $Fl \equiv F2 \equiv 0$ are the usual Cauchy-Riemann equations, which express analyticity of the complex function $u + iv$.

The matrix-operator form of (17.1) is

$$L \begin{pmatrix} u \\ v \end{pmatrix} := \begin{pmatrix} \partial_x & \partial_y \\ \partial_y & -\partial_x \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} F_1 \\ F_2 \end{pmatrix}, \tag{17.2}$$

where ∂_x and ∂_y are partial derivatives with respect to x and y, respectively. The determinant of L is the Laplace operator $-\Delta = -\partial_x^2 - \partial_y^2$. Hence (17.2) or (17.1) is a second-order elliptic system and its solution is determined by one condition along the boundary $\partial\Omega$. As such a boundary condition we can, for example, require

$$(u(x, y), v(x, y))_n = G(x, y), \qquad ((x, y) \in \partial\Omega) \tag{17.3}$$

where $(u, v)_n$ denotes the component of the vector (u, v) normal to the boundary in the outward direction. From (17.1a), (17.3) and the divergence theorem (or Stokes, or Gauss, formula) we get the "compatibility condition"

$$\int_\Omega F_1 dx dy = \int_{\partial\Omega} G ds. \tag{17.4}$$

If (17.4) holds then equations (17.1) or (17.2), with the boundary condition (17.3), is a well-posed problem: A unique solution exists and depends continuously on the data F_l, F_2 and G.

17.2 Discrete Cauchy-Riemann equations

Suppose we first try to approximate (17.1) by the central difference equations

$$\frac{u^h(x+h,y)-u^h(x-h,y)}{2h}+\frac{v^h(x,y+h)-v^h(x,y-h)}{2h}=F_1^h(x,y) \quad (17.5a)$$

$$\frac{u^h(x,y+h)-u^h(x,y-h)}{2h}-\frac{v^h(x+h,y)-v^h(x-h,y)}{2h}=F_2^h(x,y). \quad (17.5b)$$

The corresponding difference operator is

$$L^h=\begin{pmatrix} \mu_x^h\partial_x^h & \mu_y^h\partial_y^h \\ \mu_y^h\partial_y^h & -\mu_x^h\partial_x^h \end{pmatrix} \quad (17.6)$$

where the averaging and differencing operators are defined by

$$\begin{aligned} \mu_x^h\Phi(x,y) &:= \tfrac{1}{2}\left[\Phi(x+\tfrac{h}{2},y)+\Phi(x-\tfrac{h}{2},y)\right], \\ \mu_y^h\Phi(x,y) &:= \tfrac{1}{2}\left[\Phi(x,y+\tfrac{h}{2})+\Phi(x,y-\tfrac{h}{2})\right], \\ \partial_x^h\Phi(x,y) &:= \tfrac{1}{h}\left[\Phi(x+\tfrac{h}{2},y)-\Phi(x-\tfrac{h}{2},y)\right], \\ \partial_y^h\Phi(x,y) &:= \tfrac{1}{h}\left[\Phi(x,y+\tfrac{h}{2})-\Phi(x,y-\tfrac{h}{2})\right], \end{aligned} \quad (17.7)$$

hence

$$\mu_x^h\partial_x^h=\partial_x^{2h}, \qquad \mu_y^h\partial_y^h=\partial_y^{2h}$$

and

$$\det(L^h)=-(\mu_x^h\partial_x^h)^2-(\mu_y^h\partial_y^h)^2=-\Delta^{2h}$$

with the symbol (see §2.1)

$$\tilde{\det}\,L^h(\theta_1,\theta_2)=\frac{\sin^2(\theta_1)+\sin^2(\theta_2)}{h^2}. \quad (17.8)$$

This operator is not h-elliptic, since $\tilde{L}^h(\pi,0)=\tilde{L}^h(0,\pi)=\tilde{L}^h(\pi,\pi)=0$.

Indeed, the homogeneous (17.5) equations ($F_1^h\equiv F_2^h\equiv 0$) have the oscillatory solutions

$$u^h(\alpha h,\beta h)=C_0+C_1(-1)^\alpha+C_2(-1)^\beta+C_3(-1)^{\alpha+\beta} \quad (17.9a)$$

$$v^h(\alpha h,\beta h)=C_4+C_5(-1)^\alpha+C_6(-1)^\beta+C_7(-1)^{\alpha+\beta} \quad (17.9b)$$

which corresponds to nothing similar in the solution of the differential equation. Note, however, that solutions like (17.9) vanish *on the average*, i.e.,

$M^h u^h = M^h v^h = 0$ for a suitable local averaging operator M^h, such as $M^h = \mu_x^h \mu_y^h$ or $M^h = (\mu_x^h \mu_y^h)^2$. Hence, the solutions of (17.5) will be good solutions on the average. Such difference operators are called *quasi-elliptic* [BD79, §3.4]. See further remarks in §17.6.

Let us now construct an h-elliptic approximation L^h to (17.1). If the equations are to have the form

$$D_x^1 u^h + D_y^2 v^h = F_1^h \tag{17.10a}$$

$$D_y^3 u^h - D_x^4 v^h = F_2^h \tag{17.10b}$$

where D_x^j and D_y^j are some difference approximations to ∂_x and ∂_y, then $\det(L^h) = -D_x^1 D_x^4 - D_y^2 D_y^3$ should be an elliptic approximation to the Laplace operator $-\Delta$. The simplest such operator is the five-point operator which is obtained by taking either

$$D_x^1 = D_x^4 = \partial_x^h, \qquad D_y^2 = D_y^3 = \partial_y^h, \tag{17.11}$$

or one-sided differences such as

$$D_x^1 = \partial_x^{\mathrm{F}}, \quad D_y^2 = \partial_y^{\mathrm{F}}, \quad D_y^3 = \partial_y^{\mathrm{B}}, \quad D_x^4 = \partial_x^{\mathrm{B}} \tag{17.12}$$

where $\partial_x^{\mathrm{F}} := \mu^h \partial^h - \frac{h}{2} \partial^h \partial^h$ and $\partial_x^{\mathrm{B}} := \mu^h \partial^h + \frac{h}{2} \partial^h \partial^h$. Approximations like (17.12) do not give a central approximation to (17.1), and their truncation error is therefore $O(h)$. We thus prefer to use (17.11). This we can do only by using staggered grids for u^h and v^h. The grid we use and the positioning of the discrete variables are shown in Fig. 17.1.

With this positioning we can indeed approximate (17.1) by

$$\partial_x^h u^h + \partial_y^h v^h = F_1^h \quad \text{at cell centers } ① \tag{17.13a}$$

$$\partial_y^h u^h - \partial_x^h v^h = F_2^h \quad \text{at interior vertices } ② \tag{17.13b}$$

and the symbol is that of the 5-point Laplacian, namely,

$$\tilde{\det} L^h(\theta_1, \theta_2) = \frac{4}{h^2} \left(\sin^2 \frac{\theta_1}{2} + \sin^2 \frac{\theta_2}{2} \right). \tag{17.14}$$

This symbol vanishes only for $\theta_1 \equiv \theta_2 \equiv 0 (\mathrm{mod} 2\pi)$. Thus (17.13) is an elliptic (even R-elliptic – see [BD79, §3.6]) difference system.

For simplicity we consider here domains with boundaries along grid lines. It is then simple to discretize the boundary condition (17.3). On each boundary link (the heavy lines in Fig. 17.1) the variable (u, v) is already defined at the center of the link, so (17.3) is discretized to

$$(u^h, v^h)_n = G^h \text{ at midpoints of boundary links.} \tag{17.15}$$

Summing (17.13a) over all the cells of our domain we get the compatibility condition

$$\sum_{\text{cell centers}} F_1^h(x, y) = \sum_{\text{boundary midpoints}} G^h(x, y) \tag{17.16}$$

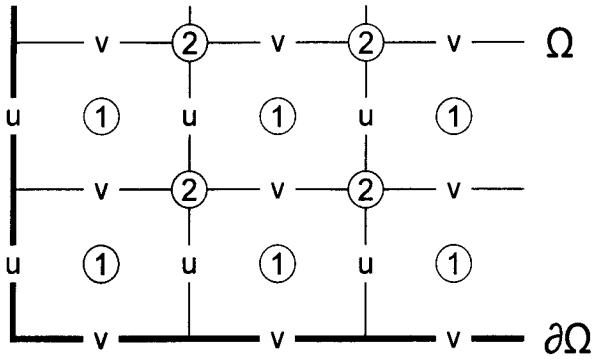


Figure 17.1. ***Discretization of Cauchy-Riemann equations.*** *A typical part of the grid is shown. The discrete unknown functions u^h and v^h and their computed approximations $\tilde{u}^h$ and $\tilde{v}^h$ (u and v in the figure) are defined at the centers of vertical and horizontal links, respectively. The first equation (17.13a) is centered at cell centers, where its right-hand side, F_1^h, is defined, and where ① is shown in the figure. The second equation, (17.13b), is centered, and F_h^2 is defined, at the grid vertices, as shown by ② in the figure.*

which is the discrete analog of (17.4).

Theorem 17.1. *If (17.16) holds, then the discrete Cauchy-Riemann equations (17.13) with the boundary conditions (17.15) have a unique solution.*

Indeed, the total number of equations (17.13), (17.15) equals the total number of cells and vertices in the grid. The number of discrete unknowns is the number of links. Hence, by a well-known formula of Euler, there is one more equation than unknowns. But the equations are dependent, as we saw in constructing the compatibility condition (17.16). Hence, if (17.16) holds, we can remove an equation and have the same number of equations as unknowns. It is therefore enough to prove the theorem for the homogeneous case $F_1^h \equiv 0$, $F_2^h \equiv 0$, $G^h \equiv 0$. In this case (17.13a) implies the existence of a discrete "stream function" ψ^h, defined at the vertices of the grid, such that $u^h = \partial_y^h \psi^h$, $v^h = -\partial_x^h \psi^h$. The homogeneous (17.13b) yields $\Delta^h \psi^h \equiv 0$, and the homogeneous x (17.15) implies that ψ^h along the boundary vertices is constant. Hence, by the maximum principle, ψ^h is constant everywhere. Thus, in the homogeneous case $u^h \equiv 0$ and $v^h \equiv 0$, which is what we had to show.

17.3 DGS relaxation and its smoothing rate

Most relaxation schemes are based on one-to-one correspondence between equations and unknowns: The basic relaxation step is to satisfy (or over-satisfy, or under-satisfy) one of the discrete equations by changing the corresponding unknown (or satisfy a block of equations by changing the corresponding block of unknowns). Such one-to-one correspondence is not always natural. In our case, it is clear already in the differential equations (17.1) that it would be unnatural to regard (17.1a), say, as the equation corresponding to the unknown u, and (17.1b) as the one corresponding to v. The entire system corresponds to (u, v). In the difference equations it would be impossible to have even a one-to-one correspondence between pairs of equations and pairs of unknowns, since the number of unknowns is one less than the number of equations.

We will therefore use "distributive relaxation", i.e., a relaxation scheme that satisfies each discrete equation in its turn by distributing changes to *several* unknowns, in a natural manner.

To derive a natural distributive scheme we note that neither (17.13a) nor (17.13b) are elliptic equations by themselves. It is their combination together which is elliptic. Hence, in relaxing (17.13a), for example, we should take (17.13b) into account. The simplest way to do it, which is also a special case of the general prescription described in §3.7, is to relax (17.13a) in such a way that equations (17.13b) are not "damaged", i.e., in a way which preserves the residuals of (17.13b). We do this by simultaneously changing four unknowns, in the following way:

Let $(\tilde{u}^h, \tilde{v}^h)$ be the current approximation to (u^h, v^h). Let (x, y) be the cell center where we next wish to relax (17.13a), and let

$$r_1^h = F_1^h - \partial_x^h \tilde{u}^h - \partial_y^h \tilde{v}^h \tag{17.17}$$

be the "dynamic residual" at (x, y). That is, r is the residual at (x, y) just before relaxing there. The relaxation step of (17.13a) at (x, y) is made up of the following four changes:

$$\begin{array}{rcl} \tilde{u}^h(x+\frac{h}{2}, y) & \leftarrow & \tilde{u}^h(x+\frac{h}{2}, y) + \delta \\ \tilde{u}^h(x-\frac{h}{2}, y) & \leftarrow & \tilde{u}^h(x-\frac{h}{2}, y) - \delta \\ \tilde{v}^h(x, y+\frac{h}{2}) & \leftarrow & \tilde{v}^h(x, y+\frac{h}{2}) + \delta \\ \tilde{v}^h(x, y-\frac{h}{2}) & \leftarrow & \tilde{v}^h(x, y-\frac{h}{2}) - \delta, \end{array} \tag{17.18}$$

where

$$\delta = \frac{1}{4} h r_1^h. \tag{17.19}$$

It is easy to check that the distribution of changes (17.18) is such that the residuals

$$r_2^h = F_2^h - \partial_y^h \tilde{u}^h + \partial_x^h \tilde{v}^h \tag{17.20}$$

at all neighboring vertices are not changed, whatever the value of δ. The choice (17.19) for the *ghost unknown* δ is made so that *after* the changes

the residual $r_1^h(x,y)$ will vanish. This is in the manner of the Gauss-Seidel relaxation , where old values are replaced by new values so as to satisfy one difference equation. Such schemes are therefore called *Distributive Gauss-Seidel* (DGS) schemes. In case k of the four values changed in (17.18) are boundary values ($k = 1$ near boundaries, except near corners), then no such change should be introduced in those values, and (17.19) is replaced by

$$\delta = \frac{1}{4-k} h r_1^h. \tag{17.21}$$

The relaxation of (17.13b) is made in a similar manner. If (x,y) is the vertex to be relaxed, the relaxation step will include the changes

$$\begin{array}{rcl} \tilde{u}^h(x,y+\frac{h}{2}) & \leftarrow & \tilde{u}^h(x,y+\frac{h}{2}) + \delta \\ \tilde{u}^h(x,y-\frac{h}{2}) & \leftarrow & \tilde{u}^h(x,y-\frac{h}{2}) - \delta \\ \tilde{v}^h(x+\frac{h}{2},y) & \leftarrow & \tilde{v}^h(x+\frac{h}{2},y) - \delta \\ \tilde{v}^h(x-\frac{h}{2},y) & \leftarrow & \tilde{v}^h(x-\frac{h}{2},y) + \delta, \end{array} \tag{17.22}$$

where

$$\delta = \frac{1}{4} h r_2^h. \tag{17.23}$$

The distribution (17.22) is such that the residuals r_1^h will be preserved, and δ in (17.23) is such that equation (17.13b) at (x,y) will be satisfied by the changed variables.

The above relaxation steps can be taken in various orders. In our programs, each complete relaxation sweep comprised of two passes: The first pass relaxes equation (17.13a) by (17.18)–(17.19), letting (x,y) traverse all cell centers, and the second pass similarly scans all the grid vertices, relaxing (17.13b) by (17.22)–(17.23). within each pass the best ordering of points is Red-Black (RB), although lexicographic ordering was used in the experiments [Din79]. Some operations can be saved by passing first on one color of cells and a matching color of vertices, then on the other cell color, and then on the remaining vertices.

In terms of the general formulation of §3.7 the above scheme is equivalent to introducing the ghost functions w_1^h and w_2^h, where

$$\begin{pmatrix} u^h \\ v^h \end{pmatrix} = \begin{pmatrix} \partial_x^h & \partial_y^h \\ \partial_y^h & -\partial_x^h \end{pmatrix} \begin{pmatrix} w_1^h \\ w_2^h \end{pmatrix} = M^h w^h \tag{17.24}$$

(accidentally $M^h = L^h$), and relaxing the resulting system $\Delta^h w_i^h = F_i^h$, $(i = 1,2)$, Δ^h being the 5-point Laplace operator. Hence the *smoothing factor* of this relaxation is the same as that of Δ^h, that is, for lexicographic ordering $\overline{\mu} = .5$, while in RB ordering $\overline{\mu}_1 = \overline{\mu}_2 = .25$ and $\overline{\mu}_3 = .32$ (cf. (3.2)).

17.4 Multigrid procedures

Assume now we have a sequence of grids (levels) with mesh-sizes $h_1, \ldots, h_M$, where $h_{k+1} = \frac{1}{2} h_k$. The relative position of the different grids is shown in

Fig. 17.2. Instead of F_1^h, F_2^h, G^h, u^h, v^h, $\tilde{u}^h$, $\tilde{v}^h$, r_1^h and r_2^h used above, the discrete functions at the *k-th* level will be denoted by F_1^k, F_2^k, G^k, u^k, v^k, $\tilde{u}^k$, $\tilde{v}^k$, r_1^k and r_2^k, respectively. Similarly, μ_x^k and μ_y^k will stand for $\mu_x^{h_k}$ and $\mu_y^{h_k}$.

The coarse-to-fine interpolation can be of first order, since this is the highest order of derivatives in the Cauchy-Riemann operator (see §4.3). An obvious way of doing such an interpolation (see Fig. 17.2) is

$$I_k^{k+1}\tilde{u}^k(x,y) = \begin{cases} \tilde{u}^k(x, y \pm \frac{1}{2}h_{k+1}) & \text{if } x \text{ is on a coarse-grid line} \\ \mu_x^k I_k^{k+1}\tilde{u}^k(x,y) & \text{otherwise} \end{cases} \tag{17.25}$$

and similarly for $I_k^{k+1}\tilde{v}^k(x,y)$. One can of course use linear interpolations instead.

The Cauchy-Riemann problem is linear. We can therefore make coarse-grid corrections either by the Correction Scheme (CS) or the Full Approximation Scheme (FAS, cf. §8). In the latter case we have to define the fine-to-coarse transfer of solution. We use the following averaging (cf. Fig. 17.2):

$$I_{k+1}^k\tilde{u}^{k+1}(x,y) = \mu_y^{k+1}\tilde{u}^{k+1}(x,y) \tag{17.26a}$$

$$I_{k+1}^k\tilde{v}^{k+1}(x,y) = \mu_x^{k+1}\tilde{v}^{k+1}(x,y). \tag{17.26b}$$

The fine-to-coarse transfer of residuals of the first equation (defined at cell

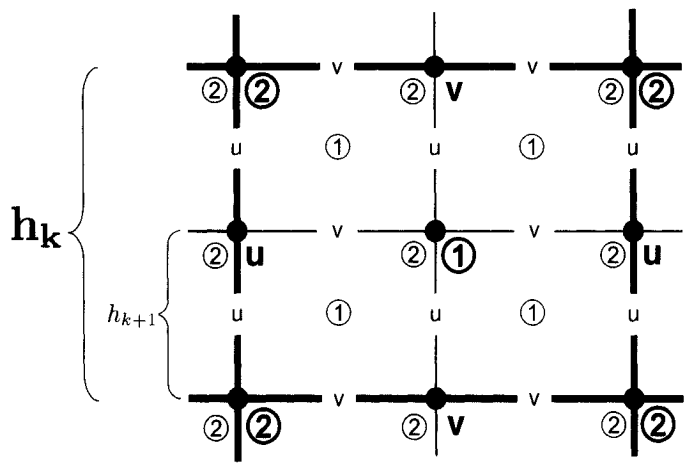


Figure 17.2. ***A coarse-grid cell divided into fine-grid cells.*** *The same notations as in Fig. 17.1, with larger and heavier type being used for the coarse grid and lighter type for the fine grid.*

centers) is also done by averaging:

$$I_{k+1}^k r_1^{k+1} = \mu_x^{k+1}\mu_y^{k+1}r_1^{k+1}. \tag{17.27}$$

When the Correction Scheme is used, (17.27) serves as the right-hand side of equation (17.13a) on the coarser level h_k. In calculating (17.27) using (17.17), observe that some terms are canceled and some of the additions need be made only once for two neighboring coarse-grid cells. It is interesting to note that when FAS is used it is not necessary to calculate (17.27). Transferring $\tilde{u}^{k+1}$ and $\tilde{v}^{k+1}$ by (17.26) and residuals by (17.27), it is easy to see that the FAS coarse-grid equation will read

$$\partial_x^k u^k + \partial_y u^k = \mu_x^{k+1}\mu_y^{k+1}F_1^{k+1}. \tag{17.28}$$

Thus, the coarse-grid equation in this case is not affected at all by the fine-grid solution: If we let $F_1^k = \mu_x^k\mu_y^k F_1^{k+1}$ in the first place, we find that (17.28) is actually identical with (17.13a) for the *k-th* level. In other words, the relative truncation error in (17.13a) vanishes.

Another feature of (17.28) is that if the compatibility condition (17.16) is satisfied on the fine grid, it will automatically be satisfied on the coarse grid too (up to round-off errors, of course). The residuals of (17.13b) can be transferred to the coarse grid either by "injection"

$$I_k^{k+1}r_2^{k+1}(x,y) = r_2^{k+1}(x,y), \tag{17.29}$$

or by the full weighting (4.6).

17.5 Numerical results

Numerical experiments with this algorithm are reported in [Din79]. They show, unsurprisingly, exactly the same convergence as in multigrid solutions for Poisson problems; e.g., a convergence factor of about .55 per RWU (relaxation work unit) when relaxation is performed lexicographically. Hence, although experiments were conducted with a cycling algorithm only, it can be safely predicted that the FMG algorithm (Fig. 1.2), with RB relaxation and $\nu_1 = \nu_2 = 1$, will solve the problem well below truncation errors.

The number of operations in such an algorithm, using the Correction Scheme, is about $40n$, where n is the number of unknowns on the finest grid. Almost all these operations are either additions or shifts (i.e., multiplications by an integer power of 2), and the algorithm is fully parallelizable.

There is a faster way of solving the discrete Cauchy-Riemann equations (17.13): Subtracting from u^h a function u_0^h which satisfies $\partial_x^h u_0^h = F^h$, a new system is obtained in which $F_1^h = O$. The problem can then be rewritten as a Poisson problem for the discrete stream function ψ^h (see §17.2). Solving that Poisson problem by a similar FMG algorithm, together with the operations of subtracting u_0^h and constructing u^h and v^h would require about $17n$ operations (additions and shifts only). The main purpose of this chapter, however, was to study methods for solving elliptic *systems*. The techniques developed for the present simple system guided us in developing the more complicated cases described in the following sections.

17.6 Remark on non-staggered grids

Any staggered-grid formulation can yield a non-staggered one (on a finer grid) simply by overlaying several staggered grids, properly shifted, on top of each other. For example, shifting equations (17.13) by the four shifts $(0,0)$, $(0,h/2)$, $(h/2,0)$ and $(h/2,h/2)$, the four systems together are equivalent to equations (17.5) for grid $h/2$. In fact, various non-staggered formulations appearing in the literature can be shown to be such interlacing of staggered formulations. They are *wasteful* in that the same accuracy is already obtained by just *one* of the interlacing subgrids, for much less work.

Also, the decomposition of the grid into interlacing subgrids locally decoupled from each other introduces *a subtle kind of instability* (typical to quasi-elliptic operators in general): Certain high-frequency modes (those which look like low frequency modes on each subgrid) are magnified in the discrete solution much beyond their size in the differential solution. This may show as large truncation errors in higher Sobolev norms. It can be corrected by averaging. Such an averaging and the multigrid solution of such a quasi-elliptic system are discussed in §18.6.

Chapter 18

Steady-State Stokes Equations

18.1 The differential problem

As a prelude to the treatment of the full Navier-Stokes equations, we consider now the steady-state Stokes equations in a d-dimensional domain

$$\underline{\nabla} \cdot \underline{u} = F_0 \tag{18.1a}$$

$$-\Delta \underline{u} + \underline{\nabla} p = \underline{F}, \tag{18.1b}$$

where $\underline{u} = (u_1, \ldots, u_d)$ represents the velocity of a fluid and p represents the pressure, $\underline{\nabla} = (\partial_1, \ldots, \partial_d)$ is the gradient operator, $\Delta = \partial_1^2 + \cdots + \partial_d^2$ is the Laplace operator, and F_0 and $\underline{F} = (F_1, \ldots, F_d)$ are given forcing functions. (18.1) are the equations of "creeping" flows (vanishing Reynolds number). (18.1a) is the "continuity equation" (usually with vanishing source term: $F_0 = 0$), and (18.1b) is the vector of d momentum equations.

The matrix-operator form of (18.1) is

$$L \begin{pmatrix} p \\ u_1 \\ \vdots \\ u_d \end{pmatrix} := \begin{pmatrix} 0 & \partial_1 & \cdots & \cdots & \partial_d \\ \partial_1 & -\Delta & 0 & \cdots & 0 \\ \vdots & 0 & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 \\ \partial_d & 0 & \cdots & 0 & -\Delta \end{pmatrix} \begin{pmatrix} p \\ u_1 \\ \vdots \\ u_d \end{pmatrix} = \begin{pmatrix} F_0 \\ F_1 \\ \vdots \\ F_d \end{pmatrix} \tag{18.2}$$

and the operator determinant is

$$\det(L) = (-\Delta)^d. \tag{18.3}$$

Hence (18.1) is a $2d$-order elliptic system and will require d boundary conditions. These are usually given by specifying the velocity on the boundary

$$\underline{u}(\underline{x}) = \underline{G}(\underline{x}), \qquad (\underline{x} \in \partial\Omega) \tag{18.4}$$

where $\underline{G} := (G_1, \ldots, G_d)$ and $\underline{x} := (x_1, \ldots, x_d)$.

Equations (18.1) with the boundary conditions (18.4) constitute a well-posed problem, provided the compatibility condition, obtained from (18.1a) and the divergence theorem,

$$\int_\Omega F_0 d\underline{x} = \int_{\partial\Omega} \underline{G} \cdot \underline{d\sigma} \tag{18.5}$$

is satisfied, where $\underline{d\sigma}$ is boundary element multiplying an outward normal unit vector.

18.2 Finite-difference equations

By arguments similar to those in §17.2 and 17.6 we find it best to discretize (18.1) on a staggered grid (but see §18.6). Such a grid, in the two-dimensional case, is shown in Fig. 18.1.

In the general d-dimensional case, the grid hyperplanes (planes if $d = 3$, lines if $d = 2$) define cells, each cell with $2d$ faces. The discrete velocity $u_j^h(\underline{x})$ values are defined at centers of j-faces, i.e., faces perpendicular to the j-*th* coordinate. The discrete pressure p^h and its computed approximation p^h are located at cell centers. The discrete approximation to (18.1) can then be written as

$$\sum_{j=1}^{d} \partial_j^h u_j^h = F_0^h \qquad \text{at cell centers} \tag{18.6a}$$

$$-\Delta^h u_j^h + \partial_j^h p = F_j^h \quad \text{at centers of } j\text{-faces}, \quad (j = 1, \ldots, d) \tag{18.6b}$$

where $\partial_j^h \Phi(\underline{x}) := \frac{1}{h}[\Phi(\underline{x} + \frac{1}{2}\underline{h_j}) - \Phi(\underline{x} - \frac{1}{2}\underline{h_j})]$, $\underline{h_j}$ is h times the unit vector in the direction x_j, and the discrete approximation Δ^h to the Laplace operator is the usual $(2d+1)$-point approximation $\sum_{j=1}^{d} (\partial_j^h)^2$. For a point $\underline{x}$ near a boundary, however, $\Delta^h u_j^h(\underline{x})$ may involve an exterior value $u_j^h(\underline{x}^e)$. This value is defined by quadratic extrapolation from $u_j^h(2\underline{x} - \underline{x}^e)$, $u_j^h(\underline{x})$ and $u_j^h(\underline{x}^b) = G_j^h(\underline{x}^b)$, where $\underline{x}^b$ is a boundary point on the segment $(\underline{x}, \underline{x}^e)$. This definition is used to eliminate the exterior value from $\Delta^h u_j^h(\underline{x})$, so that the discrete Laplacian is modified and includes a boundary value of u_j^h.

The matrix operator of (18.6) is

$$L^h := \begin{pmatrix} 0 & \partial_1^h & \cdots & \cdots & \partial_d^h \\ \partial_1^h & -\Delta^h & 0 & \cdots & 0 \\ \vdots & 0 & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 \\ \partial_d^h & 0 & \cdots & 0 & -\Delta^h \end{pmatrix}, \tag{18.7}$$

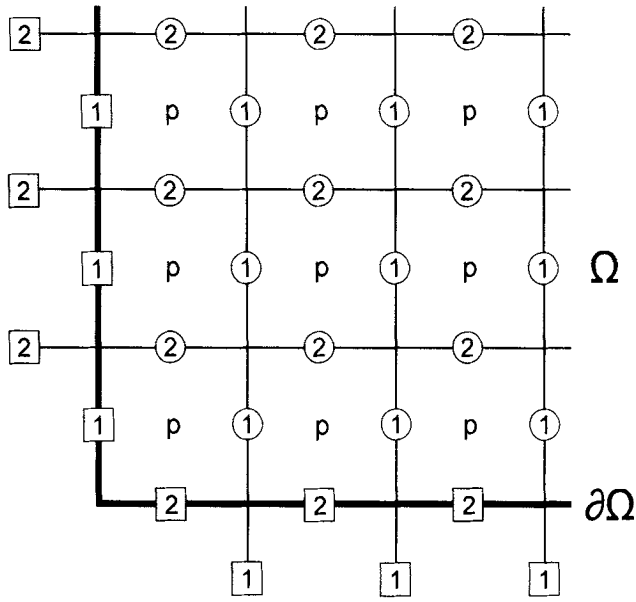


Figure 18.1. ***Discretization of two-dimensional Stokes equations.***

A typical part of the grid is shown. The discrete pressure p^h is defined at cell enters (p). The discrete velocity u_1^h is defined at centers of vertical links (① = interior centers; 1 = boundary and exterior centers), and u_2^h is defined at centers of horizontal links (② and 2). The discrete continuity equations are centered at cell centers (p). The j-th momentum equation is centered at interior values of u_j^h (ⓙ), $j = 1, 2$. The exterior values of u_1^h and u_2^h (at 1 and 2, respectively, but not on the boundary) are fictitious.

hence $\det(L^h) = (-\Delta^h)^d$ and its symbol is

$$\tilde{\det} L^h(\underline{\theta}) = h^{-2d} \left\{ \sum_{j=1}^{d} \left(2 \sin \frac{\theta_j}{2} \right)^2 \right\}^d , \tag{18.8}$$

which is positive for $0 < |\underline{\theta}| \leq \pi$. The difference system (18.6) is therefore h-elliptic (see §2.1) and even R-elliptic [BD79, §3.6].

The boundary condition (18.4) is approximated by the above way for treating boundary and exterior values of u_j^h. For simplicity, consider the case of domains whose boundary is contained in grid lines (or grid planes). In this case the velocity normal to the boundary is conveniently defined at the center of boundary faces, and the discrete analog to (18.5) is naturally

written as

$$\sum_{\underline{x}} F_0^h(\underline{x}) = \sum_{\underline{y}} G_n^h(\underline{y}), \tag{18.9}$$

where $\underline{x}$ runs over all cell centers, $\underline{y}$ runs over all centers of boundary faces, and $G_n^h(\underline{y})$ is the (given) normal velocity at $\underline{y}$.

Theorem 18.1. *The discrete Stokes equations (18.6), with exterior and boundary values determined by the boundary conditions as above, have a unique solution, up to an additive constant in p^h, if and only if (18.9) is satisfied.*

The proof is simple. The number of equations is the same as the number of unknowns, since for each interior $u_j^h(\underline{x})$ there corresponds an equation (18.6b) at $\underline{x}$, and for each unknown $p^h(\underline{y})$ there corresponds an equation (18.6a) at $\underline{y}$. The pressure values p^h are determined only up to an additive constant, but, on the other hand, the equations are dependent; summing (18.6a) over all cell centers we get (18.9). That is to say, if (18.9) is not satisfied we get a contradiction. If (18.9) *is* satisfied, it is enough to show that in the homogeneous case ($F_0^h \equiv 0$, $\underline{F}^h \equiv 0$, $\underline{G}^h \equiv 0$), the only solution is the trivial one ($\underline{u}^h \equiv 0$, $p^h \equiv$ constant). Indeed, if $\underline{F}^h \equiv 0$, it is easy to see from (18.6b) that

$$\begin{aligned} 0 &= \textstyle\sum_{j=1}^d \sum_1^j \left[-\Delta^h u_j^h(\underline{x}) + \partial_j^h p^h(\underline{x})\right] u_j^h(\underline{x}) \\ &= \textstyle\sum_{j=1}^d h^{-2} \sum_2^j \left[u_j^h(\underline{x}) - u_j^h(\underline{y})\right]^2 + \\ &\quad \textstyle\sum_{j=1}^d h^{-2} \sum_3^j \left[u_j^h(\underline{x}) - u_j^h(\underline{z})\right] u_j^h(\underline{x}) - \\ &\quad \textstyle\sum_4 p^h(\underline{x}) \sum_{j=1}^d \partial_j^h u_j^h(\underline{x}), \end{aligned}$$

where the point $\underline{x}$ in $\sum_1^j$ runs over all interior positions of of $u_j^h(x)$ (points ⓙ in Fig. 18.1, $j = 1, 2$); the pair $\{\underline{x}, \underline{y}\}$ in $\sum_2^j$ runs over all pairs of neighboring interior positions of u_j^h; the pair $\{\underline{x}, \underline{z}\}$ in $\sum_3^j$ runs over all pairs of neighboring positions u_j^h, with $\underline{x}$ being an interior position (ⓙ in Fig. 18.1, $j = 1, 2$) and $\underline{z}$ being a boundary or exterior position (j in Fig. 18.1); and $\underline{x}$ in $\sum_4$ runs over all cell centers (p in Fig. 18.1), including boundary cells due to the vanishing of the boundary conditions. The term with $\sum_4$ vanishes by (18.6a), since $F_0^h \equiv 0$. In the $\sum_3^j$ term, either $\underline{z}$ is a boundary point, therefore $u(\underline{z}) = 0$ and the term is non-negative; or $\underline{z}$ is an exterior point. By the definition of exterior values, we get (for $G^h \equiv 0$) $u_j^h(\underline{z}) = 2u_j^h(\underline{x}) - \frac{1}{3}u_j^h(\underline{y})$, where $\underline{y}$ is the interior neighbor of $\underline{x}$ opposite to $\underline{z}$. Hence,

$$\sum_{j=1}^d \left\{ \sum_5^j \left[u_j^h(\underline{x}) - u_j^h(\underline{y})\right]^2 + \sum_3^j \left[3u_j^h(\underline{x})^2 + u_j^h(\underline{y})^2 - \frac{7}{3}u_j^h(\underline{x})u_j^h(\underline{y})\right] \right\} \leq 0,$$

where $\sum_5^j$ runs as $\sum_2^j$ does, except for terms added to $\sum_3^j$. This form is positive definite, hence $u_j^h \equiv 0$. By (18.6b), $p^h \equiv$ const..

18.3 Distributive relaxation

Designing the relaxation scheme for (18.7) by the general approach of §3.7, note that L^h is almost triangular. To triangularize it, it is enough to transform its first column by multiplying L^h on the right with

$$M^h = \begin{pmatrix} -\Delta^h & 0 & \cdots & \cdots & 0 \\ -\partial_1^h & 1 & 0 & \cdots & 0 \\ \vdots & 0 & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 \\ -\partial_d^h & 0 & \cdots & 0 & 1 \end{pmatrix}, \tag{18.10}$$

(The first column of M^h could be obtained as the cofactors of the first row of L^h, divided by their maximal common divisor $(-\Delta^h)^{d-1}$.) This distribution operator M^h implies the following relaxation scheme.

First, for each $1 \le j \le d$, the *j-th* momentum equation (18.6b) is relaxed by scanning, in some prescribed order (RB ordering is best), all the interior j-face centers, changing at each such point $\underline{x}$ the current value $\tilde{u}_j^h(\underline{x})$ so as to satisfy the momentum equation at $\underline{x}$. Then, the continuity equation (18.6a) is relaxed by scanning the cell centers (preferably in RB ordering), introducing at each such center $\underline{x}$ the following changes:

$$\tilde{u}_j^h(\underline{x} + \tfrac{1}{2}\underline{h_j}) \leftarrow \tilde{u}_j^h(\underline{x} + \tfrac{1}{2}\underline{h_j}) + \delta, \quad (j = 1, \ldots, d), \tag{18.11a}$$

$$\tilde{u}_j^h(\underline{x} - \tfrac{1}{2}\underline{h_j}) \leftarrow \tilde{u}_j^h(\underline{x} - \tfrac{1}{2}\underline{h_j}) - \delta, \quad (j = 1, \ldots, d), \tag{18.11b}$$

$$\tilde{p}^h(\underline{x}) \leftarrow \tilde{p}^h(\underline{x}) + \frac{2d}{h}\delta, \tag{18.11c}$$

$$\tilde{p}^h(\underline{x} + \underline{h_j}) \leftarrow \tilde{p}^h(\underline{x} + \underline{h_j}) - \frac{1}{h}\delta, \quad (j = 1, \ldots, d), \tag{18.11d}$$

$$\tilde{p}^h(\underline{x} - \underline{h_j}) \leftarrow \tilde{p}^h(\underline{x} - \underline{h_j}) - \frac{1}{h}\delta, \quad (j = 1, \ldots, d), \tag{18.11e}$$

where

$$\delta = \frac{h}{2d} r_0^h(\underline{x}) = \frac{h}{2d} \left(F_0^h - \sum_{j=1}^{d} \partial_j^h \tilde{u}_j^h \right)_{\text{before the changes}}, \tag{18.12}$$

so that *after* these changes, the new residual $r_0^h(\underline{x})$ vanishes.

The relaxation of the continuity equation, and its modification near a boundary, are shown in Fig. 18.2 on the next page.

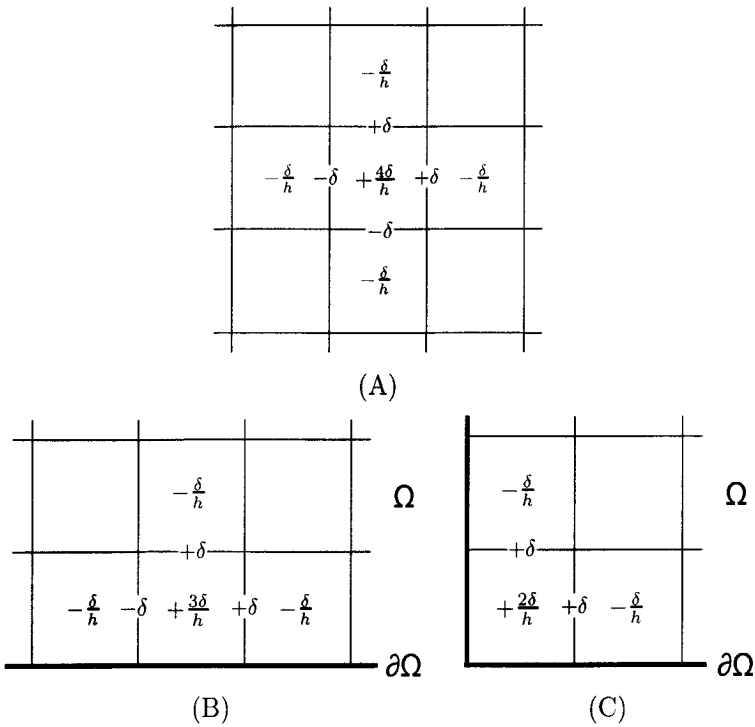


Figure 18.2. ***Continuity-equation relaxation step in two-dimensional Stokes equations.***

(A) The central cell is relaxed by 9 simultaneous changes. The amount of change is displayed at the position of the changed variable (cf. Fig. 18.1). $\delta = hr_0^h(\underline{x})/4$, where $r_0^h(\underline{x})$ is the dynamic residual at the relaxed cell.

(B) Configuration of changes in a boundary cell. $\delta = hr_0^h(\underline{x})/3$.

(C) Configuration of changes in a corner cell. $\delta = hr_0^h(\underline{x})/2$.

Observe that *near* the boundary, unlike the situation away from it, the relaxation of the continuity equation at a point does introduce slight changes in neighboring momentum residuals, as if locally contradicting the triangularity of L^hM^h. But these slight changes do not cause later (when the momentum equations are relaxed) significant "feed-back", i.e., too large changes back in r_0^h, because near the boundary feed-back changes are partly "absorbed" by the boundary.

The **smoothing factor** is the same as for GS relaxation for L^hM^h, i.e., the same as for Δ^h. Hence, in two dimensions, for lexicographic order-

ing $\overline{\mu} = .5$ and for RB ordering, if used in all three passes, $\overline{\mu}_1 = \overline{\mu}_2 = .25$ and $\overline{\mu}_3 = .32$ (cf. (3.2)). For $d = 3$ and lexicographic ordering, $\overline{\mu} = .563$.

18.4 Multi-grid procedures

For multi-grid processing of Stokes equations we use a sequence of grids (levels) with meshsizes $h_1, \ldots, h_M$, where $h_{k+1} = \frac{1}{2}h_k$, and where the grid lines (or grid planes) of level k are every other grid line (plane) of level $k+1$. Hence, each cell of level k is the union of 2^d cells of level $k+1$. In two dimensions ($d = 2$) the configuration is shown in Fig. 18.3. Instead of $\underline{F}^h$, $\underline{u}^h$, p^h, $\underline{r}^h$, μ_j^h and ∂_j^h used in §18.2–18.3 and in (17.7), the discrete functions and operators on the *k-th* level are now denoted by $\underline{F}^k$, $\underline{u}^k$, p^k, $\underline{r}^k$, μ_j^k and ∂_j^k, respectively.

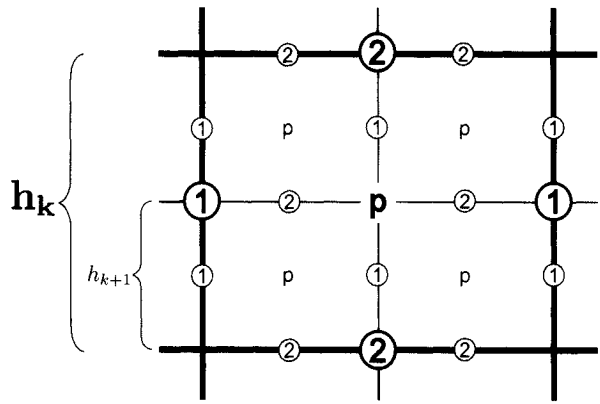


Figure 18.3. ***A coarse-grid cell divided into fine-grid cells.*** *The notation of Fig. 18.1 is used, with heavy type for the coarse grid and light type for the fine grid.*

We have solved Stokes equations both with the Correction Scheme (CS) and with the Full-Approximation Scheme (FAS), getting of course identical results. We describe here the procedures in terms of FAS, since CS is not extendible to the nonlinear Navier-Stokes equations.

Coarse-to-fine interpolations. In the FMG algorithm, to obtain residuals smaller than their truncation errors, the *first* coarse-to-fine interpolation has to be of order at least four for the velocities and at least three for the pressure (see §7.1). The design of such interpolations is straightforward, although it turns out somewhat cumbersome near boundaries. The coarse-to-fine interpolation of *corrections* has to be of orders at least two for the velocities and one for the pressure (see §4.3). We used bilinear (i.e., order two) interpolations for both.

The **fine-to-coarse transfers** are made by averaging. For the FAS transfer of u_j^{k+1} we can use the same averaging as for r_j^{k+1}, $(j = 1, \ldots, d)$,

which can be either the minimal-operation transfer

$$I_{k+1}^k r_j^{k+1} = \mu_1^{k+1} \cdots \widehat{\mu_j^{k+1}} \cdots \mu_d^{k+1} r_j^{k+1}, \quad (j = 1, \ldots, d), \tag{18.13}$$

or the full weighting

$$I_{k+1}^k r_j^{k+1} = \mu_j^{k+1} \mu_1^{k+1} \cdots \mu_d^{k+1} r_j^{k+1}, \quad (j = 1, \ldots, d), \tag{18.14}$$

where the hat in (18.13) indicates the term to be skipped in the sequence. The residual-weighting (18.13) is less expensive than (18.14), especially since it requires calculating only one half of the fine-grid residuals. But (18.14) is more reliable in the nonlinear case and near boundaries, since it is "full" (see §4.4).

The FAS transfer of $\tilde{p}^{k+1}$ can be made with the same weighting as the transfer of the continuity-equation residuals

$$I_{k+1}^k r_0^{k+1} = \mu_1^{k+1} \cdots \mu_d^{k+1} r_0^{k+1}, \tag{18.15}$$

which is both simplest and "full". In fact, if the minimal-operations transfer (18.13) is used for the velocities $\tilde{u}_j^{k+1}$, then (18.15) need not really be calculated: If the FAS continuity equation at level k is written in the form

$$\sum_{j=1}^{d} \partial_j^k u_j^k = f_0^k \tag{18.16a}$$

(where $f_0^l = F_0^l$ at the currently finest level l), it is easy to see that (18.15) is equivalent to

$$f_0^k = \mu_1^{k+1} \cdots \mu_d^{k+1} f_0^{k+1}, \quad (k < l), \tag{18.16b}$$

which does not depend on the current approximation $\underline{\tilde{u}}^{k+1}$.

The compatibility condition (18.9) is automatically obtained (up to round-off errors) on all levels provided it holds on the finest one. This results directly from (18.15).

18.5 Numerical results

Our early (1978) experiments with the above procedures, in various cycling and FMG algorithms, on two-dimensional rectangular domains (chosen only for programming simplicity), are described in [BD79] and in more details in [Din79]. The program itself is available [MUG84]. The experiments were not optimal because we have used lexicographic instead of RB ordering in relaxation (RB was used in our recent experiments with non staggered grids; see §18.6). The asymptotic convergence rates were 20% slower than those predicted by the smoothing rate $\overline{\mu}$, but not more than 6% slower than predicted by the two-level analysis. The fact that the smoothing rate is not fully exploited indicates that better convergence rates may be

obtainable with better inter-grid transfers, but this does not seem to worth the extra work: The obtained rates are good enough to give a solution below truncation errors by an FMG algorithm with one V(2,1) cycle per level (cf. Fig. 1.2). With RB relaxation, V(1,1) should already suffice (see §18.6).

18.6 Non-staggered grids

Having received complaints about the inconvenience of staggered grids despite their advantages (see §17.6), we later experimented with Stokes (and Navier-Stokes) solvers on conventional, non-staggered grids, where all unknowns are defined at the same gridpoints. The approach is to use a quasi-elliptic approximation, but to properly average the results. Thus, ∂_j^h in (18.6) or (18.7) has throughout been changed into the (long) central differencing $\partial_j^{2h} = \mu_j^h \partial_j^h$, replacing (18.8) by

$$\tilde{\det} L^h(\underline{\theta}) = h^{-2d} \left\{ \sum_{j=1}^{d} \left(2 \sin \frac{\theta_j}{2} \right)^2 \right\}^{d-1} \left\{ \sum_{j=1}^{d} \sin^2 \theta \right\}. \tag{18.17}$$

This symbol vanishes for some $|\underline{\theta}| = \pi$, showing L^h to be only quasi-elliptic.

More precisely, there are the high-frequency components

$$\begin{pmatrix} p \\ \underline{u} \end{pmatrix} = \begin{pmatrix} 1 \\ \underline{0} \end{pmatrix} e^{i\underline{\theta}\cdot\underline{x}/h} \quad (\theta_j = 0 \text{ or } \pi,\ 1 \le j \le d;\ |\underline{\theta}| = \pi), \tag{18.18}$$

which satisfy the homogeneous difference equations. These components by themselves do not really matter more than adding a constant to the pressure: They do not affect $\partial_j^{2h} p^h$, nor any other term in the difference equations. But other components, in the *neighborhood of* (18.18), do matter for the solution. They constitute high-frequency components which are not locally controlled. They will give rise to a subtle kind of instability (see §17.6), and they will not efficiently be reduced by relaxation, hence will be slow to converge in conventional multigrid algorithms. (Fast convergence can still be obtained by modified coarse-grid functions or by AMG algorithms; see §4.2.2 and 13.1.)

Both troubles (instability and slow smoothing) are closely related and easily cured by the same simple device: The bad components can be eliminated by *averaging the pressure*, i.e., replacing

$$\tilde{p}^h \to (\mu_1^h \dots \mu_d^h)^2 \tilde{p}^h. \tag{18.19}$$

This operation need to be done only on the final result. Similar bad convergence of some high-frequency components of an *intermediate* level does not matter, since those components are efficiently reduced by the next-finer-level relaxation (cf. §12).

To test this approach, without any interference of questions related to boundary conditions, we have studied the two-dimensional ($d = 2$) case, on the square $\{|x|, |y| \leq \pi\}$, with periodic boundary conditions. With such boundary conditions, u_j and p are determined only up to an additive constant each. We have used distributive Gauss-Seidel relaxation based on the distribution operator

$$M^h = \begin{pmatrix} -\Delta^h & 0 & 0 \\ -\partial_1^{2h} & 1 & 0 \\ -\partial_d^{2h} & 0 & 1 \end{pmatrix}, \tag{18.20}$$

(cf. (18.10)), with RB ordering in each of its three passes (corresponding to the three differential equations). Inter-grid transfers were standard: Full weighting (4.6), bilinear interpolation of corrections and cubic FMG interpolation of the first approximation. The coarse-grid operator is the same non-staggered, central Stokes operator as employed on the finest grid.

The compatibility condition, analogous to (18.5), seems at first to cause some troubles. In the discrete approximation it breaks up into 4 different conditions, each obtained by summing the discrete continuity equation on one of the four staggered sub-grids into which that equation is decoupled. Even if we take the trouble to satisfy these four conditions on the finest grid, they will not be satisfied on coarser grids, unless the coarse-grid equations are adjusted, e.g., by adding four suitable constants to F_0^k, one constant on each subgrid. Actually, all this is not needed. The said adjustment is below the level of the coarse-grid truncation, hence it does not improve the quality of the coarse-grid contribution to the fine-grid convergence. The fine-grid compatibility condition is itself similarly unimportant, unless one wants to solve the algebraic system below truncation errors. We simply ignore this condition at all levels.

Two additional compatibility conditions emerge because of our periodic boundary conditions, each obtained by summing one of the momentum equations over the grid. In the differential equations chosen by us these conditions are automatically satisfied, since we calculate a right-hand side $\underline{F}$ from a known solution (so that we know the exact differential solution and can compare our results with it). For reasons similar to the above, we ignored satisfying these compatibility conditions, too, in our discrete approximations, whether fine or coarse.

Components in the *neighborhood* of (18.18), eventually almost eliminated by (18.19), are still present, even though their influence on the residuals must *initially* be very small. Hence, *asymptotically* (after many cycles) we must get slow convergence. Indeed, on large $N \times N$ grids we have obtained asymptotic convergence factors of about $1 - 160N^{-2}$ per either V(1,1) or W(1,1) cycle. Initially, however, those components are not important and convergence factors are excelent; e.g., in the experiments of Table 18.1 the average convergence factor for the first five cycles was always better than .2 per V(1,1) cycle.

The main point of course is that the convergence rate of high-frequency components is immaterial by itself, since we do not need to converge them much to get below their large truncation errors. What does then matter is how many cycles per level an FMG algorithm needs in order to solve the problem to below truncation. Typical results are summarized in Table 18.1. They show that *convergence well below truncation errors is already obtained for an algorithm with only* 3.6 *RWUs* (relaxation work units). Being based on the RB relaxation, the algorithm is fully vectorizable.

FMG Algorithm			Results			
Grid Size	Cycle Type	n	$\|u-\tilde{u}^h\|_\infty$	$\|v-\tilde{v}^h\|_\infty$	$\|p-\tilde{p}^h\|_\infty$	
					Without Averaging	With Averaging
16×16	V	1	.0259	.0379	.4589	.1984
		2	.0217	.0224	.3306	.1671
		3	.0209	.0213	.3247	.1662
	W	1	.0259	.0356	.4400	.1873
		2	.0259	.0219	.3287	.1666
		3	.0259	.0216	.3275	.1662
32×32	V	1	.0056	.0107	.1216	.0821
		2	.0044	.0069	.0774	.0556
		3	.0043	.0068	.0627	.0541
	W	1	.0047	.0090	.1248	.0813
		2	.0045	.0069	.0789	.0564
		3	.0043	.0068	.0630	.0543
64×64	V	1	.0019	.0036	.0345	.0172
		2	.0020	.0029	.0180	.0141
		3	.0020	.0028	.0154	.0141
	W	1	.0020	.0032	.0367	.0187
		2	.0020	.0029	.0184	.0143
		3	.0020	.0028	.0158	.0141

Table 18.1. ***Stokes solutions on non-staggered grid.***

Results are shown for an FMG solution of a problem whose exact differential solution is $u = v = p = \sin(\cos(x+2y))$, *the right-hand sides* F_0 *and* $\underline{F}$ *being calculated accordingly.* F_0^h *and* $\underline{F}^h$ *at all levels are obtained by averaging* F_0 *and* $\underline{F}$ *on* $h \times h$ *squares. The approximate solution* $(\tilde{u}^h, \tilde{v}^h, \tilde{p}^h)$ *is obtained by the algorithm explained in the text, with relaxation counts* $\nu_1 = \nu_2 = 1$ *(see §1.4, 1.6; the V-cycle algorithm is exactly that of Fig. 1.2). The undetermined additive constants in* $\tilde{u}^h$, $\tilde{v}^h$ *and* $\tilde{p}^h$ *were of course properly subtracted off.* n *denotes the number of cycles per level.*

Chapter 19

Steady-State Incompressible Navier-Stokes Equations

19.1 The differential problem

Using the notation of §18.1, the steady-state incompressible Navier-Stokes equations in d dimensions can be written in the form

$$\underline{\nabla} \cdot \underline{u} = F_0 \tag{19.1a}$$

$$Q\underline{u} + \underline{\nabla} p = \underline{F}, \tag{19.1b}$$

$$Q := -\frac{1}{R}\Delta + \underline{u} \cdot \underline{\nabla} = -\frac{1}{R}\Delta + \sum_{j=1}^{d} u_j \partial_j, \tag{19.2}$$

R being the Reynolds number; i.e., R^{-1} is a scaled viscosity-over-density coefficient. (It is assumed that time and distance are scaled so that $\underline{u}$ is $O(1)$ and the domain dimensions are $O(1)$.) The principal part of this system is the Stokes system (18.1) (rescaling $p \leftarrow Rp$), hence for $R \leq O(1)$ the solution processes of §18 are directly applicable here (using FAS of course to deal with the nonlinearity). But we will be interested in solving also for large R, hence we will look at the corresponding *subprincipal operator* (cf. §2.1)

$$L = \begin{pmatrix} 0 & \partial_1 & \cdots & \cdots & \partial_d \\ \partial_1 & Q & 0 & \cdots & 0 \\ \vdots & 0 & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 \\ \partial_d & 0 & \cdots & 0 & Q \end{pmatrix}. \tag{19.3}$$

Observe indeed that (19.3) is not the full Newton linearization for (19.1); only subprincipal terms are kept. Since $\det L = \Delta Q^{d-1}$, (19.1) is again an elliptic system of order $2d$ and therefore requires d boundary conditions.

Usually the velocities are given on the boundary as in (18.4), leading again to the computability condition (18.5).

For $R \to \infty$, $\det L \to -\Delta(\underline{u} \cdot \underline{\nabla})^{d-1}$, hence the streamlines are the subcharacteristics (characteristic lines of the reduced equation), and in the limit only one condition is needed all along the boundary, plus $d-1$ conditions at points where the flow enter the domain (because $\underline{u} \cdot \underline{\nabla}$ is a streamwise derivative which, with the singular perturbation $-\frac{1}{R}\Delta$, permits discontinuity only at the exit end of each streamline). Typically, for $R \to \infty$, the velocities $\underline{u}$ are given at entrance points, and either the normal velocity or the pressure at other boundary points. See for example [Tem77] for theoretical investigations of this system.

19.2 Staggered finite-difference approximations

The discretization is carried out on the same staggered grid as before (Fig. 18.1), using the difference equations

$$\sum_{j=1}^{d} \partial_j^h u_j^h = F_0^h \qquad \text{at cell centers} \tag{19.4a}$$

$$Q^h u_j^h + \partial_j^h p = F_j^h \quad \text{at centers of } j\text{-faces}, \quad (j = 1, \ldots, d) \tag{19.4b}$$

where Q^h is some difference approximation to Q. (Non-staggered grids are discussed in §19.5.) Since $det L^h = -\Delta^h (Q^h)^{d-1}$, it is clear that L^h has good h-ellipticity measure if and only if Q^h does. Hence, all we have to construct is a good approximation to Q. For small to moderate $hR|\underline{u}|$ (i.e., $hR|\underline{u}|$ not much larger than 1) this can be done by central differencing. But for larger $hR|\underline{u}|$, upstream differencing or explicit artificial viscosity terms should be used.

The artificial viscosity terms may be anisotropic, so that the total (physical and artificial) viscosity has the form $-\sum_i \overline{\beta}_i h(\partial_i^h)^2$. For stability of the *simplest* DGS schemes (§19.3), $\overline{\beta}(i) \geq .5 \max |u_i(y)|$ is needed, where the maximum is taken over all y neighboring x. Upstream differencing is the same as $\overline{\beta}_i = .5|u_i|$. For large R, sharp cross-stream changes (large solution changes per cross-stream meshsize) can travel with the stream, and one may like to avoid smearing them by preventing cross-stream artificial viscosity. This is only possible by *strong alignment* (cf. §2.1), i.e., by using a grid (sometimes through the method of §9.3) such that one of its principal directions is (nearly) aligned with the stream throughout a large subdomain. One can then use $\overline{\beta}_i = O(h|\underline{u}|)$ (necessary for stability) in the stream direction, together with zero (or small) cross-stream artificial viscosity. In three-dimensional problems it may sometimes be difficult and unnecessary to have the flow (nearly) aligned with one grid direction, but a grid can be used so that each streamline (nearly) stay in one grid plane.

We call this *plane alignment*. In this case only sharp changes perpendicular to these planes are resolvable. For that purpose, zero artificial viscosity perpendicular to the planes should be used.

Another, more special case of strong alignment may arise near boundaries. Namely, to obtain sharp numerical boundary layers on grids that do not resolve the *physical* boundary layer, the Q^h operators near the boundary should be constructed so that tangential components of $Q^h \underline{u}$ do not straddle the boundary layer, i.e., do not include boundary values. Such a choice of Q^h can be made without aligning the grid. It should similarly be made at any *known* layer of sharp cross-stream transition.

The artificial viscosity may introduce $O(h)$ error, but such errors can be eliminated by omitting the artificial terms from the residuals calculated for the transfer to the next coarser grid (cf. §10.2); except that the rule of not straddling the boundary layer should still be kept by those residuals, too.

For a full discussion of the discretization and multigrid procedures for the convection-diffusion operator Q, see [Bra81a].

In case the boundary-layer interaction is important (e.g., in the driven cavity problem, where this interaction determines the entire flow), the boundary layer should be resolved. One can use then anisotropic local refinements (see §9.2), with local coordinate transformations in case the boundary is not along grid directions (§9.3).

19.3 Distributive relaxation

Generalizing the scheme in §18.3 to *any* operator Q^h, relaxation is guided by the distribution operator

$$M^h = \begin{pmatrix} Q^h & 0 & \cdots & \cdots & 0 \\ -\partial_1^h & 1 & 0 & \cdots & 0 \\ \vdots & 0 & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 \\ -\partial_d^h & 0 & \cdots & 0 & 1 \end{pmatrix} \tag{19.5}$$

and thus proceeds as follows.

The *j-th* momentum equation (19.4b) is relaxed by changing $\tilde{u}_j^h$ only, in any order and manner suitable for the operator Q^h. Thus, for $hR|\underline{\tilde{u}}| \leq O(1)$ the best perhaps is the Red-Black (RB) Gauss-Seidel scheme: $\tilde{u}_j^h$ is changed at each point so as to satisfy (19.4b) at that point, the red points being scanned first, the black next. For $hR|\underline{\tilde{u}}| \gg 1$, this relaxation is still one of the best, except for cases of intended strong alignment described above (§19.2).

In those cases, block relaxation must be used, but only in the specific strong-alignment direction, in its specific subdomain. This means streamwise line relaxation, except for plane relaxation in cases of plane alignment.

Furthermore, to obtain sharp numerical boundary layers, one should relax in blocks (i.e., simultaneously, in one or several blocks) exactly those equations where the special Q^h was constructed for that end (cf. §19.2). In many cases, the block relaxation can be replaced by a suitable downstream or ILU scheme (cf. §3.3).

In the case of plane alignment, the plane relaxation could be replaced by point relaxation if the grid is not coarsened in the direction perpendicular to the planes (cf. §4.2.1 and 3.3). Complicated alternating-direction line relaxation schemes are needed only if a fast solution is desired with errors far below truncation errors. A simple RB scheme, fully parallelizable and vectorizable, can thus most often be used.

Having relaxed in this way one pass per each momentum equation ($j = 1, \ldots, d$), we then make a pass of relaxation for the continuity equation (19.4a), by scanning the *cells* one by one, preferably in redblack ordering. At each cell the distributive relaxation step resulting from (19.5) is a generalization of Fig. 18.2: denoting by ξ^h the characteristic function of the relaxed cell (i.e., $\xi^h = 1$ at that cell center, and $\xi^h = 0$ at all other cell centers), the relaxation step changes all functions $\tilde{u}_j^h$ and $\tilde{p}^h$ by the prescription

$$\tilde{u}_j^h \leftarrow \tilde{u}_j^h - \delta h \partial_j^h \xi^h, \quad (j = 1, \ldots, d) \tag{19.6a}$$

$$\tilde{p}^h \leftarrow \tilde{p}^h + \delta h Q^h \xi^h, \tag{19.6b}$$

where δ is still given by (18.12).

The smoothing factor $\overline{\mu}$ is the slowest among the factors obtained for the triangular operator

$$L^h M^h = \begin{pmatrix} -\Delta^h & \partial_1^h & \cdots & \cdots & \partial_d^h \\ 0 & Q^h & 0 & \cdots & 0 \\ \vdots & 0 & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 \\ 0 & 0 & \cdots & 0 & Q^h \end{pmatrix}. \tag{19.7}$$

Hence, $\overline{\mu} = \max\{\overline{\mu}^\Delta, \overline{\mu}^Q\}$, where $\overline{\mu}^\Delta$ is the smoothing factor of RB relaxation for Δ^h, and $\overline{\mu}^Q$ is the smoothing factor of the relaxation of $Q^h \tilde{u}_j^h$. Hence $\overline{\mu}^\Delta = .25$ if one or two sweeps are performed per cycle, while $\overline{\mu}^\Delta = .32$ in case of three sweeps per cycle. If down-stream relaxation is used for the momentum equations (hence for Q^h), one can obtain $\overline{\mu}_Q \le \overline{\mu}^\Delta$ and hence $\overline{\mu} = \overline{\mu}^\Delta$. As discussed above, however, especially in cases of widely varying stream directions, it is not important to obtain perfect smoothing by relaxing in all these directions. The relaxation rules specified above, deviating from simple (RB) point relaxation only in some cases of strong alignment, ensure fast reduction of all high-frequency components, except for nearly characteristic components which anyway have large truncation

errors almost everywhere in the flow field (cf. §3.3). For this to hold the total viscosity coefficients $\underline{\beta}_i$ (see §19.2) should be slightly larger than the minimum, e.g. $\underline{\beta}_i = .7|u_i|$. In case the minimum $\underline{\beta}_i = .5|u_i|$ is used, as in upstream differencing, some highest frequencies are not reduced (unless downstream relaxation ordering is everywhere ensured); but this is not important from the point of view of *differential* smoothing (cf. §12), which will in fact be damaged if $\underline{\beta}_i$ is increased too much ($\underline{\beta}_i = .7|u_i|$ is still good). In any case, the distribution matrix (19.5) reduces the problem of relaxing the Navier-Stokes system into consideration concerning the relaxation of the scalar convection-diffusion operator Q^h (which is in detail studied in [Bra81a]).

19.4 Multigrid procedures and numerical results

The grids, their relative positions and the interpolation procedures between them are generally the same as for the Stokes equations (§18.4). Because of the nonlinearity, FAS is of course used (see §8), and the full weighting (18.14) is preferred over (18.13) in the fine-to-coarse transfers of both the velocities and the momentum residual functions.

For $R > O(h^{-1})$, the momentum residuals themselves can be calculated in two ways, using either the same $O(h)$ approximation Q^h used in relaxation (see §19.2), or the $O(h^2)$ central approximation to Q^h_* (i.e., Q^h with zero artificial viscosity). The latter is the "double-discretization" scheme (see §10.2) which exploits the fact that Q^h is a better approximation for the high-frequency components, hence used in relaxation, while Q^h_* is a better (higher) approximation for the low-frequency components which converge through the interaction with the coarse grid. Q^h_* should, however, respect intended discontinuities (boundary layers) in the same way that Q^h does, even if it means $O(h)$ local truncation errors; the global discretization errors (e.g., the L_1-norms of $u - \tilde{u}^h$, $v - \tilde{v}^h$ and $p - \tilde{p}^h$) will still be $O(h^2)$.

For large hR it is also advisable to use W cycles or accommodative algorithms (see §6.2). On very coarse grids, where velocity changes per meshsize are comparable to the velocity itself, BGS relaxation (see §3.4 and the end of §5.6) is safer than the DGS relaxation described above.

Our 1978 numerical results are reported in [Din79], with two examples in [BD79]. They clearly show convergence to below truncation errors in an FMG algorithm with one accommodative cycle per level, making only two relaxation sweeps on the finest grid. At the time we were worried about asymptotic convergence rates, but we should have not been; see the end of §4.1. The results will be tabulated in a separate report.

19.5 Results for non-staggered grids

The non-staggered approach to the Stokes equations (§18.6) can also be used for the Navier-Stokes equations. On a non-staggered grid the short-

central difference operator ∂_j^h in (19.4) should throughout be replaced by the long-central ∂_j^{2h}. Hence, ∂_j^h should be replaced by ∂_j^{2h} in the distribution operator (19.5), too (yielding M^h as in (18.20) with Δ^h replaced by Q^h). As in Stokes equations, error components around (18.18) are not efficiently reduced by relaxation; but in the Navier-Stokes case, non-linear interactions can actually amplify these components. Therefore, the pressure averaging (18.19) must now be applied not only to the final results but also to any pressure correction, just before it is interpolated to a finer grid.

This approach was tested with RB ordering, full residual weighting (4.6), bilinear interpolation of corrections, W(2,0) cycles and an FMG algorithm with bi-cubic interpolation of solutions. For clarity, Table 19.1 shows the results for the case of periodic boundary conditions with the smooth non-aligned solution $u = v = p = 1 + .2\sin(\cos(x + 2y))$ and $R = \infty$. Compatibility conditions were ignored (see §18.6), but the correct averages of u, v and p were enforced, so as to make the periodic problem well posed. Double discretization was used, with $\underline{\beta} = \beta|u_i|$ in relaxation and $\underline{\beta} = \beta_*|u_i|$ in residual transfers.

h	grid	$(\beta, \beta_*) \rightarrow$	(1., 1.)		(.7, .7)		(1., 0.)
		# cycles $\rightarrow$	1	2	1	2	1
1/16	16 × 16			.1602		.1101	.1233
1/32	32 × 32			.0809		.0583	.0309
1/64	64 × 64		.0420	.0416	.0298	.0291	.0067

Table 19.1. ***Differential error in the FMG Algorithm for the two-dimensional Stokes equations on non-staggered grids.***

The error $\|u - \tilde{u}^h\|_1 + \|v - \tilde{v}^h\|_1 + \|p - \tilde{p}^h\|_1$ *obtained on increasingly finer FMG levels* h.

Chapter 20

Compressible Navier-Stokes and Euler Equations

The steady-state Euler equations of inviscid flows will be treated here as a special, limit case of the full steady-state Compressible Navier-Stokes (CNS) equations. Some terms can be dropped in the inviscid limit, but there is no essential difference between the numerical solution of inviscid flows and that of slightly viscous flows, because $O(h)$ *artificial* viscosity should anyway be used in relaxation, and it should closely resemble the *physical* viscosity to ensure that only physical discontinuities are admitted at $h \to 0$. The double-discretization scheme (§10.2) will be used; in the present context this simply means that the artificial viscosity, when needed, is employed only in the relaxation operators, not in the difference operators by which residuals are calculated in the fine-to-coarse transfers. The latter can generally use simple central differencing, but both types of operators should respect, as much as possible, the flow discontinuities. The multigrid processes will first be described in terms of the simpler quasi-linear form of the equations, discretized on a staggered grid, and then their modification to the conservation form and to non-staggered discretization will be discussed.

20.1 The differential equations

20.1.1 Conservation laws and simplification

The time-dependent Compressible Navier-Stokes (CNS) equations in d dimensions may be written in in conservation form as

$$\frac{\partial W}{\partial t} + \sum_{j=1}^{d} \partial_j F_j = \underline{f}, \tag{20.1}$$

where $\partial_j = \partial/\partial x_j$ and

$$W = \begin{pmatrix} \rho u_1 \\ \vdots \\ \rho u_d \\ \rho \\ e \end{pmatrix}, \quad F_j = \begin{pmatrix} \rho u_j u_1 + \tau_{j1} \\ \vdots \\ \rho u_j u_d + \tau_{jd} \\ \rho u_j \\ e u_j + \sum_i u_i \tau_{ij} - \kappa \partial_j \varepsilon \end{pmatrix}, \quad \underline{f} = \begin{pmatrix} f_1 \\ \vdots \\ f_d \\ f_\rho \\ f_\varepsilon \end{pmatrix},$$

$$\tau_{ij} = \tau_{ji} = -\mu(\partial_i u_j + \partial_j u_i) - \lambda \delta_{ij} \sum_{k=1}^{d} \partial_k u_k + p\delta_{ij}, \tag{20.2}$$

ρ being the fluid density, $\underline{u} = (u_1, \ldots, u_d)$ the velocity vector, e the total energy per unit volume, ε the specific internal energy, p the pressure, λ and μ the viscosity coefficients, and κ the coefficient of heat conductivity. (20.1) is a system of $d+2$ equations: the first d equations are the d momentum equations, next is the continuity (mass conservation) equation, and the last is the energy equation. The $d+2$ basic unknowns may be considered to be $\underline{u}$, ρ and ε, in terms of which e is given by

$$e = \rho\left(\varepsilon + \frac{q^2}{2}\right), \qquad q^2 = \sum_{i=1}^{d} u_i^2, \tag{20.3}$$

and p by the equation of state

$$p = \overline{p}(\varepsilon, \rho). \tag{20.4}$$

For a perfect gas, for example, the equation of state is $p = (\gamma - 1)\varepsilon\rho$, where γ is the ratio of specific heats. Generally, $p_\varepsilon = \partial\overline{p}/\partial\varepsilon$ and $p_\rho = \partial\overline{p}/\partial\rho$ are positive. The coefficients λ, μ and κ are given functions, usually, of ε (or functions of the fluid temperature, which in turn is a function of ε). We will treat these coefficients as constants, since they change slowly. The whole discussion below will remain precisely valid as long as any change in any of these coefficients over a meshsize is small compared with the coefficient itself.

In most aerodynamical applications the right-hand side $\underline{f}$ vanishes, but there are other applications where the external body force $(f_1, \ldots, f_d)$, or the mass source f_ρ, or the energy source f_ε do not vanish. A general $\underline{f}$ is assumed here, mainly because many of the numerical experiments are set to have known specified solutions by pre-arranging $\underline{f}$ accordingly.

The steady-state CNS equations are given by

$$\sum_{j=1}^{d} \partial_j F_j = \underline{f}, \tag{20.5}$$

together with (20.3) and (20.4). It will be convenient below to substitute (20.3) into equations (20.5), but to treat (20.4) as an additional equation

and p as an additional unknown. Allowing (20.4) not to be held exactly satisfied until convergence will substantially simplify the solution process, and more than justify the additional storage needed for p. Thus we will have a system of $n = d + 3$ differential equations in n unknown functions. The n-vector $U = (u_1, \ldots, u_d, \rho, \varepsilon, p)$ will serve as our vector of unknowns.

Simplified equations, but not in conservation form, are obtained as follows. The *i-th* momentum equation is simplified by subtracting from it u_i times the continuity equation. Then the energy equation is simplified by subtracting from it u_i times the *i-th* (simplified) momentum equation for $i = 1, \ldots, d$, and ε/ρ times the continuity equation. The resulting system is

$$\rho \sum_j u_j \partial_j u_i + \sum_j \partial_j \tau_{ij} = f_i \qquad (i = 1, \ldots, d) \tag{20.6a}$$

$$\sum_j \partial_j (\rho u_j) = f_\rho \tag{20.6b}$$

$$\rho \sum_j u_j \partial_j \varepsilon - \sum_j \partial_j (\kappa \partial_j \varepsilon) + \sum_{i,j} \tau_{ij} \partial_i u_j = f_\varepsilon \tag{20.6c}$$

$$p = \overline{p}(\varepsilon, \rho) \tag{20.6d}$$

which, in view of (20.2), is a system of n equations for the n unknowns U. In terms of this simpler system we will now study the principal and the inviscid subprincipal parts. This will tell us what boundary conditions are appropriate and which terms are locally dominant, which is important for designing the relaxation scheme and the form of the artificial viscosity terms.

20.1.2 The viscous principal part

The principal part of (20.6), i.e., the part of the linearized operator which contributes to the highest-order terms of its determinant, is the operator

$$L_p = \begin{pmatrix} -\mu\Delta - \overline{\lambda}\partial_{11} & \cdots & -\overline{\lambda}\partial_{1d} & 0 & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ -\overline{\lambda}\partial_{d1} & \cdots & -\mu\Delta - \overline{\lambda}\partial_{dd} & 0 & 0 & 0 \\ 0 & \cdots & 0 & \underline{u} \cdot \underline{\nabla} & 0 & 0 \\ 0 & \cdots & 0 & 0 & -\kappa\Delta & 0 \\ 0 & \cdots & 0 & 0 & 0 & 1 \end{pmatrix} \tag{20.7}$$

where $\partial_{ij} = \partial_i \partial_j$, $\Delta = \partial_{11} + \cdots + \partial_{dd}$, $\underline{u} \cdot \underline{\nabla} = u_1 \partial_1 + \cdots u_d \partial_d$ and $\overline{\lambda} = \lambda + \mu$. It can be shown by dimensional analysis that on a small enough scale the behavior of solutions to the original system depends only on L_p. Thus, on a sufficiently small scale, viscosity is the main mechanism that determines velocities, convection determines density, diffusion determines the internal energy, and they are all locally independent. It also follows from (20.7)

that the full CNS system requires d boundary conditions for the velocities (usually $\underline{u}$ is given) and one for ε (usually given in the form of boundary temperature or temperature gradient) on all boundaries, and an additional condition for ρ (or for p) at one end of each streamline.

20.1.3 Elliptic singular perturbation

Since $\underline{u}\cdot\underline{\nabla}$ is a factor of $\det L_p$, the steady-state CNS system is not elliptic: The streamlines p are its characteristic lines (the *only* characteristics, as long as the flow is viscous). This means that an addition of artificial h-ellipticity will be needed in local numerical processes, unless the grid exactly aligns with the stream. Therefore, and for uniform treatment of all artificial terms in the inviscid limit, we will regard already the CNS *differential* system as a limit, $\nu \to 0$ say, of an elliptic system, obtained by adding a singular-perturbation term to the continuity equation (20.6b), rewriting it (times ρ) as

$$\sum_{j=1}^{d} [\rho\partial_j(\rho u_j) - \partial_j(\nu\partial_j\rho)] = f_\rho. \tag{20.8}$$

ν should be positive for the additional term to be compatible (i.e., give a well-posed problem together) with the *time-dependent* system (20.1), so as to make the limits $\nu \to 0$ and $t \to \infty$ interchangeable. With $\nu > 0$, the additional term indeed represents a physical effect, namely, static molecular diffusion, which could normally be neglected.

The system (20.6) with (20.8) replacing (20.6b) is called the *augmented CNS (ACNS) system*. Its principal-part determinant is $\kappa\nu\mu^d\Delta^{d+2}$, so it is indeed elliptic. It requires the same $d+1$ boundary conditions (on $\underline{u}$ and ε) as before, plus a boundary condition on ρ (or p), on all boundaries. But the sign of ν ensures that the latter condition will affect the solution in the limit $\nu \downarrow 0$ only at points where the flow *enters* the domain. At non-entry boundaries, an artificial boundary layer (discontinuity in the limit) would be formed; but it can be avoided by using the original continuity equation (20.6b) as the extra boundary condition at such points. If *all* boundaries are such, however, we will have only *gradient* conditions on ρ along the boundaries, hence we will need an extra integral condition to uniquely determine the solution to the ACNS system, and also, in the limit, to the CNS system. This condition is usually the *total mass*, or some equivalent datum. Indeed, if the flow nowhere enters the domain, rigid walls are all around, then the total mass is determined only by the initial conditions, and therefore should be added as an extra condition to the steady-state equations.

20.1.4 Inviscid (Euler) and subprincipal operators

The inviscid case (Euler equations) is the system (20.6) with vanishing viscosities and heat conduction: $\lambda = \mu = \kappa = 0$. More precisely, the flow is inviscid (free of viscous and heat-conduction effects) where λ, μ and κ are small compared with $\rho l q$, where l is a typical length of change of $\underline{u}$ and ε. Usually there will be some particular narrow subdomains, such as boundary layers, where l will be just small enough to make the flow viscous. Thus, viscosity effects can seldom be completely neglected.

Anyway, wherever the flow is inviscid, the scale where viscosity dominates is much smaller than the scale of changes in the flow, which will later also be the scale of our grid h. So we like to isolate the terms which dominate the flow in that intermediate scale (small scale in terms of the flow geometry, but large enough to neglect viscosity and heat conduction). These are the *sub-principal* terms, defined as all the terms that are either principal or become principal when λ, μ and κ, or some of them, vanish. They form the following *sub-principal operator* L_s

$$L_s = \begin{pmatrix} Q_\mu - \overline{\lambda}\partial_{11} & \cdots & -\overline{\lambda}\partial_{1d} & 0 & 0 & \partial_1 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ -\overline{\lambda}\partial_{d1} & \cdots & Q_\mu - \overline{\lambda}\partial_{dd} & 0 & 0 & \partial_d \\ \rho^2\partial_1 & \cdots & \rho^2\partial_d & Q_\nu & 0 & 0 \\ p\partial_1 & \cdots & p\partial_d & 0 & Q_\kappa & 0 \\ 0 & \cdots & 0 & -p_\rho & -p_\varepsilon & 1 \end{pmatrix}, \tag{20.9}$$

where generally

$$Q_\alpha := -\underline{\nabla} \cdot (\alpha \underline{\nabla}) + \rho \underline{u} \cdot \underline{\nabla}. \tag{20.10}$$

This is the operator that should be kept in mind in the *local* processing, such as relaxation, and in the choice of discretization to be used with relaxation. The coefficients $\underline{u}$, ρ and p appearing in L_s are actually the values of some solution around which the flow is examined through principal linearization (see §3.4); they will always be derived from the *current approximate* solution $\tilde{U}$ (see §20.3.5). It can always be assumed that the current approximation is close enough to the solution, by employing continuation if necessary (see §8.3.2). The determinant of L_s , developed by its last row, is

$$\det L_s = Q_\mu^{d-1} \left\{ Q_\kappa Q_\mu \left(Q_\mu - \overline{\lambda}\Delta \right) - \left(\rho^2 p_\rho Q_\kappa + p p_\varepsilon Q_\nu \right) \Delta \right\}. \tag{20.11}$$

The *reduced (principal) operator* L_r is defined as L_s for $\lambda = \mu = \kappa = \nu = 0$, i.e., the principal part of the inviscid limit, namely,

$$L_r = \begin{pmatrix} Q_0 & \cdots & 0 & 0 & 0 & \partial_1 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & Q_0 & 0 & 0 & \partial_d \\ \rho^2\partial_1 & \cdots & \rho^2\partial_d & Q_0 & 0 & 0 \\ p\partial_1 & \cdots & p\partial_d & 0 & Q_0 & 0 \\ 0 & \cdots & 0 & -p_\rho & -p_\varepsilon & 1 \end{pmatrix}, \tag{20.12}$$

$$\det L_r = Q_0^d \left(Q_0^2 - \rho^2 a^2 \Delta\right) = \rho^{d+2} (\underline{u} \cdot \underline{\nabla})^d \left[(\underline{u} \cdot \underline{\nabla})^2 - a^2 \Delta\right], \qquad (20.13)$$

where $a := (p_\rho + \rho^2 p p_\varepsilon)^{\frac{1}{2}}$ is the *speed of sound.* (In the time-dependent inviscid case the operator in brackets in (20.13) is replaced by $[(\frac{\partial}{\partial t} + \underline{u} \cdot \underline{\nabla})^2 - a^2 \Delta]$, showing that a is the speed relative to the flow at which small disturbances would propagate.) The ratio $M = q/a = (\underline{u} \cdot \underline{u})^{\frac{1}{2}}/a$ is called the *Mach number.* Where $M < 1$ the flow is called *subsonic*, where $M > 1$ it is called *supersonic*, and the line where $M = 1$ is the *sonic line.* We can see from (20.13) that the steady state inviscid supersonic equations are hyperbolic, regarding the stream direction $\underline{u}$ as the time-like direction, with three families of characteristic lines and three characteristic speeds: q, $|a - q|$ and $|a + q|$. The steady-state inviscid subsonic equations are neither hyperbolic nor elliptic, and have only one family of characteristic lines, namely, the streamlines. In either case the equations are of order $d + 2$, hence require $d + 2$ boundary conditions per streamline. The only restriction on these imposed by (20.13) is that, in the subsonic case, at least one condition should be given all along the boundary (on both sides of each streamline). Actually the situation is more complicated since the flow can be partly subsonic and partly supersonic and, more importantly, acceptable solutions of the inviscid equations are only those obtainable as limits of solutions of the viscous equations. This latter requirement determines which of the boundary conditions of the full CNS equations will affect the inviscid flow away from the boundary, and which will be ignored in the inviscid limit, creating a discontinuity (boundary layer). It also determines what type of discontinuities (shocks) are permissible in the interior. The derived rules for permissible discontinuities are sometimes expressed as "entropy conditions".

Our approach here to the inviscid case will generally be to imitate the physics. Instead of deriving entropy conditions and then imposing them numerically, and instead of getting fully into the question of correct boundary conditions, we will locally (i.e., in relaxation) use a numerical scheme which contain artificial viscosity exactly analogous to the physical viscosity, thus ensuring correct selection of discontinuities. Our local processes need artificial viscosity anyway, to eliminate high-frequency errors.

The required magnitude of the artificial viscosity coefficients can be seen from $\det L_s$. They should be as effective on the scale of the meshsize h as other terms, hence, regarding each differentiation symbol as $O(h^{-1})$, the coefficients (λ,μ,ν,κ) should be chosen so as to make the order of $\det L_s$ homogeneous in h. It is easy to see that this is obtained if and only if the artificial λ, μ, ν and κ are all $O(h\rho|\underline{u}|)$, in which case $\det L_s$ is homogeneously of order h^{-d-2}.

20.1.5 Incompressible and small Mach limits

The incompressible limit is the case of vanishingly small $\partial\rho/\partial p$, or indefinitely large p_ρ. The main operator in this case is the cofactor of p_ρ in (20.9)

(reducing by one the number of equations, corresponding to the fact that ρ is no longer unknown). The resulting system is reducible: the momentum and continuity equations form a separate system of equations for $\underline{u}$ and p, easily simplified to (19.3) above. An exactly analogous situation arises if p_ε, instead of p_ρ, is large, and, more generally, whenever the Mach number is small. Thus, if one develops a multigrid solver for cases which include regions with small Mach, its discretization and relaxation should be efficient in the incompressible limit (19.3).

20.2 Stable staggered discretization

20.2.1 Discretization of the subprincipal part

The stable discretization constructed here is for use in relaxation processes, and will thus (see §2.1) be based on considerations concerning the subprincipal operator (20.9) with the *coefficients* $\underline{u}$, ρ, p, p_ρ, p_ε regarded as fixed. For all admissible values of these coefficients, the discretization and relaxation processes should be uniformly effective. In particular, they should accommodate the incompressible limit (cf. §20.1.5). We will therefore use a staggered grid as in §18.2 and 19.2, with ρ^h, ε^h and p^h (the discrete approximations to ρ, ε and p, respectively) defined at

$$x^h_k = x^h_0 + (k_1 h_1, \ldots, k_d h_d) \qquad \text{(cell centers)}, \tag{20.14}$$

and u^h_i (approximating u_i) at

$$x^{h,i}_k = x^h_k + .5\underline{h_i} \qquad (i\text{-face center}), \tag{20.15}$$

where $k = (k_1, \ldots, k_d)$ are vectors of integers, $h_1 \times \cdots \times h_d$ is the size of the grid-h cells, $\underline{h_i} = h_i \underline{e_i}$, and $\underline{e_i}$ is the i-*th* coordinate unit vector. In two dimensions ($d = 2$) the staggering is depicted in Fig. 20.1. With this staggering (20.9) can indeed be approximated by replacing ∂_i by its *short* central analog ∂^h_i,

$$\partial^h_i \Phi(\underline{x}) := \frac{1}{h_i}\left[\Phi(\underline{x} + .5\underline{h_i}) - \Phi(\underline{x} - .5\underline{h_i})\right],$$

replacing ∂_{ij} by $\partial^h_{ij} = \partial^h_i \partial^h_j$, and each Q_α by a proper approximation Q^h_α. Calling the resulting operator L^h_s, we find, similarly to (20.11), that

$$\det L^h_s = (Q^h_\mu)^{d-1}\left\{Q^h_\kappa Q^h_\mu\left(Q^h_\mu - \overline{\lambda}\Delta^h\right) - \left(\rho^2 p_\rho Q^h_\kappa + p p_\varepsilon Q^h_\nu\right)\Delta^h\right\}. \tag{20.16}$$

where $\Delta = \sum_j \partial^h_{jj}$ is the usual $(2d+1)$-point discrete Laplacian. Thus, the approximation is h-elliptic, provided Q^h_α are h-elliptic, $\alpha = \mu, \nu, \kappa$.

The approximation of these diffusion-convection operators is generally similar to that of Q^h in §19.2, with ρ/μ replacing R, with $O(h\rho|\underline{u}|)$ replacing $O(h|\underline{u}|)$, and with some modifications in case of shocks. Such modifications,

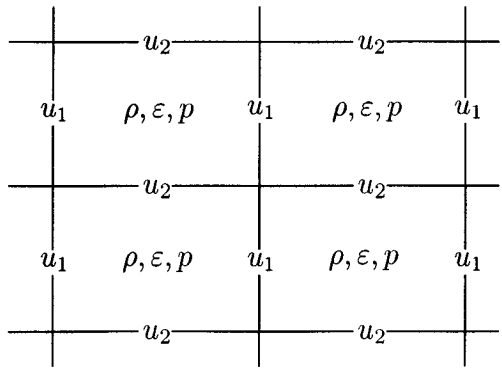


Figure 20.1. ***Grid staggering for compressible Navier-Stokes discretization.***

introduced of course in the conservative formulation, have been studied and the main emerging rule, as with boundary layers, is to avoid differencing across discontinuities: the stronger the shock, the more precisely rotated and upwinded the calculation of fluxes, hence the more weakly it straddles the shock. Where boundary layer and shocks are *not* present, we will thus approximate any Q_α at any gridpoint $\underline{x}$ by

$$Q_\alpha^h(\underline{x}) := \sum_{j=1}^{d} \left(-\partial_j^h \left(\alpha_j^h(\underline{x})\partial_j^h\right) + \overline{\rho}(\underline{x})\overline{u}_j^h(\underline{x})\partial_j^{2h}\right), \tag{20.17}$$

where $\overline{\rho}(\underline{x})$ and $\overline{u}_j^h(\underline{x})$ are central averages of ρ^h and u_j^h, respectively, over points nearest to $\underline{x}$, and

$$\alpha_j^h(\underline{x}) := \max\{\alpha, \beta h_j \rho^h |u_j^h|\}. \tag{20.18}$$

The max in (20.18) is taken in principle over all values of h_j, ρ^h and u_j^h in some neighborhood of $\underline{x}$. Usually *any* neighboring values, not necessarily exactly maximal ones, can be taken, except near *stagnation point*, where $\underline{u}$ nearly vanishes. β is an $O(1)$ parameter; $\beta = \frac{1}{2}$ normally gives upstream differencing, but slightly larger values ($\beta = .6$ or $.7$) may give better results (cf. [Bra81a]).

20.2.2 The full quasi-linear discretization

Guided by the above discretization scheme for the sub-principal part, the discrete approximation to the full CNS system (20.6) on the staggered grid

(20.14)-(20.15) will be written as

$$Q^h_\mu u^h_i - \overline{\lambda}\sum_j \partial^h_{ij} u^h_j + \partial^h_i p^h = f^h_i \text{ at } x^{h,i}_k,\ (i = 1,\dots,d) \quad (20.19a)$$

$$Q^h_\nu \rho^h + (\rho^h)^2 \sum_j \partial^h_j u^h_j = f^h_\rho \text{ at } x^h_k \quad (20.19b)$$

$$Q^h_\kappa \varepsilon^h + p^h \sum_j \partial^h_j u^h_j - B^h(\underline{u}^h) = f^h_\varepsilon \text{ at } x^h_k \quad (20.19c)$$

$$p - \overline{p}(\varepsilon, \rho) = 0 \text{ at } x^h_k, \quad (20.19d)$$

where all Q^h are defined by (20.17)–(20.18) and $B^h(\underline{u}^h)$ is the simplest central approximation to

$$B(\underline{u}) := \mu \sum_{i,j} (\partial_i u_j + \partial_j u_i)\partial_j u_i + \lambda \left(\sum_i \partial_i u_i \right)^2 . \quad (20.20)$$

The exact form of B^h is not important, since it is neither a principal nor a subprincipal term. $\underline{f}^h := (f^h_1, \dots, f^h_d, f_\rho, f_\varepsilon)$ are some local averages of $\underline{f}$; injection $\underline{f}^h = \underline{f}$ is usually used, except in some cases where this fails to give good enough approximations on the finest grid (relative to the grid-$2h$ solution, whose right-hand side is $\underline{f}^{2h} = I^{2h}_h \underline{f}^h$).

The scheme (20.19) will be used in the relaxation processes. The same scheme, but without the artificial viscosity terms ($\alpha^h_j = \alpha$ in (20.17)), will be used to calculate residuals transferred to coarse grids, thus making the overall approximation $O(h^2)$.

In the inviscid case ($\lambda, \mu, \kappa \ll \rho h|\underline{u}|$) the term with $\overline{\lambda}$ in (20.19a) and the term $B^h(\underline{u}^h)$ in (20.19c) may be omitted. They do not contribute to the h-ellipticity of the system. The resulting scheme is nothing but Euler equations with a simple form of artificial viscosity, derived from the viscous (Navier-Stokes) equations.

20.2.3 Simplified boundary conditions

At this stage of development, to separate away various algorithmic questions (cf. §4), the numerical experiments were conducted with known smooth solutions U, employing first periodic boundary conditions and then Dirichlet conditions in two dimensions ($d = 2$). In the periodic case no boundaries are actually present; gridpoint (x_1, x_2) is simply identified with gridpoint $(x_1 + 2\pi, x_2 + 2\pi)$. This enables us later to check that no slow-down is caused by boundary conditions. The Dirichlet case at this stage is the square domain $\{|x|, |y| \le \pi\}$, with $\underline{u}$, ρ and ε given on its boundary. The staggered grid is square and is placed so that the bounary of the domain coincides with cell boundaries; i.e., $h_1 = h_2 = h = 2\pi/N$ and $x^h_0 = (h/2, h/2)$ (see (20.14)).

Moreover, to simplify the code development, the Dirichlet boundary conditions are at this stage placed not exactly on the boundary but in their natural staggered-grid positions. For example, ρ and ε are specified at the cell centers immediately outside the boundary, i.e., on the lines $\{|x|, |y| \leq 2\pi + h/2\}$. This is easy to do at this stage since the numerical tests are made with known solutions U, whose values are in particular known on that staggered boundary. Ultimately, these staggered-boundary conditions will be obtained by (quadratic) extrapolation from the real boundary conditions as well as interior values; the present type of conditions is only employed in order to separate away questions as to exactly when and how this extrapolation should be made.

In the inviscid case some of the above conditions are redundant, but the code can handle this automatically (cf. §20.1.4).

20.3 Distributive relaxation for the simplified system

20.3.1 General approach to relaxation design

Since the problem at hand is not elliptic, one should not attempt obtaining "perfect smoothers" (see §3.3, §7 and end of §4.1). So the question is how to use the usual measure of relaxation performance, namely, the smoothing factor $\overline{\mu}$, in selecting the relaxation scheme. We do it by dividing the relaxation *design* into the following three stages.

(A) First, a *pointwise* (not-block) and direction-free relaxation, with low (per operation, of course, and with the other considerations of §3.2) is constructed for the *uniformly h-elliptic* operator, denoted L_e^h, obtained from L^h when sufficiently large and *isotropic* artificial viscosity terms are used. This means replacing (20.18) by

$$\alpha_j^h = \max\{\alpha, \beta\rho^h \max_l h_l |u_l^h|\}, \tag{20.21}$$

and choosing β just large enough to make excellent $\overline{\mu}$ obtainable independently of relaxation marching directions. It means that β should be appreciably larger than the minimal value $\beta = .5$ needed for stability of simple relaxation schemes. For example one can take for this purpose $\beta = 1$; any larger value of β will not substantially change the smoothing factors, and will especially not change the comparison between different relaxation schemes.

(B) Having designed the relaxation scheme, it is then actually used with the *anisotropic* artificial viscosities (20.18), rather than (20.20). Note that if the flow is not (nearly) aligned with the grid, there is no fundamental difference between the two. If the flow *is* aligned with the grid, there is one kind of high-frequency error components V which

are not deflated in the anisotropic as in the isotropic case because L^hV is much more closely singular than L_e^hV: These are the "characteristic components", i.e. high-frequency components which are smooth in the flow direction. When the flow is not aligned with the grid, neither scheme approximate these components well. If there is alignment, only the anisotropic scheme approximates them well, but exactly for this scheme and these components the pointwise relaxation is not effective. (The effectiveness of relaxation in the isotropic case is obtained for a characteristic component at the price of not really approximating its amplitude in the *differential* solution, which makes the fast convergence for this component meaningless; cf. §12). It is *meaningful* to get good smoothing for the characteristic components only if the alignment is *strong* (i.e., intended and consistent; see §3.3), and exactly then it is *possible* to do that, via block relaxation.

Thus, *the pointwise relaxation scheme, designed in terms of the isotropic artificial viscosities (20.21), is actually used with the anisotropic viscosities (20.18), and it is modified to the corresponding block scheme in the case of strong alignment.* "Corresponding" means that the same distribution matrix M^h, and a similar relaxation ordering, are maintained while the "ghost functions" w_i (see §3.7) are relaxed in blocks instead of pointwise, where the blocks are in the specific strong-alignment direction.

(C) To this basic relaxation scheme, several improvements can be added. First, the value of β can be lowered, either experimentally, or theoretically through the modified smoothing range (12.1). (As explained there (following (12.1)), if β is lowered to near its minimal stable value ($\beta = .5$) , the final result may be improved by averaging. Also note that near $\beta = .5$, downstream relaxation ordering may become badly divergent for some special *smooth* component which may not show up in one experiment but badly affect another; see §20.3.4). Then the equations may more precisely be rotated and upwinded near strong shocks, double discretization schemes may be introduced (see §10.2), etc.

In the two latter stages, (B) and (C), efficiency should of course mainly be measured not in terms of asymptotic factors, but in terms of FMG performance, whether experimentally (see §1.6) or theoretically (see §7.4, 7.5).

20.3.2 Possible relaxation scheme for inviscid flow

In the inviscid case the principal difference operator is L_r^h (cf. (20.12)), and the usual distribution operator (cf. §3.7 and §19.3) would be

$$M_r^h := \begin{pmatrix} 1 & 0 & \cdots & 0 & 0 & 0 & -\partial_1^h Q_0^h \\ 0 & \ddots & \ddots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \ddots & \ddots & 0 & 0 & 0 & \vdots \\ 0 & \cdots & 0 & 1 & 0 & 0 & -\partial_d^h Q_0^h \\ 0 & \cdots & \cdots & 0 & 1 & 0 & -\rho^2 \Delta^h \\ 0 & \cdots & \cdots & 0 & 0 & 1 & -p\Delta^h \\ 0 & \cdots & \cdots & 0 & 0 & 0 & (Q_0^h)^2 \end{pmatrix}, \tag{20.22a}$$

where the last column is made of the cofactors of the last row in L_r^h, divided by their common factor $(Q_0^h)^2$. Since

$$L_r^h M_r^h = \begin{pmatrix} Q_0^h & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & \ddots & \ddots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \ddots & \ddots & 0 & 0 & 0 & \vdots \\ 0 & \cdots & 0 & Q_0^h & 0 & 0 & 0 \\ \rho^2\partial_1 & \cdots & \cdots & \rho^2\partial_d & Q_0^h & 0 & 0 \\ p\partial_1 & \cdots & \cdots & p\partial_d & 0 & Q_0^h & 0 \\ 0 & \cdots & \cdots & 0 & -p_\rho & -p_\varepsilon & (Q_0^h)^2 - \rho^2 a^2 \Delta^h \end{pmatrix}, \tag{20.22b}$$

the relaxation process is essentially decoupled into relaxing the convection operator Q_0^h and, separately, the potential-flow operator $L_{\text{pot}}^h := (Q_0^h)^2 - \rho^2 a^2 \Delta^h$. The former has been discussed in §19.3; the latter has been studied via numerical experiments with potential flows (see remarks in §21).

For low Mach numbers the potential operator is nicely elliptic and the performance of the solver is essentially the same as in the incompressible limit (see §19, 20.1.5).

In the transonic and supersonic case, L_{pot}^h is not elliptic, so the approach outlined above (§20.3.1) is applied. Taking larger artificial viscosities shows, however, that the present approach is not optimal. This is easy to see by observing the limit case of large artificial viscosities, where $L_{\text{pot}}^h \approx (\Delta^h)^2$, i.e., the relaxed operator is essentially the biharmonic operator, for which Gauss-Seidel smoothing is relatively slow: $\overline{\mu} = .80$. Better smoothing schemes for Δ_h^2 exist (see [Bra77a, §6.2]), but they are more complicated, and not fully effective, too. The best scheme for Δ_h^2 is obtained by writing it as a system of two Poisson equations, each relaxed by red-black Gauss-Seidel, yielding $\overline{\mu}_1 = \overline{\mu}_2 = .25$ (cf. (3.2)), with operation count considerably smaller than for Gauss-Seidel relaxation of Δ_h^2. However, this requires the introduction of an auxiliary function and some special care near boundaries, and is much less convenient in case of the actual op-

erator L^h_{pot}, especially in the present framework of the overall distributive relaxation.

Two strategies to avoid this trouble are (A) Introduce the auxiliary function only during relaxation and use an operator product relaxation scheme [LB04], with additional local relaxation sweeps near boundaries (see §5.7); or (B) Choose another distribution operator M^h, where care is taken not to distribute as far as to create the need to relax the *square* of Q^h_0. Moreover, the new approach, to be described next, is applicable to the general CNS system, whereas (20.22) applies to the inviscid limit only, since it assumes $Q_\mu = Q_\kappa = Q_\nu$.

20.3.3 Distributed collective Gauss-Seidel

In view of the subprincipal operator L^h_s (cf. (20.9)), the distribution operator

$$M^h_s := \begin{pmatrix} 1 & 0 & \cdots & 0 & 0 & 0 & -\partial^h_1 \\ 0 & \ddots & \ddots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \ddots & \ddots & 0 & 0 & 0 & \vdots \\ 0 & \cdots & 0 & 1 & 0 & 0 & -\partial^h_d \\ 0 & \cdots & \cdots & 0 & 1 & 0 & 0 \\ 0 & \cdots & \cdots & 0 & 0 & 1 & 0 \\ \overline{\lambda}\partial^h_1 & \cdots & \cdots & \overline{\lambda}\partial^h_d & 0 & 0 & Q^h_{\mu+\overline{\lambda}} \end{pmatrix} \tag{20.23}$$

yields

$$L^h_s M^h_s := \begin{pmatrix} Q^h_\mu - \overline{\lambda}\partial^h_{11} & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & \ddots & \ddots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \ddots & \ddots & 0 & 0 & 0 & \vdots \\ 0 & \cdots & 0 & Q^h_\mu - \overline{\lambda}\partial^h_{dd} & 0 & 0 & 0 \\ \rho^2\partial_1 & \cdots & \cdots & \rho^2\partial_d & Q^h_\nu & 0 & -\rho^2\Delta^h \\ p\partial_1 & \cdots & \cdots & p\partial_d & 0 & Q^h_\kappa & -p\Delta^h \\ \overline{\lambda}\partial^h_1 & \cdots & \cdots & \overline{\lambda}\partial^h_d & -p_\rho & -p_\varepsilon & Q^h_{\mu+\overline{\lambda}} \end{pmatrix} \tag{20.24}$$

With few operations expended on distribution (relative to those expended on calculating residuals) the relaxation is thus "geographically" decoupled: each of the $d+1$ uniform grids composing our staggered grid (cf. (20.14)-(20.15)) is separately relaxed (in terms of the ghost functions; see §3.7). The relaxation for each i-face-centers grid is a simple Gauss-Seidel with the operator $Q^h_\mu - \overline{\lambda}\partial^h_{ii}$, which actually behaves better (has stronger h-ellipticity) than the convection-diffusion operator Q^h_μ. The relaxation of

the cell-centers grid is a relaxation on the 3×3 system

$$\begin{pmatrix} Q_\nu^h & 0 & -\rho^2\Delta^h \\ 0 & Q_\kappa^h & -p\Delta^h \\ -p_\rho & -p_\varepsilon & Q_{\mu+\overline{\lambda}}^h \end{pmatrix}. \tag{20.25}$$

We could relax this system itself by distributive relaxation. But this would yield in the inviscid case the relaxation of §20.3.2, which we have rejected, and in the viscous case would be worse yet (requiring higher order distribution, because $Q_\nu \neq Q_\kappa \neq Q_{\mu+\overline{\lambda}}$). Instead, we relax (20.25) by collective Gauss-Seidel (CGS), i.e., all the three equations defined at each cell center are relaxed simultaneously. This means a solution of a 3×3 linear system, with two convenient zeros already in it, which can be done in 13 operations, a small number compared with the work of calculating the three residuals at each cell center.

20.3.4 Relaxation ordering and smoothing rates

The relaxation of §20.3.3 is examined through the approach of §20.3.1, i.e., by calculating smoothing factors for the *isotropic* artificial viscosity (20.21). Direction-free robust schemes are sought. For such schemes $\overline{\mu}$ always improves (decreases) with increased physical viscosities $(\mu, \overline{\lambda}, \kappa, \nu)$. (The only case in which $\overline{\mu}$ *increases* with increased viscosities is the special case of large Mach numbers and downstream relaxation, a case which is not direction-free and which is also ruled out below for other reasons.) Hence, examined here is mainly the inviscid case $\mu = \overline{\lambda} = \kappa = \nu = 0$.

By (20.24), the relaxation can be performed as $d+1$ separate passes, d of them with the operator Q_0^h, and one with the system (20.25). Hence $\overline{\mu} = \max\{\overline{\mu}^Q, \overline{\mu}'\}$, where $\overline{\mu}^Q$ is the smoothing factor of relaxing Q_0^h and $\overline{\mu}'$ is the smoothing factor in the collective relaxation of (20.25). Both.depend on the order in which the corresponding passes are made, on the value of β in (20.21), and on the direction of $\underline{u}$, and $\overline{\mu}'$ also depends on the Mach number M. This dependence is shown in Table 20.1. Notice that, except for RB(2), $\overline{\mu} = \overline{\mu}' \geq \overline{\mu}^Q$, and in most cases $\overline{\mu} \approx \overline{\mu}' \approx \overline{\mu}^Q$ (assuming the same ordering is used in all passes). This results from the fact that Q_0^h is a divisor of the determinant of (20.25).

The table shows that it very efficient to use Red-Black (RB) ordering in all passes. First, because of its usual advantage of being vectorizable (cf. §3.6). Secondly, for sufficiently large β (and hence also for viscous flows), RB smoothing rates are superior to all others. Only when β approaches its minimal value (e.g., for $\beta \leq 1$), downstream ordering (e.g., lexicographic ordering in case all $u_i \geq 0$) shows better smoothing rates. But this is not direction-free: it is complicated to maintain downstream ordering in case the flow directions (i.e., the signs of $u_1, \ldots, u_d$) change with location. Also, the components for which RB is not so good at smaller β

are exactly the characteristic components one needs not care about (see §20.3.1).

Moreover, the main advantage of the downstream relaxation is shown as *super-fast smoothing* in the case of small β and large M; indeed, $\overline{\mu} \to 0$ as $\beta \searrow .5$ and $M \to \infty$. But this small $\overline{\mu}$ shows only the behavior of *high*-frequency components. There are *low*-frequency ones for which exactly this relaxation badly diverges: the amplification is about $2^{-\frac{1}{2}}M$ for a component $(\theta_1, \theta_2) \approx 2^{-\frac{1}{2}}M^{-1}(u_1, u_2)/q$. This does not happen when the slower schemes are used, such as RB, or even the downstream ordering with larger β ($\beta \geq .8$, say): For these later schemes some low-frequency components may still diverge, but the divergence is slow and easily corrected by the multigrid coarse-grid corrections.

Table 20.1 also shows *symmetric ordering* (SGS) to yield a very efficient smoother. The bad behavior of *low* frequencies for large M and $\beta \approx .5$ is still theoretically present here, although more weakly. One can eliminate this trouble simply by switching to larger β (e.g., $\beta = 1$) wherever M is large and the relaxation marching happens to be downstream. In fact, there is no reason to use the same β at the same location in all passes, especially as still another value ($\beta = 0$) is used for fine-to-coarse residual transfers.

20.3.5 Summary: relaxation of the full system

The scheme outlined above for the subprincipal part L_s^h easily translates to the following relaxation procedure for the full quasi-linear system (20.19). We denote by $\tilde{U}^h = (\tilde{\underline{u}}^h, \tilde{\rho}^h, \tilde{\varepsilon}^h, \tilde{p}^h)$ the dynamic approximation to the solution $U = (\underline{u}, \rho, \varepsilon, p)$, i.e., the approximation just prior to any described relaxation step, while $U^h = (\underline{u}^h, \rho^h, \varepsilon^h, p^h)$ will denote the exact solution of the stable difference equations (20.19). $r^h = (r_1^h, \ldots, r_d^h, r_\rho^h, r_\varepsilon^h, r_p^h)$ will denote the dynamic residuals, i.e. the left-hand sides of (20.19) applied to $\tilde{U}^h$ instead of U^h and subtracted from the right-hand sides. Thus, at each step of each relaxation pass, $\tilde{U}^h$ and r^h change.

The relaxation steps are first described for the case where there is no strong alignment between the grid and the flow direction.

A relaxation sweep consists of $d+1$ passes. The recommended ordering within each pass is either the red-black ordering (which we used throughout our numerical experiments) or the symmetric lexicographic (SGS).

First, one pass is made for each momentum equation. The *i-th* equation at the point $x_k^{h,i}$ is relaxed by the replacements

$$\tilde{u}_i^h \leftarrow \tilde{u}_i^h + \psi_{i,k}^h \tag{20.26a}$$

$$\tilde{p}^h \leftarrow \tilde{p}^h + \overline{\lambda}\partial_i^h \psi_{i,k}^h, \tag{20.26b}$$

where $\psi_{i,k}^h$ is a function defined at all i-face centers (see (20.15)), vanishing on all of them except at the relaxed point $x_k^{h,i}$. Its value at $x_k^{h,i}$ is determined

Relax. Order	β / M	$(u_1,u_2)=(1,1)$ 2.0	1.0	0.5	$(u_1,u_2)=(0,1)$ 2.0	1.0	0.5
RB(1)	*	.27	.33	1.00	.27	.31	.50
	0.0–0.1	.27	.43	1.00	.27	.31	.50
	1.0–5.0	.40–.41	.46–.48	1.00	.27–.29	.33–.36	.50–.55
RB(2)	*	.39	.50	1.00	.36	.39	.42
	0.0–0.1	.27	.65	1.00	.27	.31	.50
	1.0–5.0	.33–.35	.41–.42	1.00	.27–.29	.33–.39	.50–.55
Lex+	*	.38	.25	0	.46	.42	.47
	0.0–1.0	.44–.50	.42–.50	.42–.50	.48–.50	.47–.50	.49–.50
	2.0	.41	.35	.24	.47	.45	.48
	5.0	.39	.29	.056	.46	.44	.50
Lex−	*	.63	.75	1.00	.58	.66	.72
	0.0–5.0	.63	.75–.76	1.00	.58	.66–.67	.84–.86
Lex±	*	.54	.63	1.00	.58	.66	.72
	0.0–5.0	.54–.55	.63–.64	1.00	.58	.66–.67	.84–.86
SGS	*	.48	.42	0	.50	.49	.49
	0.0–0.1	.50	.50	.50	.50	.50	.50
	1.0	.51	.54	.60	.51	.54	.60
	5.0	.49	.46	.24	.50	.51	.53

Table 20.1. ***Smoothing factors for two-dimensional Euler equations.***

$\overline{\mu} = \max\{\overline{\mu}^Q, \overline{\mu}'\}$. *In the rows where* * *stands for* M, $\overline{\mu}^Q$ *is displayed; in the others,* $\overline{\mu}'$ *is displayed.* RB(i) *is red-black relaxation ordering with* i *sweeps per cycle (*$\overline{\mu}_i$ *derived by (3.2)).* Lex+ *is lexicographic ordering,* Lex− *is reversed lexicographic,* Lex± *is lexicographic with only the* y *coordinate reversed, and SGS is Symmetric GS (a* Lex+ *alternating with a* Lex−, $\overline{\mu}$ *calculated per pass).*

so that

$$\left(Q_\mu^h - \overline{\lambda}\partial_{ii}^h\right)\psi_{i,k}^h(x_k^{h,i}) = r_i^h(x_k^{h,i}), \tag{20.27}$$

where the coefficients of Q_μ^h are evaluated at $x_k^{h,i}$ (see (20.17)), based on $\tilde{U}^h$. Neglecting non-principal effects, this means that after the changes (20.26), the *i-th* momentum equation (20.19a) at $x_k^{h,i}$ will be satisfied.

Then, a pass is made on cell centers. The three equations defined at

the cell center x_k^h are relaxed simultaneously by replacements of the form

$$\tilde{u}_i^h \leftarrow \tilde{u}_i^h - \partial_i^h \Phi_k^h, \qquad (i = 1, \ldots, d) \tag{20.28a}$$

$$\tilde{p}^h \leftarrow \tilde{p}^h + Q^h_{\mu+\overline{\lambda}} \Phi_k^h \tag{20.28b}$$

$$\tilde{\rho}^h \leftarrow \tilde{\rho}^h + \hat{\rho}_k^h \tag{20.28c}$$

$$\tilde{\varepsilon}^h \leftarrow \tilde{\varepsilon}^h + \hat{\varepsilon}_k^h, \tag{20.28d}$$

where Φ_k^h, $\hat{\varepsilon}_k^h$ and $\hat{\rho}_k^h$ are functions defined at cell centers (see (20.14)) vanishing everywhere except at the relaxed point x_k^h. Their values at x_k^h are chosen so as to satisfy the three equations

$$\left(\tilde{Q}_\nu^h \hat{\rho}_k^h - \tilde{\rho}^2 \Delta^h \Phi_k^h\right)(x_k^h) = r_\rho^h(x_k^h) \tag{20.29a}$$

$$\left(\tilde{Q}_\kappa^h \hat{\varepsilon}_k^h - \tilde{p} \Delta^h \Phi_k^h\right)(x_k^h) = r_\varepsilon^h(x_k^h) \tag{20.29b}$$

$$\left(-\tilde{p}_\rho \hat{\rho}_k^h - \tilde{p}_\varepsilon \hat{\varepsilon}_k^h + \tilde{Q}^h_{\mu+\overline{\lambda}} \Phi_k^h\right)(x_k^h) = r_p^h(x_k^h). \tag{20.29c}$$

where $\tilde{\rho}$, $\tilde{p}$, $\tilde{p}_\rho$, $\tilde{p}_\varepsilon$ and the coefficients of $\tilde{Q}_\nu^h$, $\tilde{Q}_\kappa^h$ and $\tilde{Q}^h_{\mu+\overline{\lambda}}$ are all evaluated at x_k^h, based on $\tilde{U}^h$.

The functions $\psi_{i,k}^h$, Φ_k^h, $\hat{\rho}_k^h$ and $\hat{\varepsilon}_k^h$ mentioned above do not actually appear in the program, of course. They just serve to concisely describe the relaxation steps.

Instead of the separate $d+1$ passes just described, they could of course be merged in any desired fashion.

In the case of strong alignment, i.e., if one grid direction nearly coincides with the flow direction throughout a substantial subdomain, relaxation should be done in the corresponding blocks (lines in the alignment direction). This means that each unknown function (ψ^h in the case of (20.26), and Φ^h, $\hat{\rho}^h$ and $\hat{\varepsilon}^h$, in the case of (20.28)) are free (i.e., not fixed to be zero) not just at one gridpoint at a time, but at all gridpoints of the relaxed block, thus giving exactly the number of parameters needed to simultaneously satisfy the equations (20.27) or (20.29)) at all gridpoints of the block. In the case of *plane alignment* (cf. §19.2) it is advisable to coarsen the grid only in that plane directions (see §4.2.1), in which case no block relaxation will be needed.

The total work of the relaxation sweep is only a fraction (20% or so) larger than the work of *expressing* the differences equations (20.19), or calculating their residuals, at all gridpoints.

20.4 Multigrid procedures

The grids, their relative positions and the interpolation procedures between them are generally as for Stokes equations (§18.4), with ρ and ε transferred similarly to p, and the residuals of the energy and state equations transferred similarly to the residuals of the continuity equations. Because of the

nonlinearity, FAS is of course used (see §8), and the full weighting (18.14) is preferred over (18.13) in the fine-to-coarse transfers of both the velocity u_i and the *i-th* momentum residual.

An option for *double discretization* (cf. §10.2) is included. Namely, the artificial viscosity terms may be omitted in calculating the residuals to be transferred from any grid to the next coarser one. More generally, the artificial-viscosity coefficient β (see (20.18)), may have different values at different stages of the algorithm.

Whatever the value of β, in any stage of the solver, the discretization should also attempt not to straddle strong discontinuities, by calculating one-sided fluxes.

Note that in FAS, Dirichlet boundary conditions appear the same on all grids, whether the grid is the currently-finest or a correction grid. This can also be the case in the simplified, staggered boundary conditions (see §20.2.3): The exact differential solution is enforced at the staggered boundary points of the coarser, correction grid H, too. On those points, the value of $\hat{I}_h^H \tilde{U}^h$ is also defined to be the exact solution, hence, for the purpose of interpolating back to the finer grid (like (8.6)), the difference $\tilde{U}^H - \hat{I}_h^H \tilde{U}^h$ is defined to be zero at the staggered boundary points.

As in other non-elliptic cases, W cycles are generally preferred to V cycles (see §6.2).

Chapter 21

Remarks On Solvers For Transonic Potential Equations

21.1 Multigrid improvements

The multigrid solution of transonic potential flows was first studied in collaboration with Jerry South [SB77], [Bra77a, §6.5]. At the time of that study multigrid research was less advanced, and many of the improved approaches described in this Guide were not implemented. Collected below is a list of important improvements that the present Guide would recommend.

(A) The Neumann boundary condition and the constant-potential jump condition in the wake of an airfoil should not be *enforced* in relaxation, only *smoothed* (see §5.3). Thus, in relaxation, the potential jump at each wake point should be just set to be the average of the jumps at the neighboring wake points. The conditions should only be enforced at the coarsest level. Likewise, Kutta condition should only be applied at the coarsest level (cf. §5.6): the far-field conditions on each level should accordingly be adjusted at the stage of interpolating-and-adding the coarse-grid correction.

(B) Improved rates can be obtained if, before each full relaxation sweep, special local sweeps are made around singular points, such as trailing edges (see §5.7).

(C) Near a strong shock it may be better to use interpolation procedures which take jumps in p into account (see §4.6).

(D) Instead of stretching coordinates (to cover large exterior domains) and other transformations, a better multigrid procedure is to use increasingly coarser grids on increasingly larger domains, possibly with local refinements, anisotropic refinements and local coordinates (see §9). On such grids, simple relaxation schemes can be use: block relaxation is only needed in directions of strong alignment (see §3.3).

(E) Most importantly, because of the non-elliptic nature of the problem, perfect smoothers and good *asymptotic* convergence rates should not be attempted: much simpler and vectorizable schemes can be used if all one attempts is to get below truncation errors (see end of §4.1 and 3.3, 7 and 20.3.1). Correspondingly, the performance of the algorithm should be ascertained through direct measurements of $\|\tilde{\phi}^h - \tilde{\phi}^{2h}\|$ (see §1.6) and of $|F^h(\tilde{\phi}^h) - F^{2h}(\tilde{\phi}^{2h})|$, where $F(\Phi)$ is any solution functional one wants to get as the end result of the computations. If the norm measures discontinuous quantities, such as velocities, it should be an L_1 norm [Bra81a, §4.5].

21.2 Artificial viscosity in quasi-linear formulations

The transonic potential equation can be written in the quasilinear form

$$\left[(\underline{u} \cdot \underline{\nabla})^2 - a^2 \Delta\right] \Phi = 0, \tag{21.1}$$

where $\underline{u} = \underline{\nabla}\Phi$. This operator has appeared above in the discussion of Euler and Navier-Stokes equations (see for instance (20.13), or (20.22)). This physical origin of the operator suggests a physical form for the artificial viscosity which should be added to it, different from the Murman-Cole-type forms. Namely, the artificial viscosity should be added to $\underline{u} \cdot \underline{\nabla}$, before it is squared, using generally the form of Q_0^h (see (20.17)–(20.18) for $\alpha = 0$). In particular, if upwinding is desired, it is the operator $\underline{u} \cdot \underline{\nabla}$ that should be upwinded, before it is squared.

This form of artificial viscosity (or upwinding) is not only smoother and more physical, it is also more straightforward than the usual scheme where the operator should be rotated before it is upwinded. Also, this scheme requires no distinction between subsonic and supersonic points. The main difference between the two schemes is near sonic points ($M \approx 1$), or near shock transition from supersonic to subsonic. The Murman-Cole artificial viscosity vanishes there, and may therefore give rise to non-physical solutions.

In deeply subsonic regions, where (21.1) is uniformly elliptic, the form of the artificial viscosity does not matter of course, and one can switch to fully central approximations. In multigrid processing it is not important to do that, because $O(h^2)$ approximations can be obtained, even in the supersonic regions, by omitting the artificial viscosity (or the upwinding) from the operator used in the fine-to-coarse residuals transfer (see §10.2).

This latter operator should notwithstanding respect shocks as far as possible. Namely, it should be written in conservation form, and the stronger the shock, the weaker should it be straddled by flux calculations.

Appendix A

TestCycle: Matlab Code

The following MATLAB R2010a code solves the Poisson equation $\Delta u = F(x, y)$ with Dirichlet boundary conditions $u = G(x, y)$ on a rectangle by applying $V(\nu_1, \nu_2)$ cycles to a random initial guess. Place all files in the same directory and run the MATLAB command `TestCycle.run` (see §1.5).

A.1 addflops.m

```
function addflops(fl)
%ADDFLOPS   Increment the global flopcount variable. ADDFLOPS(fl) is
%equivalent to FLOPS(FLOPS+FL), but more efficient.
global flopcount;
if ~isempty(flopcount)
    flopcount = flopcount + fl;
end
```

A.2 BilinearInterpolation.m

```
classdef BilinearInterpolator
    %BILINEARINTERPOLATOR Bi-linear interpolation of corrections.
    %   Executes a second-order interpolation from level L to L+1.

    %====================== METHODS ================================
    methods
        function u = interpolate(obj, coarseLevel, fineLevel, uc)
            % Interpolate the coarse-level function UC at level
            % COARSELEVEL to the function U at level FINELEVEL.

            % Interpolate along one dimension at a time
            u1  = obj.interpInX(coarseLevel, fineLevel, uc);
            u   = obj.interpInY(coarseLevel, fineLevel, u1);
```

```
        end
    end

    %====================== PRIVATE METHODS ======================
    methods (Access = private)
        function u = interpInX(obj, coarseLevel, fineLevel, uc)
            % Linear interpolation in x

            % Aliases, allocate output array
            nf  = fineLevel.n;
            nc  = coarseLevel.n;
            u   = zeros(nf(1),nc(2));

            % Inject coarse points into the respective fine points
            u(1:2:nf(1),:) = uc;

            % Linearly interpolate into in-between fine-level points
            for i1 = 1:nc(1)-1
                u(2*i1,:) = 0.5*(uc(i1,:) + uc(i1+1,:));
            end
        end

        function u = interpInY(obj, coarseLevel, fineLevel, uc)
            % Linear interpolation in y

            % Aliases, allocate output array
            nf  = fineLevel.n;
            nc  = coarseLevel.n;
            u   = zeros(nf);

            % Inject coarse points into the respective fine points
            u(:,1:2:nf(2)) = uc;

            % Linearly interpolate into in-between fine-level points
            for i2 = 1:nc(2)-1
                u(:,2*i2) = 0.5*(uc(:,i2) + uc(:,i2+1));
            end
        end

    end

end
```

A.3 Cycle.m

```
classdef (Sealed) Cycle < handle
    %CYCLE Multigrid cycle.
    %   This class holds the entire multi-level data structure
    %   and executes multigrid cycles. A cycling strategy with an
```

```
%   integer index is implemented (gamma=1: V-cycle; gamma=2:
%   W-cycle).

%====================== MEMBERS ==============================
properties (GetAccess = private, SetAccess = private)
    levels              % Level list (1=finest, end=coarsest)
    options             % Contains cycle parameters
    finestRelaxWork     % Finest-level relaxation sweep cost
end

%====================== CONSTRUCTORS =========================
methods
    function obj = Cycle(options, levels)
        % Create a cycle executor with options OPTIONS, to
        % act on the level list LEVELS.
        obj.options         = options;
        obj.levels          = levels;
        obj.finestRelaxWork = prod(levels{end}.n-1);
    end
end

%====================== METHODS ==============================
methods
    function u = cycle(obj, finest, u)
        % The main call that executes a cycle at level FINEST.
        obj.printErrorNormHeader();
        u = obj.cycleAtLevel(finest, finest, u);
    end
end

%====================== PRIVATE METHODS ======================
methods (Access = private)
    function u = cycleAtLevel(obj, l, finest, u)
        % Execute a cycle at level L. FINEST is the index of
        % the finest level in the cycle. Recursively calls
        % itself with the next-coarser level until NUM_LEVELS
        % is reached.

        obj.printErrorNorm(l, 'Initial', u);
        if (l == max(1, finest-obj.options.maxCycleLevels+1))
            % Coarsest level
            u = obj.relax(l, obj.options.numCoarsestSweeps,...
                u, false);
        else
            %--- Pre-relaxation ---
            u = obj.relax(l, obj.options.numPreSweeps, u,...
                true);

            %--- Coarse-grid correction ---
```

```
            c               = l-1;
            fineLevel       = obj.levels{l};
            coarseLevel     = obj.levels{c};

            % Transfer fine-level residuals
            r               = fineLevel.residual(u);
            coarseLevel.f   = fineLevel.restrict(r);

            % Solve residual equation at coarse level
            % Correction scheme: start from vc=0
            vc = zeros(coarseLevel.n);
            for i = 1:obj.options.cycleIndex
                vc = obj.cycleAtLevel(c, finest, vc);
            end

            % Interpolate coarse-level correction and add it
            v               = fineLevel.interpolate(vc);
            u               = u + v;
            obj.printErrorNorm(l, 'Coarse-grid correction', u);

            %--- Post-relaxation ---
            u = obj.relax(l, obj.options.numPostSweeps, u, true);
        end
    end

    function u = relax(obj, l, nu, u, printEverySweep)
        % Perform NU relaxation sweeps on U at level LEVEL. If
        % PRINTEVERYSWEEP is true, prints printouts after every
        % sweep; otherwise, only after the last sweep.
        for i = 1:nu
            u = obj.levels{l}.relax(u);
            if (printEverySweep)
                obj.printErrorNorm(l, ...
                sprintf('Relaxation sweep %d', i), u);
            end
        end
        if (~printEverySweep)
            obj.printErrorNorm(l, ...
            sprintf('Relaxation sweep %d', i), u);
        end
    end

    function printErrorNormHeader(obj)
        % Print a header line for cycle debugging printouts.
        if (obj.options.logLevel >= 1)
            fprintf('%-5s %-25s %-13s %-9s\n', 'LEVEL', ...
               'ACTION', 'ERROR NORM', 'WORK');
        end
    end
```

```
        function u = printErrorNorm(obj, l, action, u)
            % A debugging printout of the error norm at level L
            % after a certain action has been applied. The work per
            % finest-level relaxation sweep is also printed.
            if (obj.options.logLevel >= 1)
                fprintf('%-5d %-25s %.3e   %6.2f\n', l, action, ...
                    errornorm(obj.levels{l}, u), ...
                    flops/obj.finestRelaxWork);
            end
        end
    end
end
```

A.4 errornorm.m

```
function e = errornorm(level, u)
%ERROR_NORM Error norm at a certain coarsening level.
%   E = ERROR_NORM(LEVEL,U) computes the grid-scale L2 residual norm
%   |F-L(U)|_2, where F and L are stored in the LEVEL structure.
r = level.residual(u);
e = norm(r(:))/sqrt(numel(r));
```

A.5 flops.m

```
function f = flops(fl)
%FLOPS  Get or set the global flopcount variable.
% FLOPS returns the current flopcount. FLOPS(F) sets flopcount to F.

global flopcount;
if nargin == 1
    flopcount = fl;
    if nargout == 1 f = fl; end
else
    f = flopcount;
end
```

A.6 FwLinearRestrictor.m

```
classdef FwLinearRestrictor
    %FULLWEIGHTINGRESTRICTOR 2nd-order full weighting of residuals.
    %   Executes a second-order full weighting from level L+1 to L.

    %====================== METHODS ================================
    methods
        function fc = restrict(obj, coarseLevel, fineLevel, f)
            % Restrict the fine-level function F at level FINELEVEL
            % to level COARSELEVEL.
```

```
            % Interpolate along one dimension at a time
            f1  = obj.restrictInX(coarseLevel, fineLevel, f);
            fc  = obj.restrictInY(coarseLevel, fineLevel, f1);
        end
    end

    %===================== PRIVATE METHODS ======================
    methods (Access = private)
        function fc = restrictInX(obj, coarseLevel, fineLevel, f)
            % Full-weighting in x
            % Aliases, allocate output array
            nf = fineLevel.n;
            nc = coarseLevel.n;
            fc = zeros(nc(1),nf(2));

            % Full-weighting of boundary residuals
            fc([1 nc(1)],:) = f([1 nf(1)],:);

            % Full-weighting of interior residuals
            for i1 = 2:nc(1)-1
                fc(i1,:) = 0.25*(f(2*i1-2,:) + ...
                    2*f(2*i1-1,:) + f(2*i1,:));
            end
        end
        function fc = restrictInY(obj, coarseLevel, fineLevel, f)
            % Full-weighting in y

            % Aliases, allocate output array
            nf = fineLevel.n;
            nc = coarseLevel.n;
            fc = zeros(nc);

            % Full-weighting of boundary residuals
            fc(:,[1 nc(2)]) = f(:,[1 nf(2)]);

            % Full-weighting of interior residuals
            for i2 = 2:nc(2)-1
                fc(:,i2) = 0.25*(f(:,2*i2-2) + ...
                2*f(:,2*i2-1) + f(:,2*i2));
            end
        end
    end
end
```

A.7 GaussSeidelSmoother.m

```
classdef (Sealed) GaussSeidelSmoother < handle
    %GAUSSSEIDELSMOOTHER Gauss-Seidel relaxation scheme.
    %   This class executes Gauss-Seidel relaxation sweeps in
```

```
    %   lexicographic order. It can be applied at any level.
    %==================== MEMBERS ============================
    properties (GetAccess = private, SetAccess = private)
        numColors       % Number of colors (1=LEX, 2=RB)
    end

    %==================== CONSTRUCTORS =======================
    methods
        function obj = GaussSeidelSmoother(numColors)
            obj.numColors = numColors;
        end
    end

    %==================== METHODS ============================
    methods
        function u = relax(obj, level, u)
            % Gauss-Seidel successive displacement in lexico-
            % graphic ordering. Because MATLAB passes array
            % parameters by value, this does not override the
            % original U array.

            %Useful aliases
            h2  = level.h^2;
            f   = level.f;

            % Impose B.C.
            i1 = [1 level.n(1)];        u(i1,:) = f(i1,:);
            i2 = [1 level.n(2)];        u(:,i2) = f(:,i2);

            % Relax in the internal domain
            for c = 0:obj.numColors-1
                for i1 = 2:level.n(1)-1
                    for i2 = 2:level.n(2)-1
                        if (mod(i1+i2, obj.numColors) == c)
                            u(i1,i2) = 0.25*(h2*f(i1,i2) ...
                                + u(i1  ,i2-1) + u(i1  ,i2+1) ...
                                + u(i1-1,i2  ) + u(i1+1,i2  ));
                        end
                    end
                end
            end

            % A relaxation sweep is counted as one flop per
            % internal gridpoint
            addflops(prod(level.n-1));
        end
    end
end
```

A.8 Level.m

```
classdef (Sealed) Level < handle
    %LEVEL A single level in the multi-level cycle.
    %   This class holds all data and operations pertinent to a
    %   single level in the multi-level cycle: right-hand-side,
    %   residual computation and single-level processes such as
    %   relaxation.
    %====================== MEMBERS ==============================
    properties (GetAccess = public, SetAccess = public)
        f               % RHS of both the interior equations & B.C.
    end

    properties (GetAccess = public, SetAccess = private)
        domainSize      % Size of domain
        h               % Mesh-size (same in all directions)
        n               % Grid array size vector
    end
    properties (GetAccess = private, SetAccess = private)
        coarseLevel     % Next-coarser level
        operator        % Computes discrete operator @ this level
        smoother        % Relaxation scheme
        interpolator    % Interpolates corrections from fineLevel
        restrictor      % Restricts residuals to fineLevel
    end

    %====================== CONSTRUCTORS =========================
    methods (Access = private)
        function obj = Level(domainSize, n,  operator, smoother,...
                coarseLevel, interpolator, restrictor)
            % Initialize a level.

            obj.domainSize      = domainSize;
            obj.n               = n+1;
            hVector             = domainSize./n;
            if (std(hVector) > eps)
                error('Incompatible domain size [%f,%f] ...
                and #intervals [%d,%d]: meshsize must be the ...
                    same in all directions', domainSize(1), ...
                    domainSize(2), n(1), n(2));
            end
            obj.h               = hVector(1);
            obj.f               = zeros(obj.n);
            obj.operator        = operator(obj);
            obj.smoother        = smoother;
            obj.coarseLevel     = coarseLevel;
            obj.interpolator    = interpolator;
            obj.restrictor      = restrictor;
        end
```

```
end
methods (Static)
    function obj = newLevel(domainSize, n,  operator, ...
            smoother, coarseLevel, interpolator, restrictor)
        % A factory method of the next-finer level over
        % COARSELEVEL, with an NxN grid of a domain of size
        % DOMAINSIZExDOMAINSIZE, discrete operator OPERATOR a
        % relaxation scheme SMOOTHER and inter-grid transfers
        % INTERPOLATOR and RESTRICTOR. The right-hand-side is
        % initialized to zero.
        obj = Level(domainSize, n,  operator, smoother, ...
            coarseLevel, interpolator, restrictor);
    end

    function obj = newCoarsestLevel(domainSize, n, operator,...
        smoother)
        % A factory method of the coarsest level, with an NxN
        % grid of a domain of size DOMAINSIZExDOMAINSIZE, a
        % discrete operator OPERATOR and a relaxation scheme
        % SMOOTHER.
        obj = Level(domainSize,n,operator,smoother,[],[],[]);
    end
end

%==================== METHODS ==============================
methods
    function r = residual(obj, u)
        % Compute the residual F-L(U) for a function U at this
        % level.
        r = obj.f - obj.L(u);
    end

    function v = relax(obj, u)
        % Perform a relaxation sweep. Delegates to the smoother
        % with a call-back to this level.
        v = obj.smoother.relax(obj, u);
    end

    function u = interpolate(obj, uc)
        % Interpolate the correction uc from the next-coarser
        % level.
        u = obj.interpolator.interpolate(obj.coarseLevel,obj, uc);
    end

    function fc = restrict(obj, f)
        % Restrict the residual FC to the next-coarser level.
        fc = obj.restrictor.restrict(obj.coarseLevel, obj, f);
    end
```

```
        function [x, y] = location(obj, i1, i2)
            % Return gridpoint locations at indices (I1,I2).
            x = obj.h*(i1-1);
            y = obj.h*(i2-1);
        end

        function result = L(obj, u)
            % Apply the discrete operator L to a function U.
            result = obj.operator.L(u);
        end

        function handle = plot(obj, u)
            % Plot the discrete function U on the grid of this level.
            [x,y]   = obj.location(1:obj.n(1), 1:obj.n(2));
            [X,Y]   = ndgrid(x,y);
            handle  = surf(X,Y,u);
        end
    end

end
```

A.9 MultilevelBuilder.m

```
classdef (Sealed) MultilevelBuilder < handle
    %MULTILEVELBUILDER Constructs the multi-level data structure.
    %   This class builds a list of increasingly-finer levels to be
    %   used in the multigrid cycle.

    %====================== CONSTRUCTORS ========================
    methods
        function obj = MultilevelBuilder
            % Explicit constructor is required for a handle class.
        end
    end

    %======================= METHODS ============================
    methods
        function levels = build(obj, options) %#ok<MANU>
            % Build the list of levels from options.
            levels = cell(options.numLevels, 1);

            % Coarsest level
            n           = options.nCoarsest;
            levels{1}   = Level.newCoarsestLevel(options.domainSize,...
                n, options.operator, options.smoother);

            % Increasingly-finer levels
            for l = 2:options.numLevels
                n = 2*n;
```

```
                lev = Level.newLevel(options.domainSize, n, ...
                    options.operator, options.smoother, ...
                    levels{l-1}, options.interpolator, ...
                    options.restrictor);

                % Initialize finest right-hand side
                if (l == options.numLevels)
                    % Interior RHS
                    MultilevelBuilder.setRhsValues(...
                        lev, 2:n(1)-1, 2:n(2)-1, options.f);

                    % Boundary RHS
                    MultilevelBuilder.setRhsValues(...
                        lev, [1 n(1)], 1:n(2)   , options.g);
                    MultilevelBuilder.setRhsValues(...
                        lev, 1:n(1)   , [1 n(2)], options.g);
                end

                levels{l} = lev;
            end
        end
    end

    %==================== PRIVATE METHODS ====================
    methods (Static, Access = private)
        function setRhsValues(lev, i1, i2, f)
            % Set the values of indices (i1,i2) of a level's RHS
            % vector to the function f, evaluated at the corres-
            % ponding gridpoint locations.
            [xInterior,yInterior]   = lev.location(i1,i2);
            % Convert singleton x,y vectors to 2-D matrices
            [X,Y]                   = ndgrid(xInterior, yInterior);
            lev.f(i1,i2)            = f(X,Y);
        end
    end
end
```

A.10 Operator.m

```
classdef (Sealed) Operator < handle
    %OPERATOR Discrete operator computer.
    %   This class computes the discrete operator L(U) of a function
    %   U at a certain level in the multi-level algorithm.

    %==================== MEMBERS ============================
    properties (GetAccess = private, SetAccess = private)
        level           % Holds convenient level variables
    end
```

```
    %==================== CONSTRUCTORS ========================
    methods
        function obj = Operator(level)
            % Initializes an operator computer at level LEVEL.
            obj.level = level;
        end
    end

    %===================== METHODS ============================
    methods
        function result = L(obj, u)
            % Apply the discrete operator L to a function U. This
            % is the 5-point Laplacian with Dirichlet boundary
            % conditions.

            % Allocate output array
            result = zeros(obj.level.n);

            % Set Dirichlet boundary conditions
            i1 = [1 obj.level.n(1)];        result(i1,:) = u(i1,:);
            i2 = [1 obj.level.n(2)];        result(:,i2) = u(:,i2);

            % 5-point discrete Laplacian in the interior domain
            rh2 = 1/obj.level.h^2;
            for i1 = 2:obj.level.n(1)-1
                for i2 = 2:obj.level.n(2)-1
                    result(i1,i2) = rh2*(...
                        4*u(i1,i2) ...
                        -u(i1  ,i2-1) - u(i1  ,i2+1) ...
                        -u(i1-1,i2  ) - u(i1+1,i2  ));
                end
            end
        end

    end

end
```

A.11 Options.m

```
classdef (Sealed) Options < handle
    %OPTIONS Multi-level algorithm options.
    %   Includes both model parameters and cycle parameters. Sets
    %   default values for parameters that can be overriden by the
    %   user.

    %==================== MEMBERS ============================
    properties
        % Model parameters
```

```
        domainSize = [2.0 3.0]                  % Domain size

        f = @(x,y)(sin(x.^2+y)+0.5)             % Right-hand-side
        g = @(x,y)(cos(2*x+y)+0.5)              % Dirichlet B.C.

        % Known solution u = (2*pi^(-2))*sin(pi*x).*sin(pi*y)
        %f = @(x,y)(sin(pi*x).*sin(pi*y))       % Right-hand-side
        %g = @(x,y)(sin(pi*x).*sin(pi*y))       % Dirichlet B.C.

        % To debug the cycle error, set f=g=0 so that u=error
        %f = @(x,y)(zeros(size(x)))             % Right-hand-side
        %g = @(x,y)(zeros(size(x)))             % Dirichlet B.C.

        % Discretization
        nCoarsest = [2 3]                % #coarsest grid intervals
        numLevels = 6                           % Total #levels
        operator  = @(level)(Operator(level)) % Discrete operator

        % Relaxation parameters
        smoother = GaussSeidelSmoother(1)       % Gauss-Seidel relaxation
                                                %(1=LEX, 2=RB)

        % Inter-grid transfers
        interpolator = BilinearInterpolator % Interp. of corrections
        restrictor = FwLinearRestrictor;    % Residual transfer

        % Cycle parameters
        maxCycleLevels = 100                    % # levels in the cycle
        cycleIndex = 1                          % V-cycle/W-cycle/etc.
        numCoarsestSweeps = 5                   % # relaxation sweeps
                                                % at coarsest level
        numPreSweeps = 2                        % # pre-CGC relax sweeps
        numPostSweeps = 1                       % # post-CGC relax sweeps

        % Multi-grid run
        numCycles = 12                          % #cycles to run

        % Miscellaneous
        logLevel = 1                            % Cycle logging level
    end

end
```

A.12 TestCycle.m

```
classdef TestCycle
    %TESTCYCLE Test the multigrid cycle for the 2D Poisson equation.
    %   This class iteratively runs multigrid V-cycles and measures
    %   their convergence factor.
```

```
    %
    %   See also: ERROR_NORM, CYCLE.

    %===================== METHODS ================================
    methods
        function [u, finestLevel] = run(obj) %#ok<MANU>
            % Initialize objects
            flops(0);       % Reset flop count
            options     = Options;
            levels      = MultilevelBuilder().build(options);
            cycle       = Cycle(options, levels);
            finest      = length(levels);
            finestLevel = levels{finest};

            % Initial guess
            u           = rand(finestLevel.n);
            eNew        = errornorm(finestLevel, u);
            % Run cycles
            for numCycle = 1:options.numCycles
                % Print debugging lines only for the first few cycles
                if (numCycle <= 3)
                    options.logLevel = 1;
                    fprintf('########## CYCLE #%d ##########\n',...
                        numCycle);
                else
                    options.logLevel = 0;
                end
                eOld    = eNew;
                u       = cycle.cycle(finest, u);
                eNew    = errornorm(finestLevel, u);
                fprintf('CYCLE %#2d   CONVERGENCE FACTOR = %.3f\n', ...
                    numCycle, eNew/eOld);
            end

        end
    end

end
```

Bibliography

[ABDP81] R. E. Alcouffe, A. Brandt, J. E. Dendy, and J. W. Painter. The multi–grid methods for the diffusion equation with strongly discontinuous coefficients. *SIAM J. Sci. Stat. Comput.*, 2:430–454, 1981.

[Ara66] A. Arakawa. Computational design for long-term numerical integration of the equations of fluid motion: Two-dimensional incompressible flow. part i. *J. Comput. Phys.*, 1:119–143, 1966.

[Arl78] B. Arlinger. Multigrid techniques applied to lifting transonic flow using full potential equation. Technical report, SAAB-SCANIA, 1978.

[AS82] W. Auzinger and H. J. Stetter. Defect correction and multigrid iterations. In W. Hackbusch and U. Trottenberg, editors, *Multigrid Methods*, volume 960 of *Lecture Notes in Mathematics*, pages 327–351, Berlin, 1982. Springer-Verlag.

[B+78] A. Brandt et al. *Lecture notes of the ICASE Workshop on Multigrid Methods.* ICASE, NASA Langley Research Center, Hampton, VA, 1978.

[Bak66a] N. S. Bakhvalov. On the convergence of a relaxation method under natural constraints on an elliptic operator. *Z. Vycisl. Mat. i. Mat. Fiz.*, 6:861–883, 1966.

[Bak66b] N. S. Bakhvalov. On the convergence of a relaxation method with natural constraints on the elliptic operator. *USSR Comp. Math. Math. Phis*, 6:101–113, 1966.

[Ban81] R. E. Bank. A multi–level iterative method for nonlinear elliptic equations. In M. H. Schultz, editor, *Elliptic Problem Solvers*, pages 1–16. Academic Press, New York, 1981.

[BB83] D. Barkai and A. Brandt. Vectorized multigrid Poisson solver for the CDC Cyber 205. *Appl. Math. Comput.*, 13:215–228, 1983.

[BB87] D. Bai and A. Brandt. Local mesh refinement multilevel techniques. *SIAM J. Sci. Stat. Comput.*, 8:109–134, 1987.

[BC83] A. Brandt and C. W. Cryer. Multigrid algorithms for the solution of linear complementarity problems arising from free boundary problems. *SIAM J. Sci. Stat. Comput.*, 4:655–684, 1983.

[BD79] A. Brandt and N. Dinar. Multigrid solutions to elliptic flow problems. In S. Parter, editor, *Numerical Methods for Partial Differential Equations*, pages 53–147. Academic Press, New York, 1979.

[BFT83] A. Brandt, S. R. Fulton, and G. D. Taylor. Improved spectral multigrid methods for periodic elliptic problems. Technical report, Colorado State University, Fort Collins, Colorado, September 1983.

[BG91] A. Brandt and J. Greenwald. Parabolic multigrid revisited. In U. Trottenberg W. Hackbusch, editor, *Multigrid Methods, Vol. III*, volume 98 of *International Series of Numerical Mathematics*. Birkhöuser, Basel, 1991.

[BHM00] W. L. Briggs, V. E. Henson, and S. F. McCormick. *A multigrid tutorial (2nd ed.)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[BK83] J. W. Boerstoel and A. Kassies. Integrating multigrid relaxation into a robust fast solver for transonic potential flows around lifting airfoils. *AIAA*, 83–1885, 1983.

[BL90] A. Brandt and A. A. Lubrecht. Multilevel matrix multiplication and fast solution of integral equations. *J. Comp. Phys.*, 90:348–370, 1990.

[BL97] A. Brandt and I. Livshits. Wave-ray multigrid method for standing wave equations. *Electronic Trans. Num. An.*, 6:162–181, 1997.

[BLE05] Diskin B., Thomas J. L., and Mineck R. E. On general quantitative analysis methods for multigrid solutions. *SIAM J. Sci. Comput.*, 27(1):108–129, 2005.

[BMR83] A. Brandt, S. F. McCormick, and J. W. Ruge. Multigrid methods for differential eigenproblems. *SIAM J. Sci. Stat. Comput.*, 4:244–260, 1983.

[BMR84] A. Brandt, S. F. McCormick, and J. W. Ruge. Algebraic multigrid (AMG) for sparse matrix equations. In D. J.

Evans, editor, *Sparsity and Its Applications*. Cambridge University Press, Cambridge, 1984.

[BO83] A. Brandt and D. Ophir. Gridpack: Toward unification of general grid programming. In B. Enquist and T. Smedsaas, editors, *IFIP Conference on PDE software*, Amsterdam, 1983. North–Holland.

[Boe82] J. W. Boerstoel. A multigrid algorithm for steady transonic potential flows areound aerofoils using Newton iteration. *J. Comput. Phys.*, 48:314–343, 1982.

[Bör81] C. Börgers. *Mehrgitterverfahren für eiene Mehrstellendiskreitsierung der Poissongleichung und für eiene zweidimensionale singulär gestörte Aufgabe.* PhD thesis, Institut für Angewandte Mathematik, Universität Bonn, 1981.

[BR02] A. Brandt and D. Ron. Multigrid solvers and Multilevel Optimization Strategies. pages 1–69. 2002.

[BR03] A. Brandt and D. Ron. Multigrid solvers and multilevel optimization strategies. In J. Cong. and J.R. Shinnerl, editors, *Multilevel Optimization and VLSICAD*, pages 1–69. Kluwer Academic Publishers, Boston, 2003.

[Bra73] A. Brandt. Multi–level adaptive technique (MLAT) for fast numerical solution to boundary value problems. In H. Cabannes and R. Teman, editors, *Proceedings of the Third International Conference on Numerical Methods in Fluid Mechanics*, volume 18 of *Lecture Notes in Physics*, pages 82–89, Berlin, 1973. Springer–Verlag.

[Bra76] A. Brandt. Multi–level adaptive techniques. Technical report, IBM Research Report, 1976.

[Bra77a] A. Brandt. Multi–level adaptive solutions to boundary-value problems. *Math. Comp.*, 31:333–390, 1977.

[Bra77b] A. Brandt. Multi–level adaptive techniques (MLAT) for partial differential equations: ideas and software. In J. R. Rice, editor, *Mathematical Software III*, pages 277–318. Academic Press, New York, 1977.

[Bra79a] A. Brandt. Multi–level adaptive finite–element methods. I. Variational problems. In J. Frehse, D. Pallaschke, and U. Trotenberg, editors, *Special Topics of Applied Mathematics*, pages 91–128. North–Holland, Amsterdam, 1979.

[Bra79b] A. Brandt. Multi-level adaptive techniques (MLAT) for singular-perturbation problems. In P. W. Hemker and J. J. H. Miller, editors, *Numerical Analysis of Singular Perturbation Problems*, pages 53–142. Academic Press, New York, 1979.

[Bra80a] A. Brandt. Multi-level adaptive computations in fluid dynamics. *AIAA J.*, 18:1165–1172, 1980.

[Bra80b] A. Brandt. Numerical stability and fast solutions to boundary value problems. In J. J. H. Miller, editor, *Boundary and Interior Layers—Computational and Asymptotic Methods*, pages 29–49. Boole Press, Dublin, 1980.

[Bra80c] A. Brandt. Stages in developing multigrid solutions. In E. Absi, R. Glowinski, P. Lascaux, and H. Veysseyre, editors, *Numerical Methods for Engineering I*, pages 23–45. Dunod, Paris, 1980.

[Bra81a] A. Brandt. Multigrid solvers for non elliptic and singular perturbation steady-state problems. Technical report, Weizmann Institute of Science, Rehovot, Israel, 1981.

[Bra81b] A. Brandt. Multigrid solvers on parallel computers. In M. H. Schultz, editor, *Elliptic Problem Solvers*, pages 39–83. Academic Press, New York, 1981.

[Bra82a] D. Braess. The convergence rate of a multigrid method with Gauss-Seidel relaxation for the Poisson equation. In W. Hackbusch and U. Trottenberg, editors, *Multigrid Methods*, volume 960 of *Lecture Notes in Mathematics*, pages 368–386, Berlin, 1982. Springer-Verlag.

[Bra82b] A. Brandt. Guide to multigrid development. In W. Hackbusch and U. Trottenberg, editors, *Multigrid Methods*, volume 960 of *Lecture Notes in Mathematics*, pages 220–312. Springer-Verlag, Berlin, 1982.

[Bra82c] A. Brandt. Multigrid solutions to steady-state compressible Navier-Stokes equations. I. In R. Glowinski and J.-L. Lions, editors, *Computing Methods in Applied Sciences and Engineering V*, pages 407–423, 1982.

[Bra83] A. Brandt. Videotape lectures on multigrid methods. Technical report, Office of Instructional Services, Colorado State University, Colorado 80523, Phone 303-491-5416 (obsolete), 1983.

[Bra86] A. Brandt. Algebraic multigrid theory: The symmetric case. *Appl. Math. Comput.*, 19:23–56, 1986.

[Bra88] A. Brandt. Multilevel computations: Review and recent developments. In *Multigrid Methods: theory, Applications and Supercomputing*, pages 35–62. Marcel-Dekker, 1988.

[Bra89] A. Brandt. The Weizmann institute research in multilevel computation: 1988 report. In J. Mandel et al., editors, *Proceedings of the Fourth Copper Mountain Conference on Multigrid Methods*, pages 13–53. SIAM, 1989.

[Bra91] A. Brandt. Rigorous quantitative analysis of multigrid, II: extensions and practical implications. Manuscript, 1991.

[Bra94] A. Brandt. Rigorous quantitative analysis of multigrid: I: constant coefficients two level cycle with l_2 norm,. *SIAM J. Num. Anal.*, 31:1695–1730, 1994.

[Bra00] A. Brandt. General highly accurate algebraic coarsening. *Elect. Trans. Numer. Anal.*, 10:1–20, 2000.

[Bra02] A. Brandt. Multiscale scientific computation: Review 2001. In T. Barth, T. Chan, and R. Haimes, editors, *Multiscale and Multiresolution Methods*, pages 3–96. Springer-Verlag, 2002.

[Bra10] A. Brandt. Principles of systematic upscaling. In *Bridging the Scales in Science and Engineering*, pages 193–215. Oxford University Press, 2010.

[Bro82] J. J. Brown. A multigrid mesh–embedding technique for three–dimensional transonic potential flow analysis. In H. Lomax, editor, *Multigrid Methods*, NASA Conference Publication 2202, pages 131–150. Ames Research Center, Moffett Field, CA, 1982.

[Cau83] D. A. Caughey. Multi–grid calculation of three–dimensional transonic potential flows. *Appl. Math. Comput.*, 13:241–260, 1983.

[CLW78] M. Ciment, S. H. Leventhal, and B.C. Weinberg. The operator compact implicit method for parabolic equations. *J. Comp. Phys.*, 28:135–166, 1978.

[CR68] F. W. C. Campbell and J. Robson. Application of Fourier Analysis to the visibility of gratings. *J. Physiol. (Lond.)*, 197:551–566, 1968.

[Den82a] J. E. Dendy. Black box multigrid. *J. Comput. Phys.*, 48:366–386, 1982.

[Den82b] J. E. Dendy. Black box multigrid for nonsymmetric problems. *Appl. Math. Comput.*, 13:261–284, 1982.

[DH82] H. Deconinck and C. Hirsch. A multigrid method for the transonic full potential equation discretized with finite elements on an arbitrary body fitted mesh. *J. Comput. Phys.*, 48:344–365, 1982.

[Din79] N. Dinar. *Fast Methods for the Numerical Solution of Boundary Value Problems.* PhD thesis, Weizmann Institute of Science, Rehovot, Isreal, 1979.

[Dou05] C. Douglas. MGNet.org free software. `http://mgnet.org/mgnet-codes.html`, 2005.

[Duf82] I. S. Duff. Sparse matrix software for elliptic PDE's. In W. Hackbusch and U. Trottenberg, editors, *Multigrid Methods*, volume 960 of *Lecture Notes in Mathematics*, pages 410–426, Berlin, 1982. Springer-Verlag.

[Fed64a] R. P. Fedorenko. The speed of convergence of one iterative process. *USSR Comput. Math. Math. Phys.*, 4:227, 1964.

[Fed64b] R. P. Fedorenko. The speed of convergence of one iterative process. *Z. Vycisl. Mat. i. Mat. Fiz.*, 4:559–563, 1964. Also in U.S.S.R. Comput. Math. and Math. Phys., 4 (1964), pp. 227–235.

[Fre75] P. O. Frederickson. Fast approximate inversion of large sparse linear systems. Technical report, Lakehead University, Ontario, Canada, 1975.

[FST81] H. Foerster, K. Stüben, and U. Trottenberg. Non–standard multigrid techniques using checkered relaxation and intermediate grids. In M. H. Schultz, editor, *Elliptic Problem Solvers*, pages 285–300. Academic Press, New York, 1981.

[Fuc82] L. Fuchs. Multi–grid solution of the Navier–Stokes equations on non–uniform grids. In H. Lomax, editor, *Multigrid Methods*, NASA Conference Publication 2202, pages 83–100. Ames Research Center, Moffet Field, CA, 1982.

[FW60] G. E. Forsythe and W. R. Wasow. *Finite-difference methods for partial differential equations.* Wiley, New York, 1960.

[FW81] H. Foerster and K. Witsch. On efficient multigrid software for elliptic problems on rectangular domains. *Math. Comput. Simulation*, XXIII:293–298, 1981.

[GG83] S. Geman and D. Geman. Stochastic relaxation, Gibbs distribution, and the Bayesian restoration of images. Technical report, Division of Appl. Math., Brown University, Providence, Rhode Island, September 1983.

[GO77] D. O. Gottlieb and S. A. Orszag. *Numerical Analysis of Spectral Methods.* SIAM, Philadelphia, 1977.

[Gre92] J. C. Greenwald. *Multigrid techniques for parabolic problems.* PhD thesis, Weizmann Institute of Science, June 1992.

[Hac80] W. Hackbusch. The fast numerical solution of very large elliptic difference schemes. *J. Inst. Math. Appl.*, 26:119–132, 1980.

[Hac82] W. Hackbusch. Multi–grid convergence theory. In W. Hackbusch and U. Trottenberg, editors, *Multigrid Methods*, volume 960 of *Lecture Notes in Mathematics*, pages 177–219, Berlin, 1982. Springer–Verlag.

[Hac85] W. Hackbusch. *Multigrid Methods and Applications.* Springer Verlag, Berlin, 1985.

[HT82] W. Hackbusch and U. Trottenberg. *Multigrid Methods*, volume 960 of *Lecture Notes in Mathematics.* Springer–Verlag, Berlin, 1982.

[Hym77] J. M. Hyman. Mesh refinement and local inversion of elliptic differential equations. *J. Comput. Phys.*, 23:124–134, 1977.

[Jam79] A. Jameson. Acceleration of transonic potential flow calculations on arbitrary meshes by the multiple grid method. *AIAA*, 79-1458, 1979.

[Jam83] A. Jameson. Solution of the Euler equations for two dimensional transonic flow by a multigrid method. *Appl. Math. Comput.*, 13:327–355, 1983.

[Jes83] D. C. Jespersen. Design and implementation of a multigrid code for the Euler equations. *Appl. Math. Comput.*, 13:357–374, 1983.

[Kau82] L. Kauffman. Matrix methods for queuing problems. Technical report, Bell Laboratories, Murray Hill, New Jersey, August 1982.

[Kel77] H. B. Keller. Numerical solution of bifurcation and nonlinear eigenvalue problems. In *Applications of Bifurcation Theory*, pages 359–384. Academic Press, New York, 1977.

[Ket82] R. Kettler. Analysis and comparison of relaxation schemes in robust multigrid and preconditioned conjugate gradient methods. In W. Hackbusch and U. Trottenberg, editors, *Multigrid Methods*, volume 960 of *Lecture Notes in Mathematics*, pages 502–534, Berlin, 1982. Springer–Verlag.

[Lan65] S. Lang. *Algebra.* addison–Wesley, New York, 1965.

[LB04] O. E. Livne and A. Brandt. Local mode analysis of multicolor and composite relaxation schemes. *Comp. Math. Appl.*, 47(2-3):301 - 317, 2004.

[Lin76] B. Lindberg. Error estimation and iterative improvement for the numerical solution of operator equations. Technical report, University of Illinois, Urbana, 1976.

[Lin81] J. Linden. *Mehrgitterverfahren für die Poisson–Gleichung in Kreis und Ringgebiet unter Verwendung lokaler Koordinaten.* PhD thesis, Institut für Angewandte Mathematik, Universität Bonn, 1981.

[Liv04] O. E. Livne. Coarsening by compatible relaxation. *Num. Lin. Alg. Appl.*, 11:205–227, 2004.

[LR78] R. E. Lynch and J. R. Rice. The hodie method and its performance. In C. de Boor, editor, *Recent Advances in Numerical Analysis*, pages 143–179. Academic Press, New York, 1978.

[Mei79] J. Meinquet. Multivariate interpolation at arbitrary points made simple. *J. Applied Math. Phys.*, (ZAMP) 30:292–304, 1979.

[Mei80] P. Meissle. Apriori prediction of roundoff error accumulation in the solution of a super-large geodetic normal equation system. Technical report, National Oceanic and Atmospheric Administration, Rockville, Maryland, 1980.

[MR82] D. R. McCarthy and T. A. Reyhner. A multigrid code for the three–dimensional transonic potential flow about inlets. *AIAA J.*, 20:45–50, 1982.

[MS83] D. R. McCarthy and R. C. Swanson. Embedded mesh multigrid treatment of two-dimensional transonic flows. *Appl. Math. Comput.*, 13:399–418, 1983.

[MUG84] MUGTAPE 84. A tape of multigrid software and programs, including GRIDPACK; MUGPACK; simple model programs (CYCLEV, CYCLEC, FASCC, FMG1 and an eigenvalue solver); stokes equations solver; SMORATE; BOXMG [Den82a]; MG00 and MG01 [ST82]. Technical report, Available at the Department of Applied Mathematics, The Weizmann Institute of Science, Rehovot, Israel, 1984.

[Ni82] R.-H. Ni. A multiple grid scheme for solving Euler equations. *AIAA J.*, 20:1565–1571, 1982.

[Nic75] R. A. Nicolaides. On multiple grid and related techniques for solving discrete elliptic systems. *J. Comput. Phys.*, 19:418–431, 1975.

[Nic77] R. A. Nicolaides. On the l^2 convergence of an algorithm for solving finite element equations. *Math. Comp.*, 31:892–906, 1977.

[NOR81] K. A. Narayanan, D. P. O'leary, and A. Rosenfeld. Multi–resolution relaxation. Technical report, Unviersity of Maryland, College Park, Maryland, July 1981.

[Oph78] D. Ophir. *Language for processes of numerical solutions to differential equations.* PhD thesis, Dept. of Applied Mathematics, Weizmann Institute of Science, Rehovot, Israel, 1978.

[RS87] J. Ruge and K. Stüben. Algebraic multigrid (AMG). In S. F. McCormick, editor, *Multigrid Methods, Frontiers in Applied Mathematics*, pages 73–130. SIAM, 1987.

[RSBss] D. Ron, I. Safro, and A. Brandt. Relaxation-based coarsening and multiscale graph organization. *SIAM Multiscale Model. Sim.*, in press. Preprint ANL/MCS-P1696-1009.

[RTW83] M. Ries, U. Trottenberg, and G. Winter. A note on MGR methods. *J. Lin. Alg. Applic.*, 49:1–26, 1983.

[San81] S. L. Sanimoto. Template matching in pyramids. *Computer Graphics and Image Processing*, 16:356–369, 1981.

[SB77] J. C. South and A. Brandt. Application of a multi–level grid method to transonic flow calculations. In T. C. Adamson and M. F. Platzer, editors, *Transonic Flow Problems in Turbomachinery*, pages 180–207. ICASE Report 76–8. Hemisphere, Washington, D.C., 1977.

[SC82] A. Shmilovich and D. A. Caughey. Application of the multi–grid method to calculations of transonic potential flow about wing–fuselage combinations. *J. Comput. Phys.*, 48:462–484, 1982.

[Sch82] S. Schaeffer. *High–order multigrid methods.* PhD thesis, Colorado State University, Fort Collins, Colorado, 1982.

[Sch83] H. Schwichtenberg. *Die Erweiterung des Mehrgitterprogramms MG01 auf nichtlineare Aufbagen.* PhD thesis, Institut für Angewandte Mathematik, Universität Bonn, 1983.

[SGS+06] E. Sharon, M. Galun, D. Sharon, R. Basri, and A. Brandt. Hierarchy and adaptivity in segmenting visual scenes. *Nature*, 442:810–813, 2006.

[ST82] K. Stüben and U. Trottenberg. Multigrid methods: Fundamental algorithms, model problem analysis and applications. In W. Hackbusch and U. Trottenberg, editors, *Multigrid Methods*, volume 960 of *Lecture Notes in Mathematics*, pages 1–176, Berlin, 1982. Springer–Verlag.

[Ste78] H. J. Stetter. The defect correction principle and discretization methods. *Numer. Math.*, 29:425–443, 1978.

[Stü83] K. Stüben. Algebraic multigrid (AMG): experiences and comparisons. *Appl. Math. Comput.*, 13:419–452, 1983.

[SWd85] P. Sonneveld, P. Wesseling, and P. M. de Zeeuw. Multigrid and conjugate gradient methods as convergence acceleration techniques. In D. J. Paddon and H. Holstein, editors, *Multigrid Methods for Integral and Differential Equations*, volume 3 of *The Institute of Mathematics and its Applications Conference Series*, pages 117–168. Clarendon Press, Oxford, 1985.

[SZH83] C. L. Streett, T. A. Zang, and M. Y. Hussaini. Spectral multigrid methods with applications to transonic potential flow. In *Preliminary Proc. Internat. Multigrid Conference*, Ft. Collins, 1983. Institute for Computational Studies at Colorado State University.

[Tan71] K. Tanabe. Projection methods for solving a singular system of linear equations and its applications. *Num. Math.*, 17:203–214, 1971.

[Tem77] R. Temam. *Navier–Stokes Equations: Theory and Numerical Analysis.* North–Holland, 1977.

[Ter83] D. Terzopoulos. Multilevel computational processes for visual surface reconstruction. *Computer Vision, Graphics, and Image Processing*, 24:52-96, 1983.

[TF81] T. Thunell and L. Fuchs. Numerical solution of the Navier–Stokes equations by multi–grid techniques. In C. Taylor and A. B. Schrefler, editors, *Numerical Methods in Laminar and Turbulent Flow*, pages 141–152, Swansea, 1981. Pineridge Press.

[Tho83] C.-A. Thole. *Beiträge zur Fourieranalyse von Mehrgittermethoden: V–cycle, ILU– Glättung, anisotrope Operatoren.* PhD thesis, Institut für Angewandte Mathematic, Universität Bonn, 1983.

[TOS00] U. Trottenberg, C. W. Oosterlee, and A. Schuller. *Multigrid.* Academic Press, first edition, December 2000.

[TTKR10] A. Thekale, Gradl T., K. Klamroth, and U. Rüde. Optimizing the number of multigrid cycles in the full multigrid algorithm. *Num. Lin. Alg. Appl.*, 17(2–3):199–210, April 2010.

[VDR79] I. Y. Vakhytinsky, L. M. Dudkin, and A. A. Ryvkin. Iterative aggregation – a new approach to the solution of large–scale equations. *Ecenometrica*, 47:821–841, 1979.

[vdWvdVM83] A. J. van der Wees, J. van der Vooren, and J. H. Meelker. Robust calculation of 3D transonic potential flow based on the nonlinear fas multigrid method and incomplete LU–decomposition. *AIAA*, 85–1950 CP, 1983.

[Ver83] R. Verfürth. Two algorithms for mixed problems. In *Preliminary Proc. for Internat. Multigrid Conference*, Ft. Collins, CO, 1983. Institute for Computational Studies at Colorado State University.

[Ver84a] R. Verfürth. The contraction number of a multigrid method with mesh ratio 2 for solving Poisson's equation. *J. Lin. Alg. Applic.*, 60:332–348, 1984.

[Ver84b] R. Verfürth. A multilevel algorithm for mixed problems. *SIAM J. Numer. Anal.*, 21:264–271, 1984.

[WB79] H. R. Wilson and J. R. Bergen. A four mechanism model for spatial vision. *Vision. Res.*, 19:19–32, 1979.

[Wei01] R. Weinands. *Extended Local Fourier Analysis for Multigrid: Optimal Smoothing, Coarse Grid Correction and Preconditioning.* PhD thesis, Universität zu Köln, 2001.

[Wes77] P. Wesseling. Numerical solution of stationary Navier–Stokes equation by means of a multiple grid method and newton iteration. Technical report, Dept. of Math., Delft University of Technology, 1977.

[WS80] P. Wesseling and P. Sonneveld. Numerical experiments with a multiple grid and a preconditioned Lanczos type method. In R. Rautmann, editor, *Approximation Methods for Navier-Stokes Problems*, volume 771 of *Lecture Notes in Mathematics*, pages 543–562. Springer-Verlag, Berlin, 1980.

[ZWH82] T. A. Zang, Y. S. Wong, and M. Y. Hussaini. Spectral multigrid methods for elliptic equations. *J. Comp. Phys.*, 48:485–501, 1982.

[ZWH83] T. A. Zang, Y. S. Wong, and M. Y. Hussaini. Spectral multigrid methods for elliptic equations II. Technical report, NASA, 1983.

Index