MATHEMATICS OF COMPUTATION, VOLUME 31, NUMBER 138
APRIL 1977, PAGES 333–390

Multi-Level Adaptive Solutions to Boundary-Value Problems*

By Achi Brandt

Abstract. The boundary-value problem is discretized on several grids (or finite-element spaces) of widely different mesh sizes. Interactions between these levels enable us (i) to solve the possibly nonlinear system of n discrete equations in $O(n)$ operations ($40n$ additions and shifts for Poisson problems); (ii) to conveniently adapt the discretization (the local mesh size, local order of approximation, etc.) to the evolving solution in a nearly optimal way, obtaining "∞-order" approximations and low n, even when singularities are present. General theoretical analysis of the numerical process. Numerical experiments with linear and nonlinear, elliptic and mixed-type (transonic flow) problems—confirm theoretical predictions. Similar techniques for initial-value problems are briefly discussed.

1. Introduction. In most numerical procedures for solving partial differential equations, the analyst first discretizes the problem, choosing approximating algebraic equations on a finite-dimensional approximation space, and then devises a numerical process to (nearly) solve this huge system of discrete equations. Usually, no real interplay is allowed between discretization and solution processes. This results in enormous waste: The discretization process, being unable to predict the proper resolution and the proper order of approximation at each location, produces a mesh which is too fine. The algebraic system thus becomes unnecessarily large in size, while accuracy usually remains rather low, since local smoothness of the solution is not being properly exploited. On the other hand, the solution process fails to take advantage of the fact that the algebraic system to be solved does not stand by itself, but is actually an approximation to continuous equations, and therefore can itself be similarly approximated by other (much simpler) algebraic systems.

The purpose of the work reported here is to study how to intermix discretization and solution processes, thereby making both of them orders-of-magnitude more effective The method to be proposed is not "saturated", that is, accuracy grows indefinitely as computations proceed. The rate of convergence (overall error E as function of computational work W) is in principle of "infinite order", e.g., $E \sim \exp(-\beta^d W)$ for a d-dimensional problem which has a solution with scale-ratios $\geqslant \beta > 0$; or $E \sim \exp(-W^{1/2})$, for problems with arbitrary thin layers (see Section 9).

The basic idea of the Multi-Level Adaptive Techniques (MLAT) is to work not

Received July 22, 1976.

AMS (MOS) subject classifications (1970). Primary 65N05, 65N10, 65N20, 65N30; Secondary 35Jxx, 35M05, 39A10, 76H05, 76G15.

*The research reported here was partly supported by the Israel Commission for Basic Research. Part of the research was conducted at the Thomas J. Watson IBM Research Center, Yorktown Heights, New York. Part of it was also conducted at the Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, Virginia.

with a single grid, but with a sequence of grids ("levels") of increasing fineness, each of which may be introduced and changed in the process, and constantly interact with each other. For description purposes, it is convenient to regard this technique as composed of two main concepts:

(1) *The Multi-Grid (MG) Method for Solving Discrete Equations.* This method iteratively solves a system of discrete (finite-difference or finite-element) equations on a given grid, by constant interactions with a hierarchy of coarser grids, taking advantage of the relation between different discretizations of the same continuous problem. This method can be viewed in two complementary ways: One is to view the coarser grids as correction grids, accelerating convergence of a relaxation scheme on the finest grid by efficiently liquidating smooth error components. (See general description in Section 2 and algorithm in Section 4.) Another point of view is to regard finer grids as the correction grids, improving accuracy on coarser grids by correcting their forcing terms. The latter is a very useful point of view, making it possible to manipulate accurate solutions on coarser grids, with only infrequent "visits" to pieces of finer levels. (This is the basis for the multi-grid treatment of nonuniform grids; cf. Sections 7.2 and 7.5. The FAS mode for nonlinear problems and the adaptive procedures stem from this viewpoint.) The two seemingly different approaches actually amount to the same algorithm (in the simple case of "coextensive" levels).

The multi-grid process is very efficient: A discrete system of n equations (n points in the finest grid) is solved, to the desired accuracy, in $O(n)$ computer operations. If $\bar{P}$ parallel processors are available, the required number of computer steps is $O(n/\bar{P} + \log n)$. For example, only $40n$ additions and shifts (or $35n$ microseconds CYBER 173 CPU time) are required for solving the 5-point Poisson equation on a grid with n points (see Section 6.3). This efficiency does not depend on the shape of the domain, the form of the boundary conditions, or the mesh-size, and is not sensitive to choice of parameters. The memory area required is essentially only the minimal one, that is, the storage of the problem and the solution. In fact, if the amount of numerical data is small and only few functionals of the solution are wanted, the required memory is only $O(\log n)$, with no need for external memory (see Section 7.5).

Multi-grid algorithms are not difficult to program, if the various grids are suitably organized. We give an example (Appendix B) of a FORTRAN program, showing the typical structure, together with its computer output, showing the typical efficiency. With such an approach, the programming of any new multi-grid problem is basically reduced to the programming of a usual relaxation routine. The same is true for nonlinear problems, where no linearization is needed, due to the FAS (Full Approximation Storage) method introduced in Section 5.

Multi-grid solution times can be predicted in advance—a recipe is given and compared with numerical tests (Section 6). The basic tool is the local mode (Fourier) analysis, applied to the locally linearized-frozen difference equations, ignoring far boundaries. Such an analysis yields a very good approximation to the behavior of the high-frequency error modes, which are exactly the only significant modes in assessing the multi-grid process, since the low-frequency error modes are liquidated by coarse-grid

processing, with negligible amounts of computational work. Thus, mode analysis gives a very realistic prediction of convergence rates per unit work. (For model problems, the analysis can be made rigorous; see Appendix C.) The mode analysis can, therefore, be used to choose suitable relaxation schemes (Section 3) and suitable criteria for switching and interpolating between the grids (Appendix A). Our numerical tests ranged from simple elliptic problems to nonlinear mixed-typed (transonic flow) problems, which included hyperbolic regions and discontinuities (shocks). The results show that, as predicted by the local mode analysis, errors in all these problems are reduced by an order of magnitude (factor 10) expending computational work equivalent to 4 to 5 relaxation sweeps on the finest grid.

(2) *Adaptive Discretization.* Mesh-sizes, orders of approximation and other discretization parameters are treated as spatial variables. Using certain general internal criteria, these variables are controlled in a suboptimal way, adapting themselves to the computed solution. The criteria are devised to obtain maximum overall accuracy for a given amount of calculations; or, equivalently, minimum of calculations for given accuracy. (In practice only near-optimality should of course be attempted, otherwise the required control would become more costly than the actual computations. See Section 8.) The resulting discretization will automatically resolve thin layers (when required), refine meshes near singular points (that otherwise may "contaminate" the whole solution), exploit local smoothness of solutions (in proper scale), etc. (see Section 9).

Multi-grid processing and adaptive discretization can be used independently of each other, but their combination is very fruitful: MG is the only fast (and convenient) method to solve discrete equations on the nonuniform grids typically produced by the adaptive procedure. Its iterative character fits well into the adaptive process. The two ideas use and relate similar concepts, similar data structure, etc. In particular, an efficient and very flexible general way to construct an adaptive grid is as a sequence of uniform subgrids, the same sequence used in the multi-grid process, but where the finer levels may be confined to increasingly smaller subdomains to produce the desired local refinement. In this structure, the difference equations can be defined separately on each of the uniform subgrids, interacting with each other through the multi-grid process Thus, difference equations should only be constructed on equidistant points. This facilitates the employment of high and adaptive orders of approximation. Moreover, the finer, localized subgrids may be defined in terms of suitable local coordinates, facilitating, for example, the use of high-order approximations near pieces of boundary, with all these pieces naturally patched together by the multi-grid process (Section 7).

The presentation in this article is mainly in terms of finite-difference solutions to partial-differential boundary-value problems. The basic ideas, however, are more general applicable to integro-differential problems, functional minimization problems, etc., and to *finite-element* discretizations. The latter is briefly discussed in Sections A.5 and 7.3 and as closing remarks to Sections 8.1 and 8.3. Extensions to *initial-value problems* are discussed in Appendix D. Section 10 presents historical notes and acknowledgements.

Contents of the article:

2. Multi-Grid Philosophy. Suppose we have a set of grids $G^0, G^1, \ldots, G^M$, all approximating the same domain Ω with corresponding mesh-sizes $h_0 > h_1 > \cdots > h_M$. For simplicity one can think of the familiar uniform square grids, with the mesh-size ratio $h_{k+1} : h_k = 1 : 2$. Suppose further that a differential problem of the form

$$LU(x) = F(x) \quad \text{in } \Omega, \qquad \Lambda U(x) = \Phi(x) \quad \text{on the boundary } \partial\Omega, \tag{2.1}$$

is given. On each grid G^k this problem can be approximated by difference equations of the form

$$L^k U^k(x) = F^k(x) \quad \text{for } x \in G^k, \qquad \Lambda^k U^k(x) = \Phi^k(x) \quad \text{for } x \in \partial G^k. \tag{2.2}$$

(See example in Section 3.1.) We are interested in solving this discrete problem on the finest grid, G^M. The main idea is to exploit the fact that the discrete problem on a coarser grid, G^k, say, approximates the same differential problem and hence can be used as a certain approximation to the G^M problem. A simple use of this fact has long been made by various authors (e.g., [14]); namely, they first solved (approximately) the G^k problem, which involves an algebraic system much smaller and thus much easier to solve than the given G^M problem, and then they interpolated their solution from G^k to G^M, using the result as a first approximation in some iterative process for solving the G^M problem. A more advanced technique was to use a still coarser grid in a similar manner when solving the G^k problem, and so on. The next natural step is to ask whether we can exploit the proximity between the G^k and G^M problems not only in generating a good first approximation on G^M, but also in the process of improving the first approximation.

More specifically let u^M be an approximate solution of the G^M problem, and let

$$L^M u^M = F^M - f^M, \qquad \Lambda^M u^M = \Phi^M - \phi^M. \tag{2.3}$$

The discrepancies f^M and ϕ^M are called the residual functions, *or residuals.* Assuming for simplicity that L and Λ are linear (cf. Section 5 for the nonlinear case), the exact discrete solution is $U^M = u^M + V^M$, where the correction V^M satisfies the *residual equations*

$$L^M V^M = f^M, \qquad \Lambda^M V^M = \phi^M. \tag{2.4}$$

Can we solve this equation, to a good first approximation, again by interpolation from solutions on coarser grids? As it is, the answer is generally negative. Not every G^M problem has meaningful approximation on a coarser grid G^k. For instance, if the right-hand side f^M fluctuates rapidly on G^M, with wavelength less than $4h_M$, these fluctua-

tions are not visible on, and therefore cannot be approximate by, coarser grids. Such rapidly-fluctuating residuals f^M are exactly what we get when the approximation u^M has itself been obtained as an interpolation from a coarser-grid solution.

An effective way to damp rapid fluctuations in residuals is by usual relaxation procedures, e.g., the Gauss-Seidel relaxation (see Section 3). At the first few iterations such procedures usually seem to have fast convergence, with residuals (or corrections) rapidly decreasing from one iteration to the next, but soon after the convergence rate levels off and becomes very slow. Closer examination (see Section 3 below) shows that the convergence is fast as long as the residuals have strong fluctuations on the scale of the grid. As soon as the residuals are smoothed out, convergence slows down.

This is then exactly the point where relaxation sweeps should be discontinued and approximate solution of the (smoothed out) residual equations by coarser grids should be employed.

The Multi-Grid (MG) methods are systematic methods of mixing relaxation sweeps with approximate solution of residual equations on coarser grids. The residual equations are in turn also solved by combining relaxation sweeps with corrections through still coarser grids, etc. The coarsest grid G^0 is coarse enough to make the solution of its algebraic system inexpensive compared with, say, one relaxation sweep over the finest grid.

The following sections further explain these ideas. Section 3.1 explains, through a simple example, what is a relaxation sweep and shows that it indeed smooths out the residuals very efficiently. The smoothing rates of general difference systems are summarized in Section 3.2. A full multi-grid algorithm, composed of relaxation sweeps over the various grids with suitable interpolations in between, is then presented in Section 4. An important modification for nonlinear problems is described in Section 5 (and used later as the basic algorithm for nonuniform grids and adaptive procedures). Appendix A supplements these with suitable stopping criteria, details of the interpolation procedures and special techniques (partial relaxation).

3. Relaxation and its Smoothing Rate.

3.1. *An Example.* Suppose, for example, we are interested in solving the partial differential equation

$$LU(x, y) \equiv a\frac{\partial^2 U(x, y)}{\partial x^2} + c\frac{\partial^2 U(x, y)}{\partial y^2} = F(x, y) \tag{3.1}$$

with some suitable boundary conditions. Denoting by U^k and F^k approximations of U and F, respectively, on the grid G^k, the usual second-order discretization of (3.1) is

$$L^k U^k_{\alpha,\beta} \equiv a\frac{U^k_{\alpha+1,\beta} - 2U^k_{\alpha,\beta} + U^k_{\alpha-1,\beta}}{h_k^2} + c\frac{U^k_{\alpha,\beta+1} - 2U^k_{\alpha,\beta} + U^k_{\alpha,\beta-1}}{h_k^2} = F^k_{\alpha,\beta}, \tag{3.2}$$

where

$$U^k_{\alpha,\beta} = U^k(\alpha h_k, \beta h_k), \qquad F^k_{\alpha,\beta} = F^k(\alpha h_k, \beta h_k); \quad \alpha, \beta \text{ integers.}$$

(In the multi-grid context it is important to define the difference equations in this

divided form, without, for example, multiplying throughout by h_k^2, in order to get the proper relative scale at the different levels.) Given an approximation u to U^k, a simple example of a relaxation scheme to improve it is the following.

Gauss-Seidel Relaxation. The points (α, β) of G^k are scanned one by one in some prescribed order; e.g., lexicographic order. At each point the value $u_{\alpha,\beta}$ is replaced by a new value, $\bar{u}_{\alpha,\beta}$, such that Eq. (3.2) at that point is satisfied. That is, $\bar{u}_{\alpha,\beta}$ satisfies

$$(3.3) \qquad a\frac{u_{\alpha+1,\beta} - 2\bar{u}_{\alpha,\beta} + \bar{u}_{\alpha-1,\beta}}{h_k^2} + c\frac{u_{\alpha,\beta+1} - 2\bar{u}_{\alpha,\beta} + \bar{u}_{\alpha,\beta-1}}{h_k^2} = F^k_{\alpha,\beta},$$

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

A complete pass, scanning in this manner all the points of G^k, is called a (Gauss-Seidel lexicographic) G^k *relaxation sweep.* The new approximation $\bar{u}$ does not satisfy (3.2), and further relaxation sweeps may be required to improve it. An important quantity therefore is the *convergence factor*, μ say, which may be defined by

$$(3.4) \qquad \mu = \|\bar{v}\| / \|v\|, \quad \text{where} \quad v = U^k - u, \quad \bar{v} = U^k - \bar{u},$$

$\|\cdot\|$ being any suitable discrete norm.

The rate of convergence of the above relaxation scheme is asymptotically very slow. That is, except for the first few relaxation sweeps we have $\mu = 1 - O(h_k^2)$. This means that we have to perform $O(h_k^{-2})$ relaxation sweeps to reduce the error order of magnitude.

In the multi-grid method, however, the role of relaxation is not to reduce the error, but to smooth it out; i.e., to reduce the high-frequency components of the error (the lower frequencies being reduced by relaxation sweeps on coarser grids). In fact, since smoothing is basically a local process (high frequencies have short coupling range), we can analyze it in the interior of G^k by (locally) expanding the error in Fourier series. This will allow us to study separately the convergence rate of each Fourier component, and, in particular, the convergence rate of high-frequency components, which is the rate of smoothing.

Thus to study the $\theta = (\theta_1, \theta_2)$ Fourier component of the error functions v and $\bar{v}$ before and after the relaxation sweep, we put

$$(3.5) \qquad v_{\alpha,\beta} = A_\theta e^{i(\theta_1\alpha + \theta_2\beta)} \quad \text{and} \quad \bar{v}_{\alpha,\beta} = \bar{A}_\theta e^{i(\theta_1\alpha + \theta_2\beta)}.$$

Subtracting (3.2) from (3.3), we get the relation

$$(3.6) \qquad a(v_{\alpha+1,\beta} - 2\bar{v}_{\alpha,\beta} + \bar{v}_{\alpha-1,\beta}) + c(v_{\alpha,\beta+1} - 2\bar{v}_{\alpha,\beta} + \bar{v}_{\alpha,\beta-1}) = 0,$$

from which, by (3.5),

$$(ae^{i\theta_1} + ce^{i\theta_2})A_\theta + (ae^{-i\theta_1} + ce^{-i\theta_2} - 2a - 2c)\bar{A}_\theta = 0.$$

Hence the convergence factor of the θ component is

$$\mu(\theta) = \left|\frac{\bar{A}_\theta}{A_\theta}\right| = \left|\frac{ae^{i\theta_2} + ce^{i\theta_2}}{2a + 2c - ae^{-i\theta_1} - ce^{-i\theta_2}}\right|. \tag{3.7}$$

Define $|\theta| = \max(|\theta_1|, |\theta_2|)$. In domains of diameter $O(1)$ the lowest Fourier components have $|\theta| = O(h_k)$, and their convergence rate therefore is, $\mu(\theta) = 1 - O(h_k^2)$. Here, however, we are interested in the *smoothing factor*, which is defined by

$$\bar{\mu} = \max_{\hat{\rho}\pi \leqslant |\theta| \leqslant \pi} \mu(\theta), \tag{3.8}$$

where $\hat{\rho}$ is the mesh-size ratio and the range $\hat{\rho}\pi \leqslant |\theta| \leqslant \pi$ is the suitable range of high-frequency components, i.e., the range of components that cannot be approximated on the coarser grid, because its mesh-size is $h_{k-1} = h_k/\hat{\rho}$. We will assume here that $\hat{\rho} = \frac{1}{2}$, which is the usual ratio (cf. Section 6.2).

Consider first the case $a = c$ (Poisson equation). A simple calculation shows that $\bar{\mu} = \mu(\pi/2, \arccos 4/5) = .5$. 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,

$$\mu\left(\frac{\pi}{2}, 0\right) = \left[\frac{a^2 + c^2}{a^2 + (c + 2a)^2}\right]^{1/2}, \tag{3.9}$$

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

Line Relaxation. Instead of treating each point (α, β) of G^k separately, one takes simultaneously a line of points at a time, where a line is the set of all points (α, β) in G^k with the same α (a vertical line). All the values $u_{\alpha\beta}$ on such a line are simultaneously replaced by new values $\bar{u}_{\alpha\beta}$ which simultaneously satisfy all the Eqs. (3.2) on that line. (This is easy and inexpensive to do, since the system of equations to be solved for each such line is a tridiagonal, diagonally dominant system. See, e.g., in [17].) As a result, we get the same relation as (3.3) above, except that $u_{\alpha,\beta+1}$ is replaced by $\bar{u}_{\alpha,\beta+1}$. Hence, instead of (3.7) we will get:

$$\mu(\theta) = \left|\frac{a}{2(a + c - c\cos\theta_2) - ae^{-i\theta_1}}\right| \tag{3.10}$$

from which one can derive the smoothing factor

$$\bar{\mu} = \max\left\{5^{-1/2}, \frac{a}{a + 2c}\right\}, \tag{3.11}$$

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

3.2. *General Results.* The above situation is very general (see [4] and Chapter 3 of [3]): For any uniformly elliptic system of difference equations, it can be shown that few relaxation sweeps are enough to reduce the high-frequency error components by an order of magnitude. The same holds for degenerate elliptic systems, provided a suitable relaxation scheme is selected. A scheme of line-relaxation which alternately uses

all line directions and all sweeping directions is suitable for *any* degenerate case. Moreover, such a scheme is suitable even for nonelliptic systems, provided it is used "*selectively*"; i.e., the entire domain is swept in all directions, but new values are not introduced at points where a local test shows the equation to be nonelliptic and the local "time-like" direction to conflict with the current sweeping direction (a conflict arises when some points are to be relaxed later than neighboring points belonging to their domain of dependence). In hyperbolic regions a selected direction would locally operate similar to marching in the local "time-like" direction, thus leaving no (or very small) residuals.

By employing *local mode analysis* (analysis of Fourier components) similiar to the example above, one can explicitly *calculate the smoothing rate* $|\log \bar{\mu}|^{-1}$ for any given difference equation with any given relaxation scheme. (Usually $\bar{\mu}$ should be calculated numerically; an efficient FORTRAN subroutine exists; typical values are given in Table 1, in Section 6.2.) In this way, one can select the best relaxation scheme from a given set of possibilities. The selection of the difference equation itself may also take this aspect into account. This analysis can also be done for *nonlinear* problems (or linear problems with nonconstant coefficients), by local linearization and coefficients freeze. Such localization is fully justified here, since we are interested only in a local property (the property of smoothing. By constrast, one cannot make similar mode analysis to predict the overall convergence rate μ of a given relaxation scheme, since this is not a local property).

An important feature of the smoothing rate $\bar{\mu}$ is its *insensitivity*. In the above example no relaxation parameters were assumed. We could introduce the usual *relaxation parameter* ω; i.e., replace at each point the old value $u_{\alpha,\beta}$ not with the calculated $\bar{u}_{\alpha,\beta}$, but with $u_{\alpha,\beta} + \omega(\bar{u}_{\alpha,\beta} - u_{\alpha,\beta})$. The mode analysis shows, however, that no $\omega \neq 1$ provides a smoothing rate better than $\omega = 1$. In other cases, $\omega = 1$ is not optimal, but its $\bar{\mu}$ is not significantly larger than the minimal $\bar{\mu}$. In delayed-displacement relaxation schemes a value $\omega < \omega_{\text{critical}} < 1$ should often be used to obtain $\bar{\mu} < 1$, but there is no sensitive dependence on the precise value of ω, and suitable values are easily obtained from the local mode analysis. Generally, smoothing rates of delayed-displacement schemes are somewhat worse than those of immediate-displacement schemes, and the latter should therefore be preferred, except when parallel processing is used. In hyperbolic regions immediate-displacement schemes *should* be used, with $\omega = 1$.

3.3. *Acceleration by Weighting.* The smoothing factor $\bar{\mu}$ may sometimes be further improved by various parameters introduced into the scheme. Since $\bar{\mu}$ is reliably obtained from the local mode analysis, we can choose these parameters to minimize $\bar{\mu}$. For linear problems, such optimal parameters can be determined once and for all, since they do not depend on the shape of the domain. For nonlinear problems precise optimization is expensive and one should prefer the simpler, more robust relaxation schemes, such as SOR.

One general way of parametrization is the *weighting of corrections.* We first calculate, in any relaxation scheme, the required correction $\delta u_\nu = \bar{u}_\nu - u_\nu$ (where $\nu =$

(α, β) or, for a general dimension d, $\nu = (\nu_1, \nu_2, \ldots, \nu_d)$, ν_j integers). Then, instead of introducing these corrections, we actually introduce corrections which are some linear combination of δu_ν at neighboring points. That is, the actual new values are

$$\tilde{u}_\nu = u_\nu + \sum_{\gamma \in \Gamma} \omega_\gamma \delta u_{\nu+\gamma}, \tag{3.12}$$

where the weights ω_γ are the parameters, $\gamma = (\gamma_1, \gamma_2, \ldots, \gamma_d)$, γ_j integers, and Γ is a small set near $(0, 0, \ldots, 0)$. For any fixed Γ we can optimize the weights. In case $\Gamma = \{0\}$, ω_0 is the familiar relaxation parameter. Weighting larger Γ is useful in delayed-displacement relaxation schemes. For immediate-displacement schemes one can examine taking weighted averages of old and new values.

Examples. In case of simultaneous displacement (Jacobi) relaxation for the 5-point Poisson equation, the optimal weights for $\Gamma = \{0\}$ is $\omega_{00} = .8$, for which the smoothing factor is $\bar{\mu} = .60$. For the set $\Gamma = \{(\gamma_1, \gamma_2)\colon |\gamma_1| + |\gamma_2| \leqslant 1\}$ the optimal weights are $\omega_{00} = 6\omega_{0,\pm 1} = 6\omega_{\pm 1,0} = 48/41$, yielding $\bar{\mu} = 9/41$. This rate seems very attractive; the smoothing obtained in one sweep equals that obtained by $(\log 9/41)/(\log 1/2) = 2.2$ sweeps of Gauss-Seidel relaxation. Actually, however, each sweep of this weighted-Jacobi relaxation requires nine additions and three multiplications per grid point, whereas each Gauss-Seidel sweep requires only four additions and one multiplication per point, so that the two methods have almost the same convergence rate per operation, Gauss-Seidel being slightly faster. The weighted Jacobi scheme is considerably more efficient than any other simultaneous-displacement scheme, but like any carefully weighted scheme, it is considerably more sensitive to various changes.

The acceleration by weighting can be more significant for higher-order equations. For the 13-point biharmonic operator, Gauss-Seidel relaxation requires twelve additions and three multiplications per grid point and gives $\bar{\mu} = .802$, while weighted Jacobi (with weights $\omega_{00} = 1.552$, $\omega_{0,\pm 1} = \omega_{\pm 1,0} = .353$) requires seventeen additions and five multiplications per point and gives $\bar{\mu} = .549$, which is 2.7 times faster. (The best relaxation sweep for the biharmonic equation $\Delta^2 U = F$ is to write it as the system $\Delta V = F$, $\Delta U = V$ and sweep Gauss-Seidel, alternatively on U and V. Such a double sweep costs eight additions and two multiplications per grid point, and yields $\bar{\mu} = .5$. But a similar procedure is not possible for general fourth-order equations.)

4. A Multi-Grid Algorithm (Cycle C) for Linear Problems. There are several actual algorithms for carrying out the basic multi-grid idea, each with several possible variations. We present here an algorithm (called "Cycle C" in [3]) which is easy to program, generally applicable and never significantly less efficient than the others ("Cycle A" and "Cycle B"). The operation of the algorithm for linear problems is easier to learn, and is therefore described first. In the next section the FAS (Full Approximation Storage) mode of operation, suitable for nonlinear problems and other important generalizations, will be described. A flow-chart of the algorithm is given in Figure 1. (For completeness, we also flowchart, in Figure 2, Cycles A and B.) A sample FORTRAN program of Cycle C, together with a computer output, is given in Appendix B.

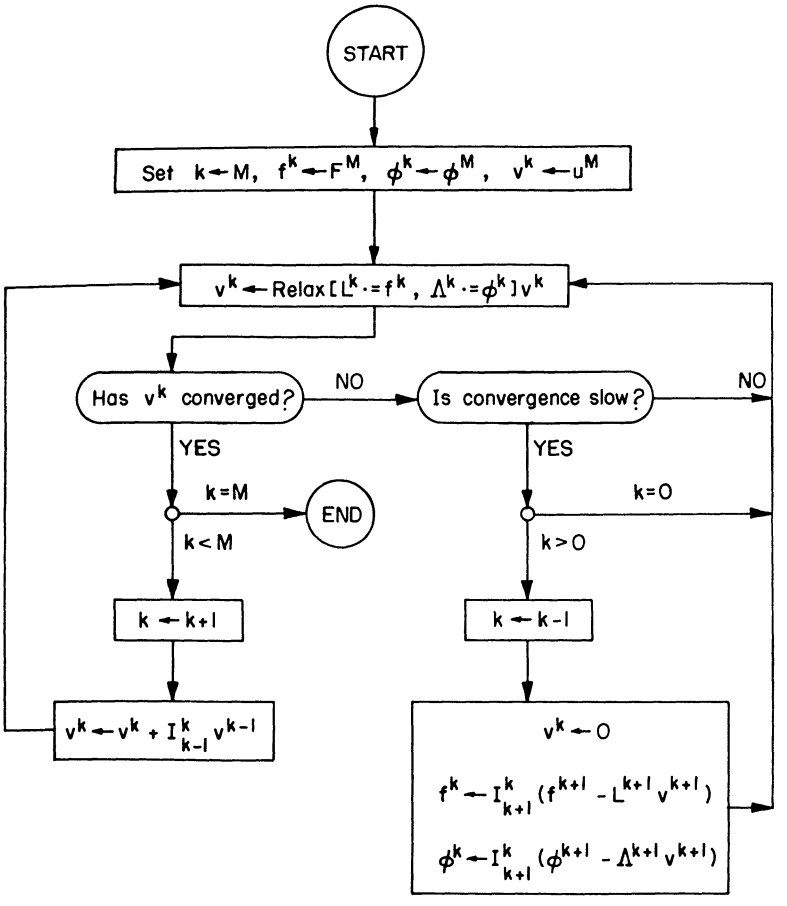

FIGURE 1. *Cycle* C, *Linear Problems*

Cycle C starts with some approximation u_0^M being given on the finest grid G^M. In the linear case one can start with any approximation, but a major part of the computations is saved if u_0^M has smooth residuals (e.g., if u_0^M satisfies the boundary conditions and $L^M u_0^M - F^M$ is smooth. As explained in Section 6, smoothing the residuals involves most of the computational effort). In the nonlinear case, one may have to use a continuation procedure, usually performed on coarser grids (cf. Section 8.2). Even for linear problems, the most efficient algorithm is to obtain u_0^M by interpolating from an approximate solution u^{M-1} calculated on G^{M-1} by a similar algorithm. (Hence the denomination "cycle" for our present algorithm, which would generally serve as the basic step in processes of continuation, refinement and grid adaptation, or as a time step in evolution problems.) For highest efficiency, the interpolation from u^{M-1} to u_0^M should be of sufficiently high order, to exploit all smoothness in u^{M-1}. (Cf. (A.7) in Section A.2, and see also Section 6.3.)

The basic rule in Cycle C is that each v^k (the function defined on the grid G^k; $k = 0, 1, \ldots, M-1$) is designed to serve as a correction for the approximation v^{k+1} previously obtained on the next finer grid G^{k+1}, if and when that approximation

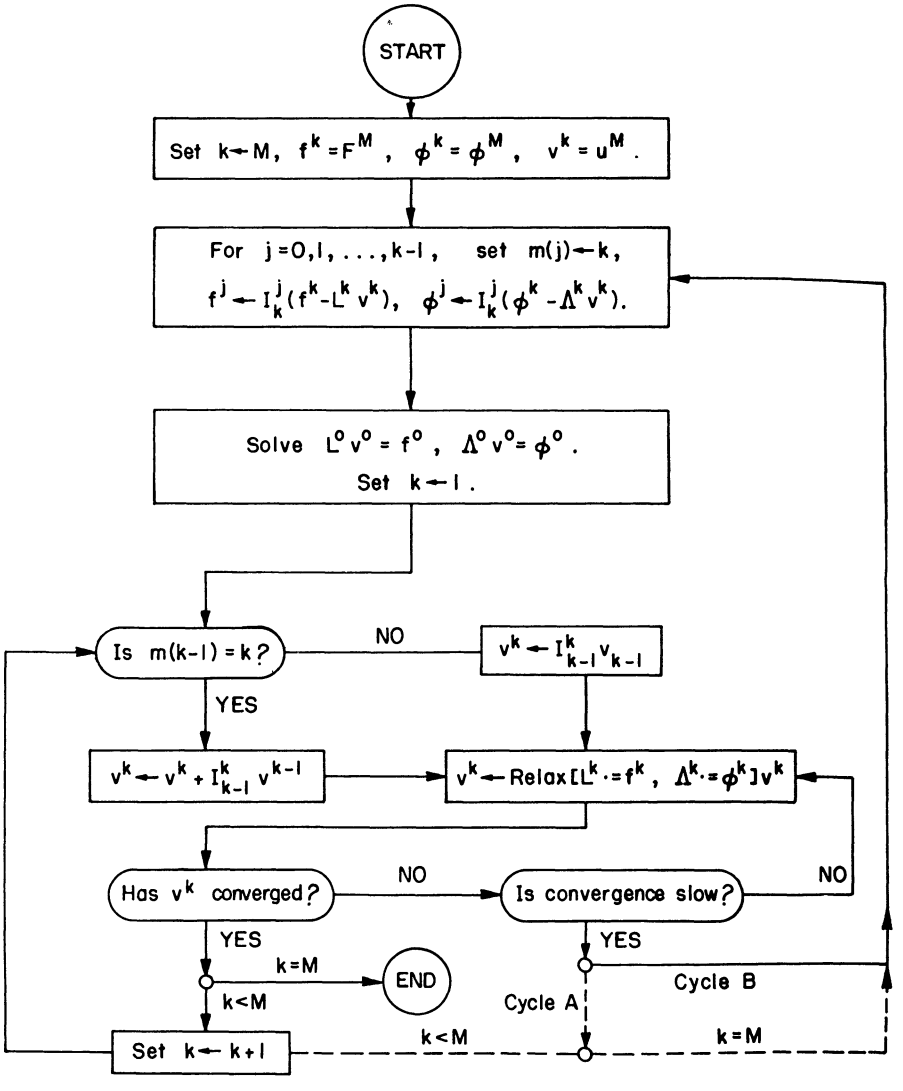

FIGURE 2. *Cycles* A *and* B, *Linear Problems*

actually requires a coarse-grid correction, i.e., if and when relaxation over G^{k+1} exhibits a slow rate of convergence. Thus, the equations to be (approximately) satisfied by v^k are**

$$L^k V^k = f^k, \qquad \Lambda^k V^k = \phi^k, \tag{4.1}$$

where f^k and ϕ^k are the residuals (to the interior equation and the boundary condition, respectively) left by v^{k+1}, that is,

$$f^k = I^k_{k+1}(f^{k+1} - L^{k+1}v^{k+1}), \qquad \phi^k = I^k_{k+1}(\phi^{k+1} - \Lambda^{k+1}v^{k+1}). \tag{4.2}$$

**We denote by V^k the functions in the equations, to distinguish them from their computed approximations v^k. When v^k is changing in the algorithm (causing V^{k-1} to change), V^k remains fixed.

We use the notation I_m^k to represent *interpolation* from G^m to G^k. In case $m > k$, I_m^k may represent a simple transfer of values to the coarser grid G^k from the corresponding points in the finer grid G^m; or instead, it may represent transfer of some weighted averages. In case $k > m$, as in step (e) below, I_m^k is usually a polynomial interpolation of a suitable order (at least the order of the differential equation. See Sections A.2 and A.4 for more details).

The equations on G^k are thus defined in terms of the approximate solution on G^{k+1}. On the finest grid G^M, the equations are the original ones; namely

$$f^M = F^M, \qquad \phi^M = \Phi^M, \qquad v^M = u^M. \tag{4.3}$$

The steps of the algorithm are the following:

(a) Set $k \leftarrow M$ (k is the working level; we start at the finest level), and introduce the given approximation $v^M \leftarrow u_0^M$.

(b) Improve v^k by one relaxation sweep for the difference equations (4.1). Symbolically, we write such a sweep as

$$v^k \leftarrow \text{Relax}[L^k \cdot = f^k, \Lambda^k \cdot = \phi^k]v^k. \tag{4.4}$$

(c) If relaxation has sufficiently converged (the precise criterion is described in Sections A.7 and A.8), go to step (e). If not, and if the convergence rate is still fast (by a criterion given in Section A.6) go back to step (b). If convergence is not obtained and the rate is slow, go to step (d).

(d) If $k = 0$ (meaning slow convergence has taken place at the coarsest grid G^0), go back to step (b) (to continue relaxation nevertheless, since on G^0 relaxation is very inexpensive. If, however, the problem is indefinite, then slow rate of *divergence* may occur, in which case the G^0 problem should be solved directly. This is as inexpensive as relaxation, but requires additional programming. See Section 4.1 below). If $k > 0$, lower k by 1 (to compute correction on the next, coarser level). Compute f^k and ϕ^k on this new level, using (4.2), put $v^k = 0$ as the starting approximation, and go to step (b).

(e) If $k = M$ (convergence has been obtained on the finest level), the algorithm is terminated. If $k < M$ (v^k has converged and is ready to serve as a correction to v^{k+1}), put

$$v^{k+1} \leftarrow v^{k+1} + I_k^{k+1}v^k. \tag{4.5}$$

Then advance k by 1 (to resume computations at the finer level) and go to step (b).

The storage required for this algorithm is only a fraction more than the number of locations, $2n$ say, required to store u^M and F^M on the finest grid. Indeed, for a d-dimensional problem, a storage of roughly $2n/2^d$ locations is required to store v^{M-1} and f^{M-1}, the next level requires $2n/2^{2d}$, etc. The total for all levels is

$$2n(1 + 2^{-d} + 2^{-2d} + \cdots) \leqslant 2n2^d/(2^d - 1). \tag{4.6}$$

(In the FAS version below, a major reduction of storage area is possible through segmental refinement. See Section 7.5.)

4.1. *Indefinite Problems and the Size of the Coarsest Grid.* If, on any grid G^k, the boundary-value problem (4.1) is a nondefinite elliptic problem, with eigenvalues

$$\lambda_1^k \leqslant \lambda_2^k \leqslant \cdots \leqslant \lambda_l^k < 0 < \lambda_{l+1}^k \leqslant \lambda_{l+2}^k \leqslant \cdots, \tag{4.7}$$

and with the corresponding eigenfunctions $V_1^k, V_2^k, \ldots, V_l^k, V_{l+1}^k, \ldots$, then it cannot be solved by straight relaxation. Any relaxation sweep will reduce the error components in the space spanned by $V_{l+1}^k, V_{l+2}^k, \ldots$, but will *magnify* all components in the span of $V_1^k, V_2^k, \ldots, V_l^k$. A multi-grid solution, however, is not seriously affected by this magnification, provided the magnified components are suitably reduced by the coarse-grid corrections. This will usually be the case, since these components are basically of low frequency and are well approximated on coarser grids. But care should be taken regarding the coarsest grid:

On the coarsest grid, an indefinite problem should be solved directly (i.e., not by relaxation of any kind. Semi-iterative solutions, like Newton iterations for nonlinear problems, are, of course, permissible). Furthermore, *this grid should be fine enough to provide rough approximation to the first* $(l+1)$ *eigenfunctions*, so that

$$0 < \lambda_j^{k+1}/\lambda_j^k < \frac{2}{w_k} \qquad (1 \leqslant j \leqslant l+1,\ 1 \leqslant k \leqslant M-1), \tag{4.8}$$

where w_k is an "under-interpolation" factor that should multiply I_k^{k+1} in (4.5) above when the usual value ($w_k = 1$) does not satisfy (4.8). This means that G^0 should contain at least $O(l)$, probably $2l$, points. Also, G^0 should be just fine enough to still have smoothing capability at any finer level G^k. For example, if SOR relaxations with $\omega < \omega_c$ are used, h_0 should satisfy (see [4] or Section 3 in [3])

$$\mathrm{Re}\{B(\theta, h)/b_0(h)\} > 0 \qquad (0 \leqslant h \leqslant h_1), \tag{4.9}$$

where $B(\theta, h)$ is the symbol of L^h (see (A.3) in Appendix A) and $b_0(h)$ is its central coefficient.

Usually, G^0 can still be coarse enough to have the direct solution of its equations still far less expensive than, say, one relaxation sweep over the finest grid, so that the indefinite problem is solved with the same overall efficiency as definite problems.

5. The FAS (Full Approximation Storage) Algorithm. In the FAS mode of the multi-grid algorithms, instead of storing a correction v^k (designed to correct the finer-level approximation u^{k+1}), the idea is to store the full current approximation u^k, which is the sum of the correction v^k and its base approximation u^{k+1}:

$$u^k = I_{k+1}^k u^{k+1} + v^k \quad (k = 0, 1, \ldots, M-1). \tag{5.1}$$

In terms of these full-approximation functions, we can rewrite the correction equations (4.1)–(4.3) as***

$$L^k U^k = \bar{F}^k, \qquad \Lambda^k U^k = \bar{\Phi}^k, \tag{5.2}$$

***Again we distinguish between the notation U^k used to write the equations and the computed approximation u^k. Equation (5.2), for $k < M$, is not equivalent to (2.2), although they both use the notation U^k.

where

$$\begin{aligned} \bar{F}^k &= L^k(I^k_{k+1}u^{k+1}) + I^k_{k+1}(\bar{F}^{k+1} - L^{k+1}u^{k+1}), \\ \bar{\Phi}^k &= \Lambda^k(I^k_{k+1}u^{k+1}) + I^k_{k+1}(\bar{\Phi}^{k+1} - \Lambda^{k+1}u^{k+1}), \quad (k = 0, 1, \ldots, M-1), \end{aligned} \tag{5.3}$$

and where for $k = M$ we have the original problem, i.e.,

$$\bar{F}^M = F^M, \qquad \bar{\Phi}^M = \Phi^M. \tag{5.4}$$

For linear problems, Eqs. (5.2)–(5.4) are exactly equivalent to (4.1)–(4.3). The advantage of the FAS mode is that Eqs. (5.2)–(5.4) apply equally well to *nonlinear* problems. To see this, consider for instance the nonlinear equation $L^M U^M = F^M$ given on the finest grid. Given an approximate solution u^M we can still improve it by relaxation sweeps, with smoothing rates $\bar{\mu}$ (varying over the domain, but still reliably estimated by mode analyses, applied locally to the linearized-frozen equation). As in the linear case, the smoothed-out functions are the residual $f^M = F^M - L^M u^M$ and the correction $U^M - u^M$. Therefore, the equation that can be approximated on coarser grids is the residual equation $L^M U^M - L^M u^M = f^M$. Its coarser-grid approximation is

$$L^{M-1}U^{M-1} - L^{M-1}I^{M-1}_M u^M = I^{M-1}_M f^M, \tag{5.5}$$

which is the same as (5.2) for $k = M - 1$. In interpolating U^{M-1} (or a computed approximation u^{M-1}) back to G^M, we should actually interpolate $U^{M-1} - I^{M-1}_M u^M$, because this is the coarse-grid approximation to the smoothed-out function $U^M - u^M$. Similarly, in interpolating an (approximate) solution u^k of (5.2) to the finer grid G^{k+1}, the polynomial interpolation should operate on the correction. Thus the interpolation is

$$u^{k+1} \leftarrow u^{k+1} + I^{k+1}_k(u^k - I^k_{k+1}u^{k+1}), \tag{5.6}$$

which is equivalent to (4.5). Note that generally,

$$I^{k+1}_k I^k_{k+1} u^{k+1} \neq u^{k+1}.$$

The FAS (Cycle C) algorithm is the same algorithm as in Section 4, with the FAS equations (5.2)–(5.4) replacing (4.1)–(4.3), and with (5.6) replacing (4.5). It is flowcharted in Figure 3.

The FAS mode has several important advantages: It is suitable for general nonlinear problems, with the same procedures (relaxation and interpolation routines) used at all levels. Thus, for example, only one relaxation routine need be written. Moreover, this mode is suitable for *composite grids* (nonuniform grids created by increasingly finer levels being defined on increasingly smaller subdomains; see Section 7.2), which is the basis for grid adaptation on one hand, and segmental refinement (see Section 7.5) on the other hand. Generally speaking, the basic feature of the FAS mode is that the function stored on a coarse grid G^k coincides there with the fine-grid solution: $u^k = I^k_M u^M$. This enables us to manipulate accurate solutions on coarse grids.

The storage required for the FAS algorithm is again given by (4.6). With segmental refinement (Section 7.5) it can be reduced far below that, even to $O(\log n)$.

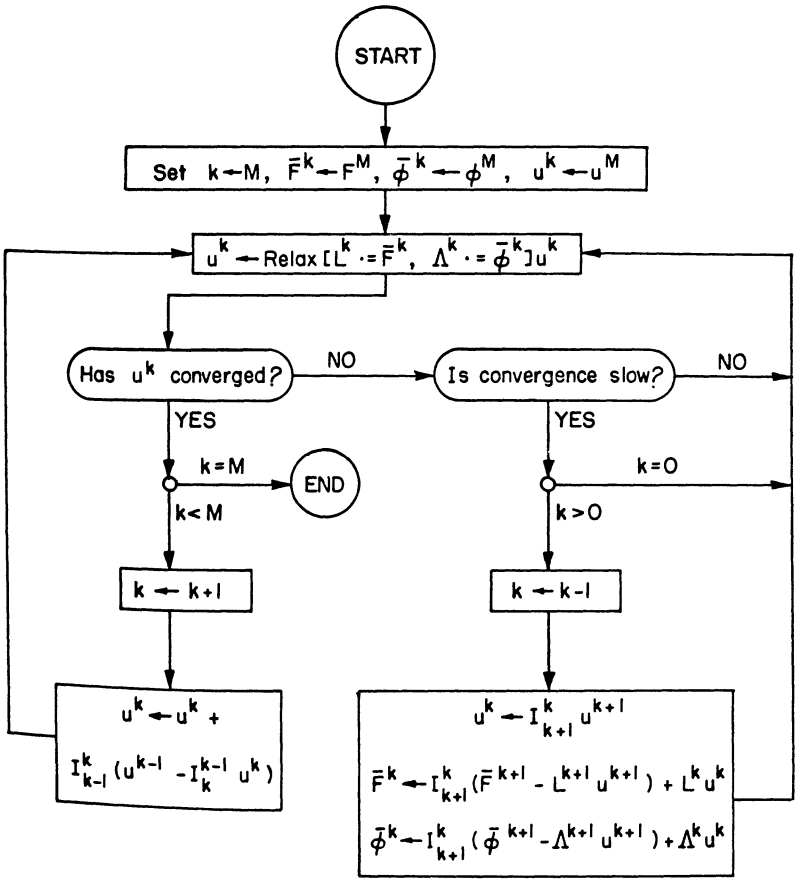

FIGURE 3. *Cycle* C, *Full Approximation Storage*

An important by-product of the FAS mode is a good *estimate for the truncation error*, which is useful in defining natural stopping criteria (see Section A.8) and grid-adaptation criteria (Section 8.3). Indeed, for any $k < m \leqslant M$ it can easily be shown (by induction on m, using (5.2)–(5.3)) that

$$\begin{aligned} \bar{F}^k - I_m^k \bar{F}^m &= L^k(I_m^k u^m) - I_m^k L^m u^m, \\ \bar{\Phi}^k - I_m^k \bar{\Phi}^m &= \Lambda^k(I_m^k u^m) - I_m^k \Lambda^m u^m, \end{aligned} \tag{5.7}$$

which are exactly the G^m approximations to the G^k truncation errors.

A slight disadvantage of the FAS mode is the longer calculation required in computing $\bar{F}^k$, almost twice longer than calculating f^k in the former (Correction Storage) mode. This extra calculation is equivalent to one extra relaxation sweep on G^k, but only for $k < M$, and is about 5% to 10% of the total amount of calculations. Hence, for linear problems on uniform grids, the CS mode is slightly preferable.

6. Performance Estimates and Numerical Tests.

6.1. *Predictability.* An important feature of the multi-grid method is that, although iterative, its total computational work can be predicted in advance by local

mode (Fourier) analysis. Such an analysis, which linearizes and freezes the equations and ignores distant boundaries, gives a very good approximation to the behavior of high-frequency components (since they have short coupling range), but usually fails to approximate the behavior of the lowest frequencies (which interact at long distances). The main point here, however, is that these lowest frequencies may indeed be ignored in the multi-grid work estimates, since their convergence is obtained on coarser grids, where the computational work is negligible. The purpose of the work on the finer grids is only to converge the high frequencies. Thus, the mode-analysis predictions, although not rigorous, are likely to be very realistic. In fact, these predictions are in full agreement with the results of our computational tests. This situation is analogous to the situation in time dependent problems, where precise stability criteria are derived from considering the corresponding linearized-frozen-unbounded problems. (See page 91 in [30].) Rigorous convergence estimates, by contrast, have no use in practice. Except for very simple situations (see Appendix C and Section 10) they yield upper bounds to the computational work which are several orders of magnitude larger than the actual work.

The predictability feature is important since it enables us to *optimize* our procedures (cf. Section 3.3, Appendix A). It is also indispensable in *debugging* multi-grid programs.

6.2. *Multi-Grid Rates of Convergence.* To get a convenient measure of convergence per unit work, we define as our *Work Unit* (WU) the computational work in one relaxation sweep over the finest grid G^M. The number of computer operations in such a unit is roughly wn, where n is the number of points in G^M and w is the number of operations required to compute the residual at each point. (In parallel processing the count should, of course, be different. Also, the work unit should be further specified when comparing different discretization and relaxation schemes.) If the mesh-size ratio is $\hat{\rho} = h_{k+1}/h_k$ and the problem's domain is d-dimensional, then a relaxation sweep over G^{m-j} costs approximately $\hat{\rho}^{dj}$ WU's (assuming the grids are co-extensive, unlike those in Section 7).

Relaxation sweeps make up most of the multi-grid computational work. The only other process that consumes any significant amount of computations is the I_k^{k-1} and I_{k-1}^k interpolations. It is difficult to measure them precisely in WU's, but their total work is always considerably smaller than the total relaxation work. In the example in Appendix B, the interpolation work is about 20% of the relaxation work. Usually the percentage is even lower, since relaxing Poisson problems is particularly inexpensive. To unify our estimates and measurements we will therefore *define the multi-grid convergence factor* $\mathring{\mu}$ *as the factor by which the errors are reduced per one WU of relaxation, ignoring any other computational work* (which is never more than 30% of the total work).

The multi-grid convergence factor may be estimated by a full local mode analysis. The following is a simplified analysis, which gives a good approximation. We assume that a relaxation sweep over any grid G^k affects error components $e^{i\Theta\cdot x}$ only in the range $\pi/h_{k-1} \leqslant |\Theta| \leqslant \pi/h_k$, where

$$(6.1)\qquad \Theta = (\Theta_1, \Theta_2, \ldots, \Theta_d), \qquad \Theta \cdot x = \sum_{j=1}^{d} \Theta_j x_j, \qquad |\Theta| = \max_{1 \leqslant j \leqslant d} |\Theta_j|.$$

(The θ/h_k of Section 3 and Appendix A is denoted here by Θ, to unify the discussion of all levels.) In fact, if a proper interpolation scheme is used (see Section A.2), only components in the range $|\Theta| \leqslant (1 + \epsilon)\pi/h_{k-1}$, say, are affected by interactions with coarser grids. But if proper residual-weighting is also used (to make $\bar{\sigma} = 1$; cf. Section A.4), then the combined action of the coarse-grid correction cycles and the G^k relaxation sweeps yield convergence rates which are slowest at $|\Theta| = \pi/h_{k-1}$ (cf. Appendix A). For such Θ the coarse-grid cycles have neutral effect, since $\bar{\sigma} = 1$, hence the convergence rate is indeed as affected only by the G^k relaxation sweeps.

One relaxation sweep over G^k reduces the error components in the range $\pi/h_{k-1} \leqslant |\Theta| \leqslant \pi/h_k$ by the smoothing factor $\bar{\mu}$. (See Section 3. If the smoothing factor near a boundary is slower than $\bar{\mu}$, which is not the usual case, smoothing may be accelerated there by partial relaxation sweeps–cf. Section A.9.) Thus a multi-grid cycle with s relaxation sweeps on each level reduces all error components by the factor $\bar{\mu}^s$. The amount of Work Units expended in these sweeps is

$$s + s\hat{\rho}^d + s\hat{\rho}^{2d} + \cdots + s\hat{\rho}^{(M-1)d} < s/(1 - \hat{\rho}^d).$$

Hence, the multi-grid convergence factor is

$$(6.2)\qquad \mathring{\mu} = \bar{\mu}^{(1-\hat{\rho}^d)},$$

which is not much bigger than $\bar{\mu}$. In case $\bar{\sigma} > 1$, the effective smoothing factor $\tilde{\mu}$ (see (A.8)) should replace $\bar{\mu}$ in this estimate.

Estimate (6.2) is not rigorous, but is simple to compute and very realistic. In fact, numerical experiments (Sections 6.4–6.5) usually show slightly faster (smaller) factors $\mathring{\mu}$, presumably because the worst combination of Fourier components is not always present.

The theoretical multi-grid convergence factors, for various representative cases, are summarized in Table 1.

Explanations to Table 1. The first column specifies the difference operator and the dimension d. Δ_h denotes the central second-order ($(2d + 1)$-point) approximation, and $\Delta_h^{(4)}$ the fourth-order ($(4d + 1)$-point "star") approximation, to the Laplace operator. Δ_h^2 is the central 13-point approximation to the biharmonic operator. The operators ∂_x, ∂_y, ∂_{xx} and ∂_{yy} are the usual central second-order approximations to the corresponding partial-differential operators. ∂_x^- is the backward approximation. Upstream differencing is assumed for the inertial terms of the Navier-Stokes equations; central differencing for the viscosity terms, forward differencing for the pressure terms, and backward differencing for the continuity equation. Rh is the Reynolds number times the mesh-size.

The second column specifies the relaxation scheme and the relaxation parameter ω. SOR is Successive Over Relaxation, which for $\omega = 1$ is the Gauss-Seidel relaxation. xLSOR (yLSOR) is Line SOR, with lines in the x (y) direction. yLSOR +, yLSOR − and yLSORs indicate, respectively, relaxation marching forward, backward and symmetrically (alternately forward and backward). CSOR means Collective SOR (see

Section 3 in [3]) and the attached ω's are ω_1 for the velocity components and ω_2 for the pressure. "Downstream" means that the flow direction and the relaxation marching direction are the same; "upstream" means that they are opposite. **ADLR** denotes Alternating Direction Line Relaxation (a sweep of x**LSOR** followed by a sweep of y**LSOR**). **SD** is Simultaneous Displacement (Jacobi) relaxation, **WSD** is Weighted Simultaneous Displacement with the optimal weights as specified in Section 3.3 above (and with other weights, to show the sensitivity). **WSD**Δ (for Δ_h^2) is like **WSD**, except that residuals are computed in less operations by making first a special pass that computes $\Delta_h u$. y**LSD** is y-lines relaxation with simultaneous displacement, **ADLSD** is the corresponding alternating-direction (y**LSD** alternating with x**LSD**) scheme.

TABLE 1. *Theoretical smoothing and MG-convergence rates*

L_h	d	Relax. Scheme	ω	$\hat{\rho}$	$\bar{\mu}$	$\mathring{\mu}$	$\lvert \ell n \mathring{\mu} \rvert^{-1}$	add	mult	W_M
Δ_h	1	SOR	1	1:3	.557	.693	2.73	2	1	9.0
				1:2	.477	.668	2.49	3	2	6.9
				2:3	.378	.723	3.08	3	2	7.5
	2	SOR	1	1:3	.667	.697	2.77	4	1	6.8
			.8	1:2	.552	.640	2.24	5	2	4.1
			1		.500	.595	1.92	4	1	3.5
			1.2		.552	.640	2.24	5	2	4.1
			1	2:3	.400	.601	1.96	4	1	2.9
		LSOR	1	1:2	.447	.547	1.66	8	4	3.1
		ADLR	1		.386	.490	1.40	8	4	2.6
			.8		.456	.555	1.70	8	4	3.1
		SD	.8	1:2	.600	.682	2.61	5	2	4.8
		WSD	1.17, .195		.220	.321	0.88	9	3	1.6
			1.40, .203		.506	.600	1.96	9	3	3.6
	3	SOR	1	1:3	.738	.746	3.42	6	1	7.8
				1:2	.567	.608	2.01	6	1	3.7
				2:3	.441	.562	1.73	6	1	2.0
$\Delta_h^{(4)}$	2	SOR	.8	1:2	.581	.665	2.46	9	3	9.1
			1		.534	.625	2.13	8	2	7.9
			1.2		.582	.666	2.46	9	3	9.1
		LSOR	1		.484	.580	1.84	14	7	6.8
	3	SOR	1		.596	.636	2.21	12	2	7.0
$\partial_{xx} + 2\partial_x\partial_y + \partial_{yy}$	2	SOR	1	1:2	. 62	.699	2.79	8	2	5.2
		LSOR,ADLR			.447	.547	1.66	12	5	3.1
Δ_h^2	2	SOR	1	1:2	.802	.847	6.04	12	3	11.1
			1	2:3	.666	.798	4.43	12	3	6.5
		WSD	1.552, .353	1:2	.549	.638	2.22	17	5	4.1
			1.4 , .353		1.03	div.	div.	17	5	div.
		WSDΔ	1.552, .353		.549	.638	2.22	14	4	4.1
NAVIER - STOKES		CSOR								
Rh = 0	2	downstr.	1, .5	1:2	.800	.846	5.98	18	6	11.0
any			1, .5		.800	.846	5.98	33	16	11.0
100			1.1, .5		1.73	div.	div.	33	16	div.
100			.8, .5		.93	.947	18.7	33	16	34.5
10		upstream	1, .5		.884	.912	10.8	33	16	20.0
100			1, .5		.994	.995	220.	33	16	400.
100			.8, .5		.984	.988	83.	33	16	150.
0	3	downstr.	1, .5		.845	.863	6.79	33	8	10.7
any			1, .5		.845	.863	6.79	60	25	10.7
10		upstream	1, .5		.874	.889	8.49	60	25	13.4
100			1, .5		.989	.990	100.	60	25	160.
STOKES' (Rh = 0)		SOR	1,.33		.707					

TABLE 1 (*Continued*). *Here* $d = 2$, $\hat{\rho} = 1 : 2$

L_h	Relax. Scheme	ω	$\bar{\mu}$
$\partial_{xx} + \varepsilon\partial_{yy}$, $\varepsilon \ll 1$	SOR, xLSOR	any	$1 - O(\varepsilon)$
$a\partial_{xx} + c\partial_{yy}$ $(q = \min(\frac{a}{c}, \frac{c}{a}))$	yLSOR	1	$\max(5^{-1/2}, \frac{a}{a+2c})$
	ADLR		$5^{-1/4}(1+2q)^{-1/2}$
	SD, yLSD, ADLSD	1	1
	SD	$(2q+2)/(3q+2)$	$(q+2)/(3q+2)$
	yLSD	$(2a+2c)/(2a+3c)$	$(2a+c)/(2a+3c)$
	ADLSD	2/3, 2/3	$\leq 3^{-1/2} = .577$
$\Delta_h - \frac{\eta}{h}\partial_x$	yLSOR	1	$\max\left(\frac{1-\eta}{3-\eta}, \left[\frac{1-\eta+\eta^2/4}{5+\eta+\eta^2/4}\right]^{1/2}\right)$
$\Delta_h - \frac{\eta}{h}\partial_x^-$ $(\eta > 0)$	yLSOR+	1	$\max\left(\frac{1}{3}, [5+6\eta+2\eta^2]^{-1/2}\right)$
	yLSOR-		$\max\left(\frac{1}{3}, \left\lvert\frac{1+\eta}{2+\eta+i}\right\rvert\right)$
	yLSORs		$\leq 3^{-1/2} = .577$
Navier - Stokes with large Rh in 2 or 3 dimensions	SOR (pressure corrected by the continuity equation), downstream or upstream, with any relaxation parameters.		$\geq 1 - \frac{2}{Rh}$

The next columns list $\hat{\rho} = h_k : h_{k+1}$ (see discussion below), the smoothing factor $\bar{\mu}$ as defined by (3.8), and $\overset{\circ}{\mu}$, calculated by (6.2). We also list the *multi-grid convergence rate* $|\log \overset{\circ}{\mu}|^{-1}$, which is the theoretical number of relaxation Work Units required to reduce the error by the factor e, and W_M, the overall multi-grid computational work (see Section 6.3). To make comparisons of different schemes possible, we also list, for each case, the number of operations per grid point per sweep. This number times n (the number of points in G^M) gives the number of operations in a Work Unit. We list only the basic number of additions and multiplications (counting shifts as multiplications), thus ignoring the operations of transferring information, indexing, etc., which may add up to a significant amount of operations, but which are too computer- and program-dependent to be specified. Also, we assumed that the right-hand sides f^k, including f^M, are stored in the most efficient form (e.g., $h^2 f^M$ is actually stored). Note that the SOR operation count is smaller for $\omega = 1$ (Gauss-Seidel) than for any other ω.

For Navier-Stokes equations with CSOR we found $\bar{\mu}$ to be largest when relaxing upstream, and smallest when relaxing downstream. We also found that in marching alternately back and forth, the worst overall smoothing rate is for flows aligned with the relaxation, i.e., flows for which this relaxation is alternately upstream and downstream. The worst $\bar{\mu}$ is therefore $(\bar{\mu}_u \bar{\mu}_d)^{1/2}$, where $\bar{\mu}_u$ and $\bar{\mu}_d$ are, respectively, the "upstream" and "downstream" values shown in the table.

Numbers in this table were calculated by Allan S. Goodman, at IBM Thomas J. Watson Research Center. A more extensive list is in preparation.

Mesh-Size Ratio Optimization. Examining Table 1, and many other unlisted examples, it is evident that the mesh-size ratio $\hat{\rho} = 1 : 2$ is close to optimal, yielding almost minimal $|\log \mathring{\mu}|^{-1}$ and minimal W_M. This ratio is more convenient and more economical in the interpolation processes (which are ignored in the above calculations) than any other efficient ratio. In practice, therefore, *the ratio* $\hat{\rho} = 1 : 2$ *should always be used*, giving also a very desirable standardization.

6.3. *Overall Multi-Grid Computational Work.* Denote by W_M the computational work (in the above Work Units) required to solve the G^M problem ((2.2), $k = M$) to the level of its truncation errors τ^M (cf. Section A.8). If the problem is first solved on G^{M-1} to the level τ^{M-1}, and if the correct order of interpolation is used to interpolate the solution to G^M (so that unnecessary high-frequencies are not excited, cf. Section A.2, and in particular (A.7) for $i = 1$), then the residuals of this first G^M approximation are $O(\tau^{M-1})$. The computational work required to reduce them to $O(\tau^M)$ is $\log O(\tau^M/\tau^{M-1})/\log \mathring{\mu}$. Hence,

$$W_M = W_{M-1} + \log \frac{\tau^M}{\tau^{M-1}} \Big/ \log \mathring{\mu}. \tag{6.3}$$

Similarly, we can solve the G^{M-j} problem expending work

$$W_{M-j} = W_{M-j-1} + \hat{\rho}^{jd} \log \frac{\tau^{M-j}}{\tau^{M-j-1}} \Big/ \log \mathring{\mu} \tag{6.4}$$

(since a G^{M-j} work unit is $\hat{\rho}^{jd}$ times the G^M unit). If we use p-order approximations, then

$$\tau^k/\tau^{k-1} \leqslant O(h_k^p)/O(h_{k-1}^p) = O(\hat{\rho}^p). \tag{6.5}$$

Hence, using (6.4) for $j = 0, 1, 2, \ldots, M-1$ and neglecting W_0,

$$W_M \leqslant (1 + \hat{\rho}^d + \hat{\rho}^{2d} + \cdots) p \log \hat{\rho}/\log \mathring{\mu}.$$

Or, by (6.2),

$$W_M \leqslant (p \log \hat{\rho})/((1 - \hat{\rho}^d)^2 \log \bar{\mu}). \tag{6.6}$$

(The same $\hat{\rho}$ was assumed in computing the first approximation and in the improvement cycles. This of course is not necessary.)

Typical values of this theoretical W_M are shown in Table 1 above. In actual computations a couple of extra Work Units are always expended in solving a problem, because we cannot make nonintegral numbers of relaxation sweeps or MG cycles, and also because we usually solve to accuracy below the level of the truncation errors.

For 5-point Poisson problems, for example, the following procedure gives a G^M solution with residuals smaller than τ^M. (i) Obtain u^{M-1} on G^{M-1}, with residuals smaller than τ^{M-1}. (ii) Starting with the *cubic* interpolation $u^M \leftarrow I^M_{M-1}u^{M-1}$ (preferably by using the difference operator itself; cf. [7]), make a MG correction cycle such as Cycle C with $\eta = 0$ (i.e., switching to G^{k-1} after two sweeps on G^k), with I^{k-1}_k transfer by injection (cf. Section A.4) and I^k_{k-1} by linear interpolation, and with "convergence" on G^k defined as obtained after the first sweep following a return from G^{k-1}. A precise count shows Step (ii) to require $30n + O(n^{1/2})$ operations, where n is the number of points in G^M. Thus, the total number of operations is

$$\left(1 + \frac{1}{4} + \frac{1}{16} + \cdots\right) 30n + O(n^{1/2}) \leqslant 40n + O(n^{1/2}).$$

Incidentally, none of these operations is a full multiplication: only additions and shifts (multiplications or divisions by 2 or 4) are used. The theoretical W_M for this problem (sixth line in Table 1) amounts to only $17.5n$ operations, since it ignores interpolation work ($10.3n$ operations in the above procedure) and allows nonintegral numbers of sweeps and cycles. In fact, numerical tests showed the above algorithm to yield residuals considerably below the truncation errors. (The only cases in which the residuals approached 50% of the truncation errors were cases with high smoothness, in which the correct MLAT discretization would be different; namely, of higher order. (Cf. Section 8 and the remark following formula (A.7).) If one is interested in still smaller residuals, then another MG correction cycle can be added to the above algorithm. This will require $20n$ more operations and will make the residuals much smaller than (typically 2% of) the truncation errors. For $n \geqslant 500$, a program implementing the above algorithm runs less than $40n$ microseconds on CDC CYBER 173.

6.4. *Numerical Experiments: Elliptic Problems.* A typical numerical experiment is shown in Appendix B, including the FORTRAN program and the computer output. The output shows a multi-grid convergence factor

$$\mathring{\mu} = \left(\frac{.009051}{28.1}\right)^{1/12.92} = .537$$

which is close to, and slightly faster than, the theoretical value $\mathring{\mu} = .595$ shown in Table 1.

Many numerical experiments with various elliptic difference equations in various domains were carried out at the Weizmann Institute in 1970–1972, with the collaboration of Y. Shiftan and N. Diner. Some representative results were reported in [2], and many others in [11]. These experiments were made with other variants of the multigrid algorithm (variants A and B), but their convergence factors agree with the same theoretical rates $\mathring{\mu}$. The experiments with equations of the form $aU_{xx} + cU_{yy}$, with $a \gg c$, showed poor convergence rates, since the relaxation scheme used was Gauss-Seidel, and not the appropriate line relaxation (cf. Section 3.1). Some of these rates were better than predicted by the mode analysis, because the grids were not big enough to show the worst behavior. The convergence rates found in the experiments with the biharmonic equation were also rather poor (although nicely bounded, independently of the grid size), again because we used Gauss-Seidel relaxations and injections

instead of the appropriate schemes (cf. Sections 3.3 and A.4). All these points were later clarified by mode analyses, which fully explain all the experimental results. In solving the stationary Navier-Stokes equations, as reported in [2], SOR instead of CSOR was employed (cf. Table 1 above), and an additional over-simplification was done by using, in each multi-grid cycle, values of the nonlinear terms from the previous cycle, instead of using the FAS scheme (Section 5).

Nevertheless, these experiments did clearly demonstrate important features of the multi-grid method: The rate of convergence was essentially insensitive to several factors, including the shape of the domain Ω, the right-hand side F (which has some influence only at the first couple of cycles; cf. Section A.2) and the finest mesh-size h_M (except for mild variations when h_M is large). The experiments indicated that the order I of the interpolations I_{k-1}^k should be the order of the elliptic equation, as shown in Section A.2 below. (Note that in [2] the order was defined as the degree l of the polynomial used in the interpolation, whereas here $I = l + 1$.)

More numerical experiments are now being conducted at the Weizmann Institute in Israel and at IBM Research Center in New York, and will be reported elsewhere. In this article they are briefly mentioned in Sections 6.3, 7.2, A.4, A.6, A.7. We will separately report here only an extreme case of the multi-grid tests–the solution of transonic flow problems.

6.5. *Numerical Experiments: Transonic Flow Problems.* These experiments were started in 1974 at the Weizmann Institute with J. L. Fuchs, and recently conducted at the NASA Langley Research Center in collaboration with Dr. Jerry South while the present author was visiting the Institute for Computer Applications in Science and Engineering (ICASE). They are preliminarily reported in [12], and will be further reported elsewhere. One purpose of this work was to examine the performance of the multi-grid method in a problem that is not only nonlinear, but more significantly, is also of mixed (elliptic-hyperbolic) type and contains discontinuities (shocks).

We considered the transonic small-disturbance equation in conservation form

$$[(K - \bar{K}\phi_x)\phi_x]_x + c\phi_{yy} = 0, \tag{6.7}$$

for the velocity disturbance potential $\phi(x, y)$ outside an airfoil. Here $K = (1 - M_\infty^2)/\tau^{2/3}$, $\bar{K} = \frac{1}{2}(\gamma + 1)M_\infty^2$, M_∞ is the free-stream Mach number, and $\gamma = 1.4$ is the ratio of specific heats. τ is the airfoil thickness ratio, assumed to be small. $c = 1$, unless the y coordinate is stretched. The airfoil, in suitably scaled coordinates, is located at $\{y = 0, |x| \leqslant \frac{1}{2}\}$, and we consider nonlifting flows, so that the problem domain can, by symmetry, be reduced to the half-plane $\{y \geqslant 0\}$, with boundary conditions

$$\phi(x, y) \longrightarrow 0 \quad \text{as} \quad x^2 + y^2 \longrightarrow \infty, \tag{6.8}$$

$$\phi_y(x, 0) = \begin{cases} 0 & \text{for } |x| > \frac{1}{2}, \\ F'(x) & \text{for } |x| < \frac{1}{2}, \end{cases} \tag{6.9}$$

where $\tau F(x)$ is the airfoil thickness function which we took to be parabolic. Equation (6.7) is of hyperbolic or elliptic type depending on whether $K - 2\bar{K}\phi_x$ is negative or positive (supersonic or subsonic).

The difference equations we used were essentially the Murman's conservative scheme ([9]; for a recent account of solution methods, see [8]), where the main idea is to adaptively use upstream differencing in the hyperbolic region and central differencing in the elliptic region, keeping the system conservative. For relaxation we used vertical (y) line relaxation, marching in the stream direction. The multi-grid solution was programmed both in the CS (Section 4) and the FAS (Section 5) modes, with practically the same results. We used cubic interpolation for I_k^{k+1} and injection for I_k^{k-1}.

Local mode analysis of the linearized-frozen difference equations and vertical-forward line relaxation gives the smoothing factor

$$\bar{\mu} = \max\left\{\left|\frac{b_+}{b_+ + b_- + ib_-}\right|, \frac{b_+}{2c + b_+}\right\}, \qquad b_\pm(x) = K - 2\bar{K}\phi_x\,(x \pm h/2), \tag{6.10}$$

at elliptic (subsonic) points, and $\bar{\mu} = 0$ at supersonic points. We were interested in cases where $K < 1$ and $\phi_x \geqslant 0$, and hence, in smooth elliptic regions ($b_+ \approx b_-$) without coordinate stretching we get $\bar{\mu} \approx 1/|2 + i| = 0.45$ and $\mathring{\mu} = \bar{\mu}^{3/4} = 0.55$.

The actual convergence factors, observed in our experiments with moderately supercritical flows ($M_\infty = 0.7$ and $M_\infty = 0.85$, $\tau = 0.1$) on a 64×32 grid, were $\mathring{\mu} = 0.52$ to 0.53, just slightly faster than the theoretical value. (See detailed output in [12]. The work count in [12] is slightly different, counting also the work in the I_{k+1}^k transition.)

For highly supercritical flows ($M_\infty = 0.95$, $\tau = 0.1$) the MG convergence rate deteriorated, although it was still three times faster than solution by line relaxation alone. The worse convergence pattern was caused by a conceptual mistake in our test for slow convergence (cf. Section A.6). For switching to a coarse grid in a transonic problem, it is not enough that slow reduction per relaxation sweep is shown in the residual norm. Slow change per sweep should also be exhibited in the number of supersonic (or subsonic) points. When this extra test was introduced we obtained fast convergence ($\mathring{\mu} \leqslant .73$) even for $\mu = .98$. Further improvement may be obtained by including partial relaxation sweeps (see Section A.9) in a narrow region behind the shock, where $b_+ \gg b_-$ so that $\bar{\mu}$ is close to 1. We had certain difficulties in the convergence on the coarsest grids, which may indicate the need for the residual weighting (A.12).

Coordinate stretching, which transforms the bounded computational domain to the full half-plane, gave difference equations that again exhibited slow multi-grid convergence rates. This, too, is explainable by the mode analysis. For example, in the regions where the y coordinate is highly stretched, c in (6.7) becomes very small, and hence, $\bar{\mu}$ in (6.10) is close to 1. The theoretical remedies: alternating-direction line relaxations and partial relaxation sweeps. The latter was tried in one simple situation (stretching only the x coordinate), and indeed restored the convergence rate of the corresponding unstretched case.

7. Nonuniform Grids. Many problems require very different resolution in different parts of their domains. Special refinement of the grid is required near singular points, in boundary layers, near shocks, and so on. Coarse grids (with higher approxi-

mation order) should be used where the solution is smooth, or in subdomains far from the region where the solution is accurately needed, etc. A general method for locally choosing mesh-sizes and approximation orders is described in Section 8. An important feature of the method is *adaptivity*: the grid may change during the solution process, adapting itself to the evolving solution. In this section, we propose a method of organizing nonuniform grids so that the local refinement is highly flexible. The main idea is to let the sequence of uniform grids $G^0, G^1, \ldots, G^M$ (cf. Section 2) be open-ended and noncoextensive (i.e., finer levels may be introduced on increasingly smaller subdomains to produce higher local refinement, and coarser levels may be introduced on increasingly wider domains to cover unbounded domains), and, furthermore, to let each of the finer levels be defined in terms of suitable local coordinates. The multi-grid FAS process remains practically as before (Section 5), with similar efficiency. Also discussed is a method which employs this grid organization for "segmental refinement", a multi-grid solution process with substantially reduced storage requirement.

7.1. *Organizing Nonuniform Grids.* How are general nonuniform grids organized for actual computations? There are two popular approaches: One, usually used with the *finite element method*, is to keep the entire system very flexible, allowing each grid point to be practically anywhere. This requires a great deal of bookkeeping: grid points' locations and pointers to neighbors need to be stored; sweeping over the grid is complicated; obtaining the coefficients of the difference equations (or the local "stiffness") may require lengthy calculations, especially where the grid is irregular; and these calculations should be repeated each relaxation sweep, or else additional memory areas should be allocated to store the coefficients. Also, it is more difficult to organize a multi-grid solution on a completely general grid (see, however, Sections 7.3 and A.5), and complete generality is not necessary for obtaining any desired refinement pattern.

Another approach for organizing a nonuniform grid is by a *coordinate transformation*, with a uniform grid being used over the *transformed* domain. On such grids, topologically still rectangular, the multi-grid method can be implemented in the usual way, the lines of G^{k-1} being every other line of G^k. Decisions (stopping criteria, residual weighting, relaxation mode and relaxation directions) should be based on the transformed difference equations. Very often, however, coordinate transformation does not offer enough flexibility. A local refinement is not easy to produce, unless it is a one-dimensional refinement, or a tensor product of one-dimensional refinements. The difficulties are enlarged in adaptive procedures, where it should be inexpensive to change *local* mesh-sizes several times in the solution process. Moreover, the transformation usually makes the difference equation much more complicated (requiring additional storage for keeping coefficients, or additional work in recomputing them every sweep), especially when the transformation does become sophisticated (i.e., adaptive, and not merely a product of one-dimensional transformations), and in particular if higher-order approximations should be used in some or all subdomains.

Thus, be it in the original or in some transformed domain, one would like to have a convenient system for local refinements, with minimal bookkeeping and efficient methods for formulating and solving difference equations. The following system is

proposed (and then generalized, in Sections 7.3, 7.4):

A nonuniform grid is a *union of uniform subgrids,* $G^0, G^1, \ldots, G^M$, with corresponding mesh-sizes $h_0, h_1, \ldots, h_M$. Usually $h_k : h_{k+1} = 2 : 1$ and every other grid line of G^{k+1} is a grid line of G^k. Unlike the description in Section 2, however, the subgrids are not necessarily extended over the same domain. The domain of G^{k+1} may be only part of the domain of G^k (but *not* vice versa). Thus we may have different levels of refinement at different subdomains.

For problems on a bounded domain Ω, several of the first (the coarsest) subgrids may extend to the entire domain Ω. That is, they do not serve to produce different levels of refinement; but they are kept in the system for serving in the multi-grid process of solving the difference equations. G^0 *should be coarse enough* to have its system of difference equations relatively inexpensive to solve (i.e., requiring less than $O(\Sigma n_k)$ operations, where n_k is the number of grid points in G^k. But cf. Section 4.1). The finer subgrids typically extend only over certain subdomains of Ω, not necessarily connected. Generally, G^k *is stretched over those subdomains where the desired mesh-size is* h_k *or less.* Thus, very fine levels (e.g., with $M = 20$, so that $h_M = 2^{-20}h_0$) may be introduced, provided they are limited to suitably small subdomains.

Such a system is very flexible, since grid refinement (or coarsening) is done by extending (or contracting) uniform subgrids. There are several possible ways of storing functions on a (possibly disconnected) uniform grid, allowing for easy grid changes. For example, each string (i.e., connected row or column) of function values can be stored separately, at an arbitrary place in one big storing area, with a certain system of pointers leading from one string to the next. The extra storage area needed for these pointers is small compared with the area needed for storing the function values themselves. One such system, with subroutines for creating, changing and interpolating between the grids, is now under construction, and is described in [26].

If the (original or transformed) *problem's domain is unbounded,* we usually put suitable boundary conditions on some finite, "far enough" artificial boundary. In the present system, we do not have to decide in advance where to place the artificial boundary: We can extend (or contract) the coarsest subgrid(s) as the solution evolves. Moreover, we can add *increasingly coarser levels* ($G^{-1}, G^{-2}, \ldots$) to cover increasingly wider domains, if required by the evolving solution. In this way, we may reach computational domains of large diameter R, by adding only $O(\log R)$ grid points (assuming the desired mesh-size, out at distance r, is proportional to r, or larger. This should usually be the case, especially if appropriate higher-order approximations are used at large distances).

There appears to be a certain waste in the proposed system, as one function value may be stored several times, when its grid point belongs to several levels G^k. This is not the case. First, because the amount of such extra storage is small (less than 2^{-d} of the total storage; see (4.6)). Moreover, the stored values are exactly those needed for the multi-grid process of solution: In fact, in that process, the values stored for different levels at the same grid point are not identical; they only converge to the same value as the process proceeds.

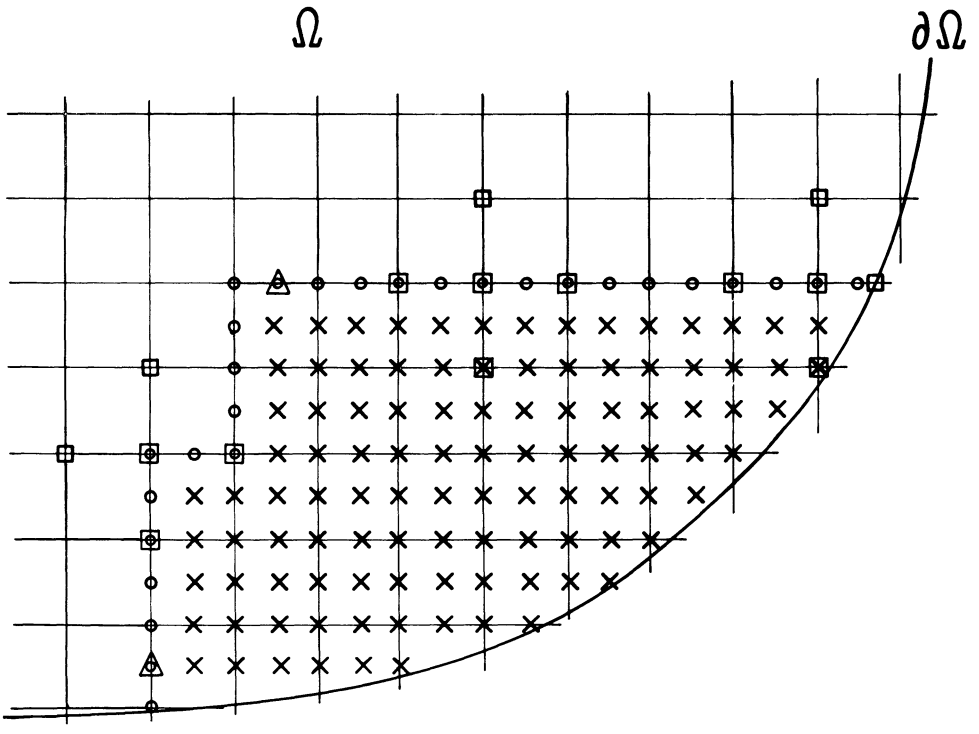

FIGURE 4. *Example of Nonuniform Grid*

A section of the domain Ω and its boundary $\partial\Omega$ is shown, covered with a coarser grid G^k (line intersections) and a finer grid G^{k+1} (crosses and circles). For the case of 5-point (or 9-point "box") difference equations, G^{k+1} inner points are marked with crosses, its outer points with circles. (For convenient interpolation, outer points should lie on G^k lines.) At outer points belonging to G^k, the converged solution satisfies the G^k difference equations, such as the 5-point relations indicated by squares. At other outer points, such as those shown with triangles, the solution is always an interpolation from values at adjacent G^k points. (Note that starting values at outer points should be such that these interpolation relations are satisfied. The FAS interpolation steps will then automatically preserve these relations.)

7.2. *The Multi-Grid Algorithm on Nonuniform Grids.* The following is a description of the modification in the FAS multi-grid algorithm (Section 5) in case of a nonuniform grid with the above structure. The algorithm remains almost the same, except that the difference equations (5.2)–(5.4) are changed to take account of the fact that the levels G^k do not necessarily cover the same domain. Denoting by G^k_m the set of points of G^k which are *inner points* of a finer level G^m (i.e., points where the G^m difference equations are defined†; see Figure 4), the modified form of the difference equations on G^k is

†We use the term "inner", and not "interior", because these points may well be boundary points. Indeed, at boundary points difference equations *are* defined, although they are of a special type, called boundary conditions. The only G^m points where G^m difference equations are not defined are points on or near the *internal* boundary of G^m; i.e., the boundary beyond which the level G^m is not defined, but some coarser levels are. If the grid lines of G^k do not coincide with grid lines of G^m, G^k_m is defined as the set of points of G^k to which proper *interpolation* from inner points of G^m is well defined. For $m > M$, G^k_m is empty.

$$L^k U^k = \bar{F}^k, \quad \Lambda^k U^k = \bar{\Phi}^k, \tag{7.1}$$

where

$$\bar{F}^k = F^k \quad \text{and} \quad \bar{\Phi}^k = \Phi^k \quad \text{in } G^k - G^k_{k+1} \text{ and for } k = M, \tag{7.2}$$

$$\bar{F}^k = F^k_{k+1} \quad \text{and} \quad \bar{\Phi}^k = \Phi^k_{k+1} \quad \text{in } G^k_{k+1}, \tag{7.3}$$

$$F^k_m = I^k_m(\bar{F}^m - L^m u^m) + L^k(I^k_m u^m), \tag{7.4}$$

$$\Phi^k_m = I^k_m(\bar{\Phi}^m - \Lambda^m u^m) + \Lambda^k(I^k_m u^m). \tag{7.5}$$

F^k and Φ^k, as in Section 2, are the G^k approximation to the original right-hand sides F and Φ, respectively.

Observe that, by (7.2)–(7.3), each intermediate level G^k plays a double role: On the subdomain where the finer subgrid G^{k+1} is not defined, G^k plays the role of the finest grid; and the difference equation there is an approximation to the original differential equation. At the same time, on the subdomain where finer subgrids are present, G^k serves for calculating the coarse-grid correction. These two roles are not confused owing to the FAS mode, in which the correction v^k is only implicitly computed, its equation being actually written in terms of the full approximation u^k. In other words, F^k_m may be regarded as the usual G^k right-hand side (F^k), corrected to achieve G^m accuracy in the G^k solution. Indeed

$$F^k_m - I^k_m \bar{F}^m = L^k(I^k_m u^m) - I^k_m(L^m u^m), \tag{7.6}$$

which is the G^m approximation to the G^k truncation error.

The only other modification required in applying Cycle C to nonuniform grids is in the convergence switching criteria. See Section A.10.

When converged, the solution so obtained satisfies Eqs. (2.2) in the inner part of $G^k - G^k_{k+1}$ ($k = 0, 1, \ldots, M$). On *outer* (i.e., noninner) points the solution *automatically* satisfies either a coarser-grid difference equation (if the point belongs to a coarser grid) or a coarser-grid interpolation relation (see Figure 4). Note that, in this procedure, *difference equations should be defined on uniform grids only.* This is an important advantage. Difference equations on equidistant points are much simpler, more accurate. The basic weights for each term (e.g., the weights $(1, -2, 1)$ for the second-order approximation to $\partial^2/\partial x^2$) can be read from small standard tables; whereas on a general grid those weights should be recomputed (or stored) separately for each point, and they are very complicated for high-order approximations.

Another advantage is that the *relaxation sweeps, too, are on uniform grids only.* This simplifies the sweeping, and is particularly important where symmetric and alternating-direction sweeps of line relaxation are required (cf. Section 3).

Numerical experiments indicate that the typical multi-grid convergence factors, measured by the overall error reduction per work unit and predicted by local mode analysis (cf. Section 6), are retained in multi-grid solutions on nonuniform grids. The work unit, though, is somewhat different: It is the computational work of one sweep on *all* levels, not only on G^M, since here G^M may make up only a small part of the points of the final nonuniform grid.

7.3. *Finite-Element Generaliztion.* The structure and solution process outlined above can be generalized in various ways. An important generalization is to employ piecewise uniform, rather than strictly uniform, levels.

Quite often, especially in problems that use finite-element discretizations, the "basic" partition G^0 (e.g., the coarsest triangulation) of the domain is a nonuniform one, but one which is particularly suitable for the geometry of the problem. Finer levels $G^1, G^2, \ldots$, are defined as uniform refinements of that basic level; e.g., $h_k = 2^{-k}h_0$; so that h_k is constant within each basic element.

Having defined the levels G^k in this manner, the rest may in principle be as before: The actual composite grid may use only certain, arbitrary portions of each level; i.e., the actual subgrids G^k need not be coextensive, allowing for adaptive refinements. Coarser levels ($G^{-1}, G^{-2}, \ldots$) may be added if the basic level G^0 is not coarse enough for full-speed multi-grid solution. (Although there is no general algorithm for coarsening a nonuniform G^0, and usually G^0 is coarse enough). Data structures, similar to the uniform case may be used, but should be constructed separately for each basic element (or each set of identical basic elements).

The multi-grid algorithm is the same as in Section 7.2. The discrete equations are thus defined separately for each level. The reproduction of these equations during relaxation is not as convenient as in the strictly uniform case, but still, in the interior of any basic element the equations can readily be read from fixed tables, one table for each set of identical basic elements.

7.4. *Local Transformations.* Another important generalization of the above structure is to subgrids which are defined each in terms of another set of variables. For example, near a boundary or an interface, the most effective local discretizations are made in terms of local coordinates in which the boundary (or interface) is a coordinate line. In particular, with such coordinates it is easy to formulate high-order approximations near the boundary; or to introduce mesh-sizes that are different across and along the interface (or the boundary layer); etc. Usually it is easy to define suitable *local* coordinates, and uniformly discretize them, but it is more difficult to patch together all these local discretizations.

A multi-grid method for patching together a collection of local grids $G_1, G_2, \ldots, G_m$ (each being uniform in its own local coordinates) is to relate them all to a basic grid G_0, which is uniform in the global coordinates and stretches over the entire domain. The relation is essentially as above (Section 7.2); namely, finite-difference equations are separately defined in the inner points of each grid, and the FAS multi-grid process automatically combines them together through its usual interpolation periods.

A Remark: To a given collection of local grids we may have to add intermediate grids to obtain fast multi-grid convergence. That is, if a given local grid G_k is much finer than the basic grid G_0, we have to add increasingly coarser grids, all of them uniform grids in the same local coordinates, such that the coarsest of them has a mesh-size which is (in the global coordinates) nowhere much smaller than the basic mesh-size h_0. Similarly, if the basic global grid G_0 is not coarse enough, the usual multi-grid

sequence of global grids $G^0, G^1, \ldots, G^M = G_0$ should be introduced. Thus, in each set of coordinates we will generally have several grids.

Such a system offers much flexibility. Precise treatment of boundaries and interfaces by the global coordinates is not required. The local coordinates may be changed in the course of computations, e.g., to fit a moving interface. New sets of local coordinates may be introduced (or deleted) as the need arises.

The data structure required for creating, changing and employing such grids is basically again just any data structure suitable for changeable uniform grids. This, however, should be supplemented by tables for the local transformations, such that one can efficiently (i) reproduce the local difference equation, and (ii) interpolate from local to global grid points, and vice versa. See [26].

7.5. *Segmental Refinement.* The multi-grid algorithm for nonuniform grids (Section 7.2) can be useful even in the case of uniform grids, if the computer memory is not sufficiently large to store the finer levels.

"Segmental refinement" is the refinement of one subdomain at a time. To see why and how this is possible, observe that with the FAS mode (Section 5) the full solution u^M is obtained on all grids. But on a coarser grid G^k, the u^M solution satisfies a "corrected" difference equation, with $\overline{F}^k = F_M^k$ replacing F^k. It is therefore not necessary to keep the fine grid, once F_M^k has been computed.

The corrected forcing function F_M^k can be computed by segmental refinement. Refining only one subdomain, one can use the algorithm above (Section 7.2) to obtain a multi-grid solution, including the values of F_M^k in the refined subdomain. Keeping this F_M^k (instead of F^k), one can then discard this refinement, and refine a second subdomain. And so on, through a sequence of subdomains covering the entire domain.

Since subsequent subdomain refinements change the solution everywhere, some further changes are also due in the values of F_M^k on former subdomains. However, at points inner to (and few meshes away from the boundary of) such a former subdomain, these further changes are much smaller than the first correction $F_M^k - F^k$, since they represent changes in the G^k truncation error due to small smooth changes in the solution, while the first correction represents the full G^k truncation error. Thus, if the refinement segments are chosen so that neighboring segments overlap (several mesh intervals into each other), then the further corrections may be ignored. If extra accuracy is desired, another cycle of segmental refinements may be performed. Another way of viewing this technique is to observe that the roll of the finer levels, relative to the coarser ones, is only to liquidate high-frequency error components which cannot be "seen" on the coarser grids. These components have a short (just few mesh-sizes) coupling range, and can therefore be computed at any point by refining only few neighboring meshes.

With this technique one can *operate the multi-grid algorithm almost in its full efficiency, using a storage area which is much smaller than that of the finest grid.* This has been confirmed by preliminary (one-dimensional) numerical tests.

In principle, the required storage area can be reduced to only a constant cube, of J^d locations, on each level (where even $J = 15$ probably offer enough overlap without

substantial reduction in efficiency). Thus, *the overall storage requirement can in principle be reduced to*

$$2J^d\left\{1 + \log \frac{R}{h}\Big/\log J\right\}$$

locations, where h is the finest mesh-size and R is the diameter of the domain. No external memory is needed.

8. Adaptive Discretization Techniques. The previous section described a flexible data structure and solution process which facilitate implementation of variable mesh-sizes h. The difference equations in that process are always defined at inner points of uniform subgrids, which make it easy to employ high and variable approximation orders p. How, then, are mesh-sizes and approximation-orders to be chosen? Should boundary layers, for example, be resolved by the grid? What is their proper resolution? Should we use high-order of approximation at such layers? How to detect such layers automatically? In this section we propose a general framework for automatic selection of h and p in a (nearly) optimal way. In Section 9, we will study some special cases, and show how this proposed system automatically resolves or avoids resolving a thin layer, depending on the alleged goal of the computations.

8.1. *Basic Principles.* We will treat the problem of selecting the discretization parameters h and p (and possibly other parameters, see Section 8.4) as an optimization problem: We will seek to minimize a certain error estimator E, subject to a given amount of computational work W. (Or, equivalently, minimize the work W to obtain a given level E of the error estimator. We will see that the actual control quantity is neither E nor W, but their rate of exchange.) It is important, however, to promptly emphasize that we should not take this optimization too pedantically; it is enough, for instance, to obtain E which is one or two orders of magnitudes larger than the minimum (or, equivalently, to invest work W which is by some fraction more than theoretically needed. Note below that $\log(1/E_{\min})$ is usually proportional to W). Full optimization is not our purpose, is enormously harder and, in fact, is self-defeating, since it requires too much computational work to be invested in controlling h and p. We will aim at having the control work much smaller than the actual numerical work W, using the optimization problem only as a loose directive for sensible discretization.

The Error Estimator E is a functional that estimates, for any given numerical approximation, the overall error in solving the differential boundary-value problem. In principle, such a functional should be furnished whenever a problem is submitted for numerical solution; in practice, it is seldom provided. To have such an estimator depends on having a clear and well-defined idea about *the goal of the computations,* i.e., an idea about what error norm we intend to minimize. Given the goal, even roughly, we can usually formulate E quite easily. We assume that the numerical approximation U^h is in some suitable neighborhood of the true solution (this is a necessary and justifiable assumption; see Section 8.2), so that E can be written as a linear functional

$$E = \int_\Omega G(x)\tau(x)\,dx. \tag{8.1}$$

$\tau(x)$ is a local estimate of the *truncation error*, i.e., the error by which the numerical solution U^h fails to satisfy the differential equation $LU = F$; or, more conveniently, the error by which the differential solution U fails to satisfy the discrete equation $L^h U^h = F^h$. That is,

$$\tau(x) = |LU(x) - L^h U(x)|. \tag{8.2}$$

$G(x)$ is the nonnegative "*error-weighting function*" (or distribution), through which the goal of the computations should be expressed.

The choice of G can be crude. In fact, multiplying G by a constant does not change our optimization problem. Also, we can make large errors, up to one or two orders of magnitudes, in the relative values of G at two different points, since we are content in having E only to that accuracy. What matters is only large changes in G, e.g., near boundaries. For example, if we have a uniformly elliptic problem of order m, and if we are interested in computing good approximations to U and its derivatives up to order l and up to the boundary, then a suitable choice is

$$G(x) = d_x^{m/2-l}, \tag{8.3}$$

where d_x is the distance of x from the boundary. (The formula should be suitably modified near a boundary corner.) This and similar choices of G are easily found by local one-dimensional crude analysis of the relation between a perturbation in the equations and the resulting perturbation in the quantity we wish to approximate. Even though crude, such choice of G would specify our goal much closer than people usually bother to. Moreover, we can change G if we learn that it fails to properly weigh a certain region of the computation; it can serve as a convenient control, conveying our intentions to the numerical discretization and solution.

The Work Functional W. In solving the discrete equations by the multi-grid method, the main overall computational work is the number of Work Units invested in relaxations, times the amount of computations in each Work Unit (see Section 6). If the discretization and relaxation schemes are suitable, the number of Work Units is almost independent of the relaxation parameters h and p. (See e.g., the rate $\mathring{\mu}$ for $\Delta_h^{(4)}$ vs Δ_h in Table 1 above.) Since for our optimization problem we need W only up to a multiplicative constant, we can take into account only the amount of computations in a single Work Unit, i.e., the work in one relaxation sweep over the domain. The local number of grid points per unit volume is $h(x)^{-d}$, and the amount of computation at each grid point is a function $w(p(x))$, where $p(x)$ is the local order of approximation. Hence, we can regard the work functional as being

$$W = \int_\Omega \frac{w(p(x))}{h(x)^d}\, dx. \tag{8.4}$$

Global Optimization Equations. Treating the discretization parameters as spatial variables, $h(x)$ and $p(x)$, the Euler equations of minimizing E for fixed W are

$$\frac{\partial E}{\partial h(x)} + \lambda \frac{\partial W}{\partial h(x)} = 0, \tag{8.5a}$$

$$\frac{\partial E}{\partial p(x)} + \lambda \frac{\partial W}{\partial p(x)} = 0, \tag{8.5b}$$

where λ is a constant (the Lagrange multiplier). It is easily seen that λ is actually the marginal rate of exchange between work and optimal accuracy, i.e.,

$$\lambda = -\frac{dE_{\min}}{dW} = E\frac{d \log 1/E}{dW}, \tag{8.6}$$

and the meaning of (8.5) is that we cannot lower E by trading work (e.g., by taking smaller h at one point and larger at another, keeping W constant, or trading a change in h with a change in p).

Equations (8.5) make some essential simplifications in the optimization problem: They regard h and p as defined at all points $x \in \Omega$. Also, h and p are assumed to be continuous variables, whereas in practice they are discrete. (p should be a positive integer, in some schemes a positive even integer. Values of h are restricted by some grid-organization considerations.) These simplifications are crucial for our approach, and they are altogether justified by the fact that we are content in having only an approximate optimum. The practical aspect, of choosing permissible h and p close to the solution of (8.5), is discussed in Section 8.3. One restriction we should, however, take into account in the basic equations, namely, the restriction

$$p_0 \leqslant p(x) \leqslant p_1(x). \tag{8.7}$$

Without such a restriction, the optimization equations may give values of p which cannot be approximated by permissible values. p_0 is usually 1 or (in symmetric schemes) 2. The upper bound p_1 may express the highest feasible order due to round-off errors; or the highest order for which we actually have appropriate (stable) discretization formulae, with special such restriction near boundaries (hence the possible dependence of p_1 on the position x). With this restriction, Euler's equation (8.5b) should be rewritten as

$$\frac{\partial E}{\partial p(x)} + \lambda \frac{\partial W}{\partial p(x)} \begin{cases} \geqslant 0 & \text{if } p(x) = p_0, \\ = 0 & \text{if } p_0 < p(x) < p_1(x), \\ \leqslant 0 & \text{if } p(x) = p_1(x). \end{cases} \tag{8.8}$$

Local Optimization Equations. Substituting (8.1) and (8.4) into (8.5a) and (8.8), we get the following equations at each point $x \in \Omega$:

$$G\frac{\partial \tau}{\partial h} - \frac{\lambda dw(p)}{h^{d+1}} = 0, \tag{8.9a}$$

$$G\frac{\partial \tau}{\partial p} + \frac{\lambda w'(p)}{h^d} \gtreqless 0, \tag{8.9b}$$

where the equality-inequality sign, in (8.9b) and hereinafter, corresponds to the three cases introduced in (8.8). In principle, the pair of equations (8.9) determines, for each $x \in \Omega$, the local optimal values of the pair (h, p), once λ is given.

Thus λ is our global control parameter. Choosing larger λ, we will get an optimized grid with less work and poorer accuracy; lowering λ, we invest more work and get

higher accuracy. For each λ, however, we get (approximately) the highest accuracy for the work invested. In principle λ should be given by whoever submits the problem for numerical solution; i.e., he should tell at what rate of exchange he is willing to invest computational work for additional accuracy (see (8.6)). In practice this is not done, and λ usually serves as a convenient control parameter (see Sections 8.2 and 8.3).

To compute h and p from (8.9) we should know the behavior of τ as a function of h and p. Generally,

$$\tau(x, h, p) \approx t(x, p)h^p, \tag{8.10}$$

where $t(x, p)$ depends on the equations and on the solution. Since it is assumed that all our numerical approximations are in some neighborhood of the solution (see Section 8.2), we may assume that the truncation-error estimates, automatically calculated by the multi-grid processing (see (5.7), for example), give us local estimates for $t(x, p)$. In practice, we never actually solve (8.9), but use these relations to decide upon changes in h and p (see Section 8.3), so that we need to estimate $\tau(x, h, p)$ only for h and p close to the current $h(x)$ and $p(x)$.

In *finite-element formulations* the differential problem is often given as a problem of minimizing a functional $A(U)$. Thus the natural discretization optimization problem is to minimize $A(U^k)$ in a given amount of work W, where the space S^k of approximations U^k is not fixed. This can be translated to practical criteria for adapting S^k. (See end of Section 8.3.)

8.2. *Continuation Methods.* Continuation methods are generally used in numerical solutions of nonlinear boundary value problems. A certain *problem-parameter*, γ say, is introduced, so that instead of a single isolated problem we consider a continuum of problems, one problem $P(\gamma)$ for each value of γ in an interval $\gamma_0 \leqslant \gamma \leqslant \gamma_*$, where $P(\gamma_0)$ is easily solvable (e.g., it is linear), and $P(\gamma_*)$ is the target (the given) problem. The continuation method of solution is to advance γ from γ_0 to γ_* in steps $\delta\gamma$. At each step we use the final solution of the previous step (or extrapolation from several previous steps) as a first approximation in an iterative process for solving $P(\gamma)$. The main purpose of such continuation procedures is to ensure that the approximations we use in the iterative process are always "close enough" to the solution (of the current $P(\gamma)$), so that some desirable properties are maintained. Usually γ is some natural physical parameter (the Reynolds number, the Mach number, etc.) in terms of which either the differential equations or the boundary conditions, or both, are expressed.

The continuation process is not a waste, for several reasons. In many cases, the intermediate problems $P(\gamma)$ are interesting by themselves, since they correspond to a sequence of cases of the same physical problem. More importantly, in solving nonlinear discretized problems the continuation process is not only a method of computing the solution, but also, in effect, the only way to *define* the solution, i.e., the way to select one out of the many solutions of the nonlinear algebraic system. The desired solution is *defined* as the one which is obtained by continuous mapping from $[\gamma_0, \gamma_*]$ to the solution space with a given solution at γ_0 (e.g., the single solution, if $P(\gamma_0)$ is linear). By the continuation process, we keep every intermediate numerical solution in the vicinity of a *physical* solution (to an intermediate problem), hence the target numerical

solution is, hopefully, near the target physical solution, and is not some spurious solution of the algebraic system. Thus, although sometimes we may get away without a continuation process (simply because a starting solution is "close enough", so that the continuation may be done in just one step), in principle a continuation process *must* be present in any numerical solution of nonlinear problems. Moreover, such a process is usually inexpensive, since it can be done with crude accuracy, so that its intermediate steps usually total less computational work than the final step of computing an accurate solution to $P(\gamma_*)$.

A continuation process is necessary, in principle, not only for nonlinear problems but also for linear problems with grid adaptation. In fact, when h or p are themselves unknown, the discrete problem is nonlinear, even if the differential problem is linear.

In our system, a continuation process with crude accuracy and little work is automatically obtained by selecting a large value for the control parameter λ (cf. Section 8.1). Then, in the final step ($\gamma = \gamma_*$), λ is decreased to refine the solution. Thus, the overall process may be viewed as *a multi-grid process of solution, controlled by the two parameters* γ *and* λ.

The most efficient way of changing γ is probably to change it as soon as possible (e.g., when the multi-grid processing exhibits convergence to a crude tolerance), and to control the step-size $\delta\gamma$ by some automatic procedure, so that $\delta\gamma$ is sharply decreased when divergence is sensed (in the multi-grid processing), and slowly increased otherwise.

In changing γ it is advisable to keep the residuals as smooth as possible, since higher-frequency components are more expensive to liquidate (lower components being liquidated on coarser grids). Thus, for example, if a boundary condition should be changed while changing γ, it is advisable to introduce this change into the system at a stage when the algorithm is working at the coarsest level.

γ-Extrapolation. In some cases the given problem ($\gamma = \gamma_*$) is much too difficult to solve, e.g., because the differential solution fluctuates on a scale too fine to be resolved. In such cases one is normally not interested in the details of the solution but rather in a certain functional of the solution. It is sometimes possible in such cases to solve the problem for certain values of γ far from γ_*, and to extrapolate the corresponding functional values to $\gamma = \gamma_*$.

8.3. *Practice of Discretization Control.* The main practical restrictions imposed on the theoretical discretization equations (8.9) are the following: The approximation order p should be a positive integer. In many problems p is also restricted to be even, since odd orders are less efficient. The mesh-size function $h(x)$ should be such that a reasonable grid can be constructed with it. Thus, in the grid structure outlined in Section 7.1, h is restricted to be of the form $h = 2^{-k}h_0$, where k is an integer. Also, in the multi-grid discretization method outlined in Section 7.2, any uniform subgrid truly influences the global solution only if it is large enough, i.e., if at least some of its inner points belong also to coarser grids. These discretization restrictions will actually help us in meeting another practical requirement, namely, the need to keep the control work (computer work invested in testing for and affecting discretization reformulations) small compared with the numerical work (relaxation sweeps and interpolations).

The practical adaptive procedure is proposed to be generally along the following lines:

A. *Testing.* In the multi-grid solution process (possibly incorporating a continuation process), at some natural point we get an estimate of the decrease in the error estimator E introduced by the present discretization parameters. For example, in FAS Cycle C (see its flowchart in Figure 2), at the point where new $\overline{F}^k$ is computed, the quantity

$$-\Delta E = G\,|\overline{F}^k - I_{k+1}^k \overline{F}^{k+1}| \tag{8.11}$$

at each point may serve as a local estimate for the decrease in E per unit volume (cf. (8.1) and (5.7)), owing to the refinement from h_k to h_{k+1}. Each such decrease in E is related to some additional work ΔW (per unit volume). For example, the refinement from h_k to h_{k+1} requires the additional work

$$\Delta W = \frac{w(p)}{h_{k+1}^d} - \frac{w(p)}{h_k^d} \quad \text{(per unit volume)}. \tag{8.12}$$

Hence we compute the ratio of exchanging accuracy per work $Q = -\Delta E/\Delta W$. At regions where this ratio is much bigger than λ (the control rate of exchange; cf. Section 8.1) we say that the present parameter (h_{k+1} in the example) is highly profitable, and it is worth trying to further refine the discretization (e.g., introduce there the subgrid G^{k+2} with $h_{k+2} = h_{k+1}/2$). At regions where Q is much smaller than λ, we may coarsen the discretization (abolish the G^{k+1} subgrid).

Extrapolated Tests. More sophisticated tests may be based on assuming the truncation error to have some form of dependence on h and p, such as (8.10) above. Instead of using ΔE and ΔW at the previous change (from h_k to h_{k+1}, in the above example) we can then anticipate the corresponding values $\overline{\Delta E}$ and $\overline{\Delta W}$ at the next change (from h_{k+1} to h_{k+2}), which are the more appropriate values in testing whether to make that next change. Thus, in the above example, assuming (8.10) and $h_{k+2} = h_{k+1}/2 = h_k/4$, we get $\overline{\Delta E} = 2^{-p}\Delta E$, $\overline{\Delta W} = 2^d \Delta W$, and hence

$$\overline{Q} = \frac{-\overline{\Delta E}}{\overline{\Delta W}} = 2^{-p-d}Q = \frac{h_{k+1}^d G|\overline{F}^k - I_{k+1}^k \overline{F}^{k+1}|}{w(p)(2^d - 1)2^p}. \tag{8.13}$$

The extrapolated ratio $\overline{Q}$ is used in testing for grid changes. This may seem risky, since it depends on assuming (8.10). But in fact there is no such risk, because we can see from (8.13) that testing with $\overline{Q}$ is not that much different from testing with Q. (In fact, if p is constant, testing with $\overline{Q}$ is equivalent to testing with Q against another constant λ.) And the test with Q does not presume (8.10); it only assumes that the finer (G^{k+1}) approximation is considerably better than the coarser one, so that their difference roughly corresponds to an added accuracy due to the refinement. Note also that the multi-grid stopping criteria ((A.17) or (A.20) in Appendix A) are precisely such that Q can be reliably computed from the final approximation.

B. *Changing the Discretization.* The desired grid changes are first just recorded (e.g., incidentally to the stage of computing $\overline{F}^k$) and only then they are simultaneously introduced, taking into account some organizational and stabilizational considerations:

A change (e.g., refinement) is introduced only if there is a point where the change is "overdue" (e.g., a point where $\bar{Q} > 10\lambda$). Together with such a point the change is then also introduced at all neighbor (and neighbor of neighbor, etc.) points where the change is "due" (e.g., where $\bar{Q} > 3\lambda$). The changed subgrid (G^{k+2} in the above example) is then augmented as follows: (i) Around each new grid point we add extra points, if necessary, so that the grid point (corresponding to a G^{k+1} point where a refinement was due) becomes an *inner* point (cf. Section 7.2) in the new subgrid (G^{k+2}). (ii) Holes are filled; that is, if, on any grid line, a couple of points are missing in between grid points, then the missing points are added to the grid.

The control work in this system is negligible compared with, say, the work of relaxing over G^{k+1}, because: (i) The tests are made in the transition from G^{k+1} to G^k, which takes place only once per several G^{k+1} relaxation sweeps. (ii) Q is computed and tested only at points of the coarser grid G^k, and at each such point the worl is smaller than the relaxation work per point. (iii) Changing the discretization is itself inexpensive since it is done by extending or contracting uniform grids (cf. Section 7.1), the main work being in interpolating the approximate solution to the new piece of uniform subgrid.

Practical discretization control in finite-element formulations (see closing remark to Section 8.1) is natural: Let $u^k \in S^k$ be the evolving approximate solution, and u^l its projection on a subspace S^l obtained from S^k by (locally) removing some of the parameters (omitting some grid points or lowering the local polynomial degree). It is easy to locally calculate $\Delta A = A(u^l) - A(u^k)$. It is also easy to estimate $\Delta W = W_k - W_l$, where W_j is the work in relaxing over S^j; in a suitable work unit, $W_k - W_l$ may simply equal the number of parameters removed. The removed parameters are "highly profitable" if $Q = \Delta A/\Delta W$ is much larger than λ, in which case it is worth adding more such parameters (more grid points or higher polynomial terms, correspondingly).

8.4. *Generalizations.* In some problems it is not enough to adapt h and p. Some times different increments $h^{(1)}, h^{(2)}, \ldots, h^{(d)}$ should be used at the d different directions, and each $h^{(j)}$ should be separately adapted. Basically the same procedures as above can be used to test and execute, for example, a change from $h^{(j)}$ to $h^{(j)}/2$. More generally, one would like to adapt the local coordinates (cf. Section 7.4), e.g., near discontinuities. Automatic procedures for such adaptation have not been so far developed, but are conceivable.

Other discretization parameters, such as the centering of each term in the difference operator, may be treated adaptively. (In fact, such adaptive discretization is already in use in mixed-type problems, where it was introduced by Murman to obtain stability. See, e.g., [9].) In problems with unbounded domains, the discrete domain may be determined adaptively (with increasingly coarser levels; cf. Section 7.1), using a procedure that decides to extend the domain if the previous extension was highly profitable in terms of $-\Delta E/\Delta W$. In many problems, some terms in the difference operator can altogether be discarded on most levels G^k. In particular, in singularly perturbed problems, the highest-order terms may be kept only on the finest-narrowest levels. Decision can again be made in terms of $-\Delta E/\Delta W$, in an obvious way.

9. Adaptive Discretization: Case Studies. To get a transparent view of the discretization patterns and the accuracy-work relations typical to the adaptive procedures proposed above, we consider now several test cases which are simple enough to be analyzed in closed forms. That is, we consider problems with known solutions and simple behavior of the local truncation errors, and we calculate the discretization functions $h(x)$ and $p(x)$ that would be selected by the local optimization equations (8.9), and the resulting relation between the error estimator E and the computational work W.

9.1. *Uniform-Scale Problems.* A problem is said to have the uniform scale $\eta(x)$ if the local truncation error (8.2) has the behavior

$$\tau(x, h, p) \approx t(x)\left[\frac{h}{\eta(x)}\right]^p \qquad (p_0 \leqslant p \leqslant p_1). \tag{9.1}$$

Such a behavior occurs, for example, when the solution is a trigonometric or exponential function $\exp(\theta \cdot x)$, where θ is either a constant or a slowly varying function (see example in Section 9.2). We will also assume for simplicity that (see (8.4))

$$w(p) = w_0 p^l. \tag{9.2}$$

Usually $l = 1$, since the number of terms in the difference equations, and hence also the amount of computer operations at each grid point, are proportional to p. $l = 2$ is appropriate if we assume that we have to increase the precision of our arithmetic when we increase p. Rescaling W, we can assume that $w_0 = 1$.

Using (9.1)–(9.2) in Eqs. (8.9), we get

$$G\tau = \lambda d p^{l-1} h^{-d}, \tag{9.3a}$$

$$G\tau \log \frac{h}{\eta} + \lambda l p^{l-1} h^{-d} \gtreqless 0. \tag{9.3b}$$

Hence, denoting by $\tilde{p}$ the value of p that satisfies

$$p^{l-1} e^{lp/d} = \lambda^{-1} G t \eta^d e^{-l} d^{-1}, \tag{9.4}$$

we have

$$h = \eta e^{-l/d}, \qquad p = \tilde{p} \quad \text{if } p_0 \leqslant \tilde{p} \leqslant p_1, \tag{9.5a}$$

$$h = (\lambda d p_0^{l-1} \eta^{p_0} t^{-1} G^{-1})^{1/(p_0+d)}, \qquad p = p_0 \quad \text{if } \tilde{p} \leqslant p_0, \tag{9.5b}$$

$$h = (\lambda d p_1^{l-1} \eta^{p_1} t^{-1} G^{-1})^{1/(p_1+d)}, \qquad p = p_1 \quad \text{if } p_1 \leqslant \tilde{p}. \tag{9.5c}$$

Notice that at any point either p or h, but never both, is "adaptive", i.e., dependent of λ. Where p is adaptive ($p_0 \leqslant p = \tilde{p} \leqslant p_1$), h is fixed and each "scale cube" η^d is divided into e^l mesh cells.

Assume now further that the computer precision is unlimited (which is never really the case, but may provide insight), so that $l = 1$ and $p_1 = \infty$. If sufficiently high accuracy is desired, then λ is sufficiently small to have $\tilde{p} > p_0$, so that (9.5a) applies. By (8.1) and (9.3a) this implies

$$E = \lambda d e \int \eta^{-d}\, dx, \tag{9.6}$$

and hence, by (8.6),

$$E = C_0 e^{-W/(ed\int \eta^{-d}\,dx)} = C_0 e^{-c\beta^d W}, \tag{9.7}$$

where β is some average value of the scale $\eta(x)$. In this (idealized) case, E decreases exponentially with W. For realistic W this convergence rate becomes poor when β is very small, as in singularly perturbed problems. In such problems, however, for realistic W (9.5a) no longer applies, and another rate of convergence, independent of β, takes over (see Section 9.3).

9.2. *One-Dimensional Case.* Consider a 2-point boundary-value problem

$$\frac{\eta}{2}\frac{d^2U}{dx^2} + \frac{dU}{dx} = 0 \quad \text{in } 0 < x < 1, \tag{9.8}$$

with constant $\eta > 0$ and with boundary conditions $U(0)$ and $U(1)$ such that the solution is $U = e^{-2x/\eta}$. An elliptic (stable) difference approximation to such an equation can be central for $\eta \geqslant h$ but should be properly directed for $\eta < h$. (The first-order term being the main term, the second-order term should be differenced backward relative to it with approximation order $p' = p - [\log \eta/\log h]$. See [4] and Section 3.2 in [3].) In either case, the truncation error is approximately

$$\tau(x, h, p) = t(x)\left(\frac{h}{\eta}\right)^p, \quad \text{where } t(x) = \frac{1}{2\eta}e^{-2x/\eta}. \tag{9.9}$$

We now choose the error weighting function to be

$$G(x) \equiv 1, \tag{9.10}$$

which would be the choice (see (8.3)) when one is interested in accurate computation of boundary first-order derivatives (corresponding, e.g., to boundary pressure or drag, in some physical models). We again assume no precision limitations, so that $l = 1$ and $p_1 = \infty$. We take $p_0 = 2$ since second-order is no more expensive than first-order approximation. Inserting these into (9.5), we get

$$h = \frac{\eta}{e}, \quad p = \log\frac{1}{2\lambda} - 1 - \frac{2x}{\eta} \quad \text{for } 0 < x \leqslant x_0, \tag{9.11a}$$

$$h = \frac{\eta}{e}e^{2(x-x_0)/(3\eta)}, \quad p = 2 \quad \text{for } x_0 \leqslant x < 1, \tag{9.11b}$$

where

$$x_0 = \frac{\eta}{2}\left(\log\frac{1}{2\lambda} - 3\right). \tag{9.11c}$$

If $x_0 \geqslant 1$, then (9.11a) applies throughout, and hence

$$W = \int_0^1 \frac{p}{h}\,dx = \frac{e}{\eta}\left(\log\frac{1}{2\lambda} - 1 - \frac{1}{\eta}\right), \tag{9.12}$$

$$E = \int_0^1 \tau\,dx = \frac{\lambda e}{\eta} = \frac{1}{2\eta}e^{-\eta W/e - 1/\eta} \tag{9.13}$$

and the condition $x_0 \geqslant 1$ itself becomes, by (9.11c), (9.12),

$$W \geqslant (2 + 1/\eta)e/\eta. \tag{9.14}$$

Thus, if W satisfies (9.14), E converges like (9.13).

9.3. *Singular Perturbation: Boundary-Layer Resolution.* When η is very small, problem (9.8) is singularly perturbed, and its solution has a boundary layer near $x = 0$. The above mesh-size $h = \eta/e$ is too small to be practical. Indeed, in the optimal discretization (9.11), for small η we get small x_0, and an "external region" $x_0 \leqslant x < 1$ is formed where the mesh-size grows exponentially from η/e. The small mesh-size is used only to resolve the boundary layer. In this simplified problem the solution away from the boundary layer (i.e., for $x \gg \eta$) is practically constant, so that indefinitely large h is suitable. Usually h will grow exponentially, as in (9.11b), from $h = \eta/e$ to some definite value suitable for the external region. In the transition region we have $p = 2$, i.e., the minimal order of differencing is used in the region where h changes. This may be useful in practical implementations.

From (9.11) and (9.9) we get for small η

$$W = \int_0^1 \frac{p}{h}\,dx \approx \frac{e}{4}\left(\log\frac{1}{2\lambda}\right)^2, \tag{9.15}$$

$$E = \int_0^1 \tau\,dx = \frac{\lambda e}{2}\log\frac{1}{2\lambda} \approx \left(\frac{e}{4}W\right)^{1/2} e^{-(4W/e)^{1/2}}, \tag{9.16}$$

where the integrals are separately calculated in $(0, x_0)$ and $(x_0, 1)$. Thus, E converges exponentially as a function of $W^{1/2}$ instead of W, but this rate is independent of η and does not deteriorate as $\eta \longrightarrow 0$.

9.4. *Singular Perturbation Without Boundary-Layer Resolution.* To see the effect of choosing different error weighting functions, consider again the above problem (Sections 9.2, 9.3), but with the choice $G(x) = x$. This choice is typical to cases where one is not interested in calculating boundary derivatives of the solution (see (8.3)). We then get

$$\tilde{p} = \log\frac{x}{2\lambda} - 1 - \frac{2x}{\eta} \leqslant \log\frac{\eta}{4\lambda} - 2. \tag{9.17}$$

Therefore, for small η and reasonable λ, $\tilde{p} < 0$ and $p = 2$ for all x. Hence, no resolution of the boundary layer is formed. Indeed, by (9.5b), for very small η (singular-perturbation case)

$$\left(\frac{h}{\eta}\right)^3 = \frac{2\lambda}{x}e^{2x/\eta} \geqslant \frac{4\lambda e}{\eta} \gg 1 \tag{9.18}$$

so that $h \gg \eta$. In the practical situation where the solution in the external region is not constant, the actual mesh-size will be determined by the external regime.

9.5. *Boundary Corners.* Consider the two-dimensional Poisson equation $\Delta U = F$ with smooth F and homogeneous boundary conditions, near a boundary corner with angle π/α, $\frac{1}{2} \leqslant \alpha < 1$. Denoting by r the distance from the corner, at small r the solution U is $O(r^\alpha)$, and so is also the error weighting function G (if accuracy is sought in the solution, but not in its derivatives near the boundary). Hence, $\tau = O(h^p r^{\alpha-p-2})$ and $\partial\tau/\partial h = O(h^{p-1} r^{\alpha-p-2})$. If we fix the order of approximation p, then the optimal mesh-spacing derived from (8.9a) is

$$h = O(\lambda^{1/(p+2)} r^\beta), \qquad \beta = \frac{p + 2 - 2\alpha}{p + 2}. \tag{9.19}$$

Hence, by (8.4) and (8.1) the total work and total error contribution from a region of radius r around the corner are, respectively,

$$W = \int \frac{p}{h^2}\, dx\, dy = O(\lambda^{-2/(p+2)} r^{2-2\beta}),$$

$$E = \int G\tau\, dx\, dy = O(\lambda^{p/(p+2)} r^{2-2\beta}).$$

Hence the relation $E \approx W^{-p/d}$ (the usual relation in d-dimensional smooth problem with pth-order approximation) still holds uniformly. The corner does not "contaminate the global convergence.

In the practical grid organization (Section 8.3) finer levels G^k with increasingly smaller mesh-sizes $h_k = 2^{-k} h_0$ will be introduced near the corner. By (9.19), the level G^k will extend from the corner to a distance $r_k = C\lambda^{2\alpha-p-2} h_k^{1/\beta}$. Since $\beta < 1$, for small h_k we get $h_k > r_k$. This gives us in practice a natural stopping value for the refinement process: The finest mesh-size near the corner is such that $h_k \approx 4r_k$, so that level G^k still has an inner point belonging to G^{k-1}.

9.6. *Singularities.* Like boundary corners, all kinds of other problem singularities when treated adaptively, cause no degradation of the convergence rate (of E as a function of W).

Consider for example the differential equation $LU = F$ where F is smooth except for a jump discontinuity at $x = 0$. Whatever the approximation order p, the system will find $-\Delta E$ (see (8.11)) to be $O(1)$ at all points whose difference equation include values on both sides of the discontinuity. At these points further refinements will, therefore, be introduced as long as $-\Delta E/\Delta W > O(\lambda)$. Thus, around $x = 0$, some fixed number (depending only on p) of mesh-points will be introduced at each level G^k, until a mesh-size $\bar{h} = O(\lambda^{1/d})$ is reached. The total amount of added work is therefore proportional to the number of levels introduced, which is $O(\log \bar{h})$. The error contribution of the discontinuity is $O(\bar{h}^d)$, which is exponentially small in terms of the added work.

This and similar analyses show that the adaptive scheme retains its high-order convergence even when the problem is only piecewise smooth, or has algebraic singularities, etc.

10. Historical Notes and Acknowledgements. Coarse-grid acceleration techniques were recommended and used by several authors, including Southwell [25], [13], [14], Stiefel [15], Fedorenko [5], Ahamed [19], Wachspress [17], de la Vallée Poussin [16] and Settari and Aziz [24]. Southwell called his technique "block" and more generally "group relaxation", described it as "almost essential to practical success", and gave heuristic explanation as well as practical implementation methods based on variational considerations ("the aim being to reduce the total energy by as great an amount as possible"). He also depicted procedures of "advance to a finer net" [14]. Techniques of multiplicative coarse-grid corrections (special cases of which appeared in [14], [19]) were developed by Wachspress [17, Chapter 9], who called them "variational techniques". This work motivated several studies, by Froelich, Wagner, Nakamura and Reed (see a brief survey in [18]) and was applied in nuclear reactor design computations.

All these were two-level methods. The multi-grid idea was introduced by Fedorenko [6], mainly for theoretical purposes. Namely, he rigorously proved that $W(n, \epsilon)$, the number of operations required to reduce the residuals, of a Poisson problem on a rectangular grid with n points, by a factor ϵ, is $O(n|\log \epsilon|)$. Bakhvalov [1] generalized this result to any second-order elliptic operator with continuous coefficients. For large n, this is the best possible result—except for the actual value of the coefficient. The Fedorenko estimate can be written as

$$W(n, .01) \leqslant 210000n + W(10^6, .01),$$

and the Bakhvalov constants are still much larger. For admissible values of n these estimates are therefore far worse than estimates obtained in other methods, and they did not encourage any development of the method. Fedorenko experimented with a two-level algorithm only, and seemed to imply that for practical grid sizes ADI may be more efficient. He did not realize the true practical potential, in both efficiency and programming simplification, of a full, systematic multi-grid approach. (It can be proved that $W(n, .01) \leqslant 168n$, and in practice $W(n, .01) \approx 50n$ is obtainable. See Appendix C.)

The first full multi-grid algorithms and numerical tests were described in [2]. Our original approach was to regard the finer levels as "correcting" the coarser level (cf. Sections 1, 7.2 and 7.5 above). For uniform nonadaptive grids this approach turns out to be equivalent to the one implied by [6], but fundamentally it is different and more powerful, since the process is not confined to preassigned discrete systems.

A systematic multi-grid approach for a restricted class of problems, with somewhat different procedures of relaxation and transfer to coarser grids, is described in [21]. Another similar method was independently developed in [28]. The multi-grid method is also portrayed in [23] and a new rigorous $O(n)$ convergence theorem for finite-element formulations is given in [29].

Adaptive discretization procedures were introduced by several authors. See for example [10], [20], [22] and references in [22]. The present approach is different, not only in its multi-level setting, but also in its basic criteria and procedures. It has common features with procedures for initial-value problems in ordinary differential equations, described in Chapter 5 of [27].

It is my pleasure to acknowledge the help I received from my students and colleagues throughout the work reported here. Yosef Shiftan, Nathan Diner, Yehoshua Fuchs and Dan Ophir in the Weizmann Institute; Jerry South in NASA Langley Research Center; and Will Miranker, Don Quarles, Fred Gustavson and Allan Goodman at IBM Thomas J. Watson Research Center—thank you all. I am also grateful for valuable discussions I had with Olof Widlund, Eugene Wachspress, Antony Jameson, Perry Newman, Jim Ortega, Ivo Babuška and Werner Rheinboldt.

Appendix A. Interpolations and Stopping Criteria: Analysis and Rules. The multi-grid algorithms described above (Sections 4 and 5) need to be supplemented with some rules of interpolation and some stopping criteria. More specifically, for the interpolation I_k^{k-1}, transferring weighted residuals from a fine grid G^k to the next

coarser grid G^{k-1}, we should prescribe the weights, while for I_{k-1}^{k}, interpolating corrections from G^{k-1} back to G^{k}, the method and order of interpolation should be prescribed. Stopping criteria should define convergence at the various levels and detect slow convergence rates. Numerical tests show that the parameters to be used are very robust: Full efficiency of the multi-grid algorithm is obtained for stopping parameters that do not depend on the geometry and the mesh-size, and which may change over a wide range (see, e.g., Appendix B), provided the correct *forms* of the stopping criteria are used, and some basic rules of interpolation are observed. To find the correct forms and rules, and to determine the stopping parameters, we have to analyze the Coarse-Grid Correction (CGC) cycle, which consists of interpolating (I_{k}^{k-1}) the residuals to the coarser grid G^{k-1}, solving the corresponding residual problem on G^{k-1}, and then interpolating (I_{k-1}^{k}) that solution back as a correction to the G^{k} approximation.

We can use a local mode analysis (for the linearized, coefficient-frozen difference equations), similar to the example in Section 3. Such an analysis may be inaccurate for the lowest frequency modes, for which the interaction with the boundary is significant. But these lowest modes are of little significance in our considerations, since they are efficiently approximated on the coarsest grids with little computational work, and since care will be taken (i) to choose interpolation schemes that do not convert small low-frequency errors into large high-frequency errors; and (ii) to stop relaxation sweeps before low-frequency error components become so large that they significantly feed the high frequencies (e.g., by boundary and nonlinear interactions). In fact, we will see that the *dominant components* (i.e., the components that are slowest to converge in the combined process of relaxation and coarse-grid corrections) are the Fourier components $e^{i\theta\cdot x/h}$ for which $|\theta|$ is close to $\hat{\rho}\pi$, where (in a general d-dimensional problem)

$$\text{(A.0)} \qquad \theta = (\theta_1, \theta_2, \ldots, \theta_d), \qquad \theta \cdot x = \sum_{j=1}^{d} \theta_j x_j, \qquad |\theta| = \max_{1 \leqslant j \leqslant d} |\theta_j|,$$
$$h = h_k = \hat{\rho} h_{k-1}.$$

These components feed on each other in the interpolation processes between G^{k} and G^{k-1}, they are slower to converge by relaxation, and in the CGC cycles they may even diverge.

To simplify the discussion we will assume that the mesh-size ratio has its usual value $\hat{\rho} = \frac{1}{2}$, which is the only one to be used in practice (cf. Section 6.2).

A.1. *Coarse-Grid Amplification Factors.* For any given set of difference operators L^{k} and a multi-grid scheme, a local mode analysis of the complete MG cycle can be made (cf. Appendix C), and the various parameters can be optimized. The essential information can, however, be obtained from a much simpler analysis that treat separately the two main processes, relaxation sweeps and CGC cycles. The smoothing rate $\bar{\mu}$ (see Section 3) is the main quantity describing the relaxation sweeps. The CGC local mode analysis is summarized below (for algebraic details see Section 4.5 of [3]).

In the CGC analysis, together with each basic Fourier component $e^{i\theta\cdot x/h}$ $(0 < |\theta| \leqslant \pi/2)$ we should treat all the G^{k} components that coincide with it on G^{k-1}, i.e., all components $e^{i\theta'\cdot x/h}$ $(0 < |\theta'| \leqslant \pi)$ such that $\theta_j' \equiv \theta_j \pmod{\pi}$ for $j = 1, 2, \ldots, d$. We call such components θ' *harmonics* of θ. We are especially interested in those

harmonics that are not separated from θ by the relaxation sweeps, e.g., the set

$$\Upsilon_\theta = \{\theta' \equiv \theta \pmod{\pi}: \mu(\theta') \geqslant \mu(\theta)^2\}.$$

Denote by $|\Upsilon_\theta|$ the number of members in this set. (Usually $|\Upsilon_\theta| = 2^\alpha$, where α is the number of coordinates j for which $|\theta_j| \approx \pi/2$.) In terms of the θ Fourier component and its harmonics, the CGC cycle has two effects:

(i) Assuming the components not in Υ_θ to be comparatively small when the CGC cycle is entered, the set of components in Υ_θ is transformed in the cycle by a certain matrix, whose spectral radius turns out to be

$$\sigma(\theta) = \begin{cases} \sigma_0(\theta) & \text{if } |\Upsilon_\theta| = 1, \\ \max(1, \sigma_0(\theta)) & \text{if } |\Upsilon_\theta| > 1, \end{cases} \tag{A.1}$$

where

$$\sigma_0(\theta) = \left| 1 - \sum_{\theta' \in \Upsilon_\theta} B_k(\theta')\, R(\theta, \theta')\, B_{k-1}(2\theta)^{-1} \rho(\theta') \right|. \tag{A.2}$$

The functions $\rho(\theta')$, $R(\theta, \theta')$ and $B_l(\theta)$ are the "symbols" of I_k^{k-1}, I_{k-1}^k and L^l, respectively, i.e.,

$$\begin{aligned} I_k^{k-1} e^{i\theta'\cdot x/h} &= \rho(\theta') e^{i\theta\cdot x} \qquad \text{(cf. (A.10) below)}, \\ I_{k-1}^k e^{i\theta\cdot x/h} &= \sum_{\theta' \equiv \theta \pmod{\pi}} R(\theta, \theta') e^{i\theta'\cdot x/h}, \\ L^l e^{i\theta\cdot x/h_l} &= B_l(\theta) e^{i\theta\cdot x/h_l} \qquad (l = k,\ k-1). \end{aligned} \tag{A.3}$$

(If L is a *system* of equations, and the right-hand side of (A.2) is therefore a matrix, then $\sigma_0(\theta)$ is meant to be the spectral radius of that matrix.) For small $|\theta|$ we have $|\Upsilon_\theta| = 1$ and hence

$$\sigma(\theta) = \sigma_0(\theta) = 1 - \rho(0) + O(|\theta|^p + |\theta|^I), \tag{A.4}$$

where p is the approximation order of L_k and L_{k-1} (or the minimum of the two) and I is the order of the I_{k-1}^k interpolation ($I = 2$ for linear interpolation, etc.). The principal CGC amplification factor is

$$\bar{\sigma} = \max_{0 \leqslant |\theta| \leqslant \pi/2} \sigma(\theta) = \max(1, \bar{\sigma}_0), \quad \text{where} \quad \bar{\sigma}_0 = \max_{0 \leqslant |\theta| \leqslant \pi/2} \sigma_0(\theta). \tag{A.5}$$

(ii) The CGC cycles also generate new secondary harmonics $\theta'' \notin \Upsilon_\theta$. The rate of generating these, i.e., the ratio of the new θ'' amplitude to the old amplitude of the combined harmonics, turns out to be

$$\sigma_1(\theta'') = |\, B_k(\theta'')\, R(\theta, \theta'')\, B_{k-1}(2\theta)^{-1} \rho(\theta'')| = O(|\theta|^{I-m}), \tag{A.6}$$

where m is the order of the differential equations.

It follows from (A.4), (A.6), that if $\rho(0) = 1$, as it is always chosen to be (cf. Section A.4), and if $I \geqslant m$, then components with small $|\theta|$ are very efficiently reduced in the multi-grid process, together with their harmonics.

A.2. *The Coarse-to-Fine Interpolation* I_{k-1}^{k}. On the other hand, it follows from (A.6) that if $I < m$ then even a small and smooth residual function may produce large high-frequency residuals, and significant amount of computational work will be required to smooth them out. This effect was clearly shown in numerical experiments [2], [11] Hence we have

The Basic Rule: *The order of interpolation should be no less than the order of the differential equations.* ($I \geqslant m$.) In particular, polynomial interpolation should be made with polynomials of degree $\geqslant m - 1$.

Higher interpolation orders ($I > m$) are desired in the initial stages of solving a problem, when the residuals are (locally) smooth. For instance, in regions where the given problem has smoothness of order q (i.e., $F(x) = \Sigma A_\theta e^{i\theta \cdot x/h}$, $A_\theta = O(|\theta|^{-q} h^q)$), in order to ensure that the high-frequency residuals remain $O(h^q)$, *at the ith interpolation from* u^{M-1} *to* u^M *the order should be*

$$I \geqslant m + \max[q - (i-1)p, 0]. \tag{A.7}$$

(In fact, as long as $q > ip$, this interpolation need not be followed by G^k relaxation sweeps, since the low-frequency amplitudes are still dominant. Relaxation would only feed from these low components to high-frequency ones, causing additional work later. Still better, however, instead of this multi-grid mode without intermediate G^k relaxation, is to make a higher-order correction on G^{k-1}.)

Eventually, however, the smoothness of F (which is the original residual function) is completely lost in subsequent residuals and the convergence of components in the dominant range ($|\theta| \approx \pi/2$) becomes our main concern. For these components, *higher interpolation orders* ($I > m$) *is no more effective than the minimal order* ($I = m$). This again was exhibited in numerical experiments [2], [11], which confirmed that the multi-grid efficiency is not improved (except in the $\lceil q/p \rceil$ first cycles) by using $I > m$.

An efficient method to implement high-order interpolations in case of equations of the form $\Delta^m U = F$ is to base the interpolation on suitably-rotated difference approximations. See [14, p. 53] and [7].

A.3. *The Effective Smoothing Rate.* The smoothing factor $\bar{\mu}$ was defined in (3.8) as the largest convergence factor for all components not represented at the coarser level. More relevant, however, is the largest factor among all components for which the coarse-grid correction is not effective, namely,

$$\tilde{\mu} = \max\{\mu(\theta):\ \pi/2 \leqslant |\theta| \leqslant \pi \quad \text{or} \quad \sigma_0(\theta) \geqslant 1\}, \tag{A.8}$$

which we call the "effective smoothing factor". It is clear, on one hand, that no factor smaller than $\tilde{\mu}$ can be generally obtained as a convergence factor per G^k relaxation sweep, no matter how well and how often the G^{k-1} problem is solved. On the other hand, the factor $\tilde{\mu}$ can actually be attained (or approached) by correctly balancing the number of relaxation sweeps in between CGC cycles (see Section A.6). In most cases (all cases examined by us) one can make $\sigma_0(\theta) \leqslant 1$ for all $|\theta| \leqslant \pi/2$ by proper choice of I_k^{k-1} (see Section A.4), and it is therefore justifiable to use $\bar{\mu}$ as the effective factor when relaxation schemes are studied by themselves.

A.4. *The Fine-to-Coarse Weighting of Residuals* (I_k^{k-1}), *and the Coarse-Grid Operator* L^{k-1}. The transfer of the G^k residuals $r^k = f^k - L^k u^k$ to the coarser grid G^{k-1}, to serve there as the right-hand side f^{k-1} (see Section 4, Step (e)) can be made in many ways. Generally, f^{k-1} is defined as some weighted average of the residuals in neighboring G^k points:

$$f^{k-1}(x) = I_k^{k-1} r^k(x) = \sum \rho_\nu r^k(x + \nu h), \tag{A.9}$$

where $\nu = (\nu_1, \nu_2, \ldots, \nu_d)$, ν_j integers, and the summation is over a small set. In terms of these weights, $\rho(\theta)$ in (A.2) is given by

$$\rho(\theta) = \sum \rho_\nu e^{i\theta \cdot \nu}. \tag{A.10}$$

The coarse-grid operator L^{k-1} can also be chosen in many ways, e.g., as some weighted average of the operator L^k in neighboring points.

How are these choices to be made? The main purpose should be to minimize $\bar{\sigma}$, but without investing too much computational work in the weighting. Usually, it is preferable to adjust ρ_ν and not L^{k-1}, because this provides enough control on $\bar{\sigma}$ (cf. (A.2)) and because complicating L^{k-1} adds many more computations and gets increasingly complicated as one advances to still coarser levels. For the programmer, using the same operators at all levels is an important simplification (cf. Appendix B), especially for nonlinear problems. Moreover, if L^{k-1} is derived from L in the same way as L^k, its coefficients will automatically be the suitable weighted averages of the coefficient of L^k. (In case L^k is given and L is not, and if the coefficients vary rapidly, then the weighting (A.12) below should be used for defining the L^{k-1} coefficients.)

It is clear from (A.4) that we should take $\rho(0) = \Sigma \rho_\nu = 1$. There is no a priori restriction, however, on the signs of the weights ρ_ν. The trivial weighting

$$\rho_0 = 1, \quad \rho_\nu = 0 \quad \text{for } \nu \neq 0; \quad \rho(\theta) \equiv 1, \tag{A.11}$$

called *injection*, has an important advantage in saving computations, not only because the weighting itself is saved, but mainly because it requires the computation of r^k only at the G^{k-1} points, while other weighting schemes compute r^k at all G^k points, an additional work comparable to one G^k relaxation sweep.

Nontrivial weighting of residuals is *especially important for difference equations with rapidly varying coefficients* (large variations per mesh-width), and, in particular, for nonlinear equations, especially on coarse grids. In the G^k relaxation sweeps for such equations, the slowly converging low frequencies are coupled with high-frequency modes. In the correction function V^k (cf. Sections 2 or 4) these accompanying high-frequency modes are small (i.e., their amplitudes are small relative to the low-frequency amplitudes), and hence V^k can still be well approximated by a function V^{k-1} on the coarser grid G^{k-1}. In the residual r^k, however, these accompanying high-frequency modes may be large (comparable in amplitude to the low-frequency modes) and their distortion of V^{k-1} will be small only if their contribution to $f^{k-1} = I_k^{k-1} r^k$ is also of high frequency (in G^{k-1}). This is indeed the case, except for residual components $\theta = (\theta_1, \ldots, \theta_d)$ where $|\theta| \approx \pi$ and each $|\theta_j|$ is close either to π or to 0, ($j = 1, \ldots, d$). Such a component has on G^{k-1} the low frequency 2θ (mod 2π), and is therefore

disproportionately magnified in V^{k-1}. The weighting of residuals should eliminate exactly these components. A suitable weighting is

$$\text{(A.12)} \qquad \rho_\nu = 2^{-d-|\nu|} \quad \text{for } |\nu| = \max|\nu_i| \leqslant 1, \qquad \rho_\nu = 0 \quad \text{for } |\nu| > 1,$$

for which $\rho(\theta) = \Pi_1^d(1 + \cos\theta_j)/2$, so that components with $\theta_j \approx \pi$ are sharply reduce

Similar weighting is required whenever high frequencies are coupled to low ones. Such coupling may be introduced by the relaxation scheme itself, even in constant-coefficient problems. For example, a Gauss-Seidel relaxation sweep with red-black ordering, starting from a smooth $O(1)$ error function, will generate high-frequency modes with $O(h^m)$ amplitudes in the error and $O(1)$ amplitutes in the residual.

Examples. For symmetric second-order equations, injection should usually be used. For the *5-point Laplace operator*, for example, if we take I_k^{k-1} to be injection, I_{k-1}^k linear interpolation, and L^{k-1} also a 5-point Laplace operator, we get $\bar{\sigma} = \bar{\sigma}_0 = 1$, the minimal possible value. Any weighting is a pure waste, including the "optimal" weighting

$$\text{(A.12a)} \quad \rho_{00} = \tfrac{1}{2}, \qquad \rho_{01} = \rho_{0-1} = \rho_{10} = \rho_{-10} = \frac{1}{8}, \qquad \rho_{\alpha\beta} = 0 \quad \text{for } |\alpha| + |\beta| > 1,$$

which minimized $\bar{\sigma}_0$, giving $\bar{\sigma}_0 = 1/3$, but does not lower $\bar{\sigma}$. Numerical tests (modifying the program of Appendix B) indeed showed no improvement by weighting.

For higher-order equations, nontrivial weighting offers an important advantage. If, for example, L^k and L^{k-1} are 13-point *biharmonic* operators and I_{k-1}^k is cubic interpolation, then $\bar{\sigma} = 3$ for injection, while $\bar{\sigma} = 1$ for the weighting

$$\text{(A.12b)} \quad \rho_{01} = \rho_{0-1} = \rho_{10} = \rho_{-10} = \frac{1}{4}, \qquad \rho_{\alpha\beta} = 0 \quad \text{for } |\alpha| + |\beta| \neq 1.$$

Numerical experiments were conducted with difference equations of the form (3.2), where the coefficients a and c vary wildly, e.g., a and c on G^M are random; or $a = .1$ and $c = 1$ at all points of G^M, except for points coinciding with G^{M-1}, where $a = c = 1$; etc. When, and only when, the algorithm (with proper line relaxation) used the proper residual weighting ((A.12), with similar weighting for transferring a and c to coarser grids, since in these tests a and c were not defined by a differential equation), it was found that even in the worst cases, convergence factors $\mathring{\mu}$ smaller than 0.7 were always obtained. With the weighting (A.11) or (A.12a) cases of divergence were detected.

A.5. *Finite-Element Procedures.* The main difference between finite-element and finite-difference multi-grid procedures is in the interpolation schemes. In the finite element case, interpolation procedures follow automatically from the variational formulation and the definition of the approximation spaces S^k (corresponding to the levels G^k). Usually, S^k is a subspace of S^{k-1}. The coarse-to-fine interpolation is, therefore, simply the identity operation. Also, if the variational problem in S^k is to minimize $A_k(V^k)$, then, for any given approximation v^k, the correction problem in the coarser space S^{k-1} is, simply, to minimize

$$\text{(A.13)} \qquad A_{k-1}(V^{k-1}) = A_k(v^k + V^{k-1}) \qquad (V^{k-1} \in S^{k-1}).$$

Example. Consider the standard example, where S^k is the space of piecewise linear functions on the triangulation G^k and A_k is a Dirichlet integral whose minimiza-

tion is equivalent to the difference equation $\Delta^k V^k = F^k$, Δ^k being the 5-point Laplacian. Computing A_{k-1} by (A.13), it turns out to be equivalent to the equation $\Delta^{k-1} V^{k-1} = I_k^{k-1}(F^k - \Delta^k v^K)$, where I_k^{k-1} has the weights (cf. Section A.4)

$$\rho_{00} = \frac{1}{4}, \qquad \rho_{01} = \rho_{11} = \rho_{10} = \rho_{0-1} = \rho_{-1-1} = \rho_{-10} = \frac{1}{8}.$$

These weights give the same multi-grid convergence rate as injection (and are therefore redundant).

A.6. *Criteria for Slow Convergence Rates.* (A) Relaxation sweeps, say on G^k, should be discontinued, and a switch should be made to a coarse-grid correction, when the rate of convergence becomes slow; e.g., when

$$\frac{\text{residual norm}}{\text{residual norm a sweep earlier}} \geqslant \eta \equiv \frac{\bar{\sigma}^3 \hat{\mu} + 3\tilde{\mu}}{\bar{\sigma}^3 + 3}. \tag{A.14}$$

The norm here is a suitable (e.g., L_2, L_∞ or (A.18) below) discrete measure, usually of the "dynamic" residuals, that is, residuals computed incidentally to the relaxation process. $\hat{\mu}$ is the largest amplification factor for Fourier components on the relaxed region. $\hat{\mu} = 1$ can usually be used, except when relaxing on small grids (in partial relaxation sweeps, for example). $\tilde{\mu}$ and $\bar{\sigma}$ are defined in (A.8) and (A.5), respectively. Usually, one can choose the I_k^{k-1} weighting so that $\bar{\sigma} = 1$, in which case $\tilde{\mu} = \bar{\mu}$. In any case, (A.14) is designed to ensure that, on one hand, the CGC cycle is delayed enough to make its $\bar{\sigma}$ magnification small compared with the intermediate reduction by relaxation sweeps. On the other hand, for θ with $\mu(\theta)$ considerably slower than $\tilde{\mu}$, the CGC cycles are still sufficiently frequent to compensate for the slower μ, since their reduction rate $\sigma(\theta)$ decreases rapidly ((A.4) with $\rho(0) = 1$). The stopping rule (A.14) also prevents low error frequencies from dominating the relaxation, thus avoiding significant feeding from low to high frequencies (through boundary and nonlinear interactions).

If the "stopping factor" η varies over the domain of computations (as a result of variations in L, in case of nonlinear or nonconstant-coefficients problems), the largest η should be chosen for the stopping criterion (A.14). If $\log \eta$ changes too much over the domain (which should not happen when a proper relaxation scheme is used), then (A.14) must be checked separately in subdomains, and partial sweeping (see Section A.9) might be used.

An appropriate value of η may also easily be found by direct trial and error. Such a value is typical to the (locally linearized, coefficient-frozen) problem, is independent of either h, Ω or F, and may therefore be found, once and for all, on a moderately coarse grid. In some nonlinear problems the value may need some adjustment as the computations proceed. Whenever the coarse-grid corrections seem to be ineffective, η should be increased, e.g., to $(1 + 3\eta)/4$. Generally, the overall multi-grid convergence rate is not very sensitive to increasing η: At worst, the rate may become η instead of the theoretically best rate $\max_\Omega \tilde{\mu}^{3/4}$ (cf. Section 6.2).

For the *Poisson equation* with Gauss-Seidel relaxation, for example, we have $\bar{\sigma} = 1$, $\tilde{\mu} = \bar{\mu} = .5$, hence $\eta = .625$. The example in Appendix B shows that the optimal MG convergence rate $\tilde{\mu}^{3/4} \approx .595$ is indeed attained. Experimenting with this program gave similar results for any smaller η (the reason being that the minimal num-

ber of two sweeps at each level is good enough in this problem), while for any $\eta \leqslant .95$ the total amount of computational work was no more than twice the work at $\eta = .62$.

(B) Another way to decide upon discontinuation of relaxation is to directly *measure the smoothness of the residuals.* The switch to coarser grids can be made, for instance, when differences between residuals at neighboring points are small compared with the residuals themselves.

A.7. *Convergence Criteria on Coarser Grids.* In the CGC mode analysis above it was assumed that the problem on the coarser grid G^{k-1} was fully solved and then interpolated as a correction to the G^k approximation. In the actual multi-grid algorithm (Section 4) we solve the G^{k-1} problem iteratively, stopping the iterations when some convergence criterion is met. This criterion should roughly detect the situation at which more improvement (per unit work) is obtained by relaxing on the G^k grid (after interpolating) then by further iterating the G^{k-1} problem (before interpolating). A crude mode analysis (similar to Section 4.6.2 in [3]) shows that such a criterion is

$$\|r^{k-1}\| \leqslant \delta \|r^k\|, \qquad \delta = \frac{\bar{\sigma}(1 - \mu_k^{2^{-d}})}{\bar{S}(\mu_k^{2^{-d}} - \mu_{k-1})}, \tag{A.15}$$

where d is the dimension, $\bar{\sigma}$ is given by (A.5),

$$\bar{S} = \max_{|\theta| \leqslant \pi/2} \left| \sum_{\theta \in \Upsilon_\theta} B_k(\theta') R(\theta, \theta') B_{k-1}(2\theta)^{-1} \rho(\theta') \right|$$

and $\mu_l = \tilde{\mu}^{(1-2^{-d})}$ on the G^l grid (cf. (A.8)). $\|r^{k-1}\|$ is any norm, such as L_2, L_∞ or (A.18), of the current residuals in the G^{k-1} problem, while $\|r^k\|$ is the corresponding norm in the G^k problem. It is important that these norms are comparable: They shoul be discrete approximations to the same continuum norm. Also, if r^{k-1} are the "dynamic" residuals (i.e., computed incidentally to the last G^{k-1} relaxation sweep, using at each point the latest available values of the relaxed solution), then r^k should be the G^k dynamic residuals, *unlike* the residuals transferred to G^{k-1} (to define f^{k-1}; cf. Section A.4) which must be "static" residuals (i.e., computed over the grid without changing the solution at the same time). If, however, r^k and r^{k-1} are static and dynamic, respectively, the parameter δ in (A.15) should be multiplied by a certain factor $\bar{\beta}$ (see Section 4.6.2 in [3]).

The stopping criterion (A.15) is based on the assumption that error components with $|\theta| = \pi/2$ dominate the process. In the first $\lceil q/p \rceil$ CGC cycles, however, lower components are dominant, and the main consideration is to converge them. Hence, at that initial stage, the G^{k-1} convergence criteria should be

$$\|r^{k-1}\| \leqslant \|\tau^{k-1}\|, \tag{A.16}$$

where τ^{k-1} are the G^{k-1} truncation errors (cf. Section A.8).

The key factor δ can also be found by trial and error. Like η above, it is essentially independent of h, Ω and F, and may, therefore, be found once and for all by tests on moderately coarse grids. Numerical experiments show that the overall multi-grid efficiency is not very sensitive to very large variations in δ and, in particular, δ

may be lowered by orders of magnitudes without large changes in the efficiency. For example:

For the 5-point *Poisson equation* with Gauss-Seidel relaxation, injection and linear interpolations, (A.15) yields $\delta = .219$. Numerical experiment (e.g., with the program in Appendix B) shows that with any $.001 \leqslant \delta \leqslant .5$ the computational work is no more than 25% above the work with $\delta = .22$, and no more than 100% extra work for any $.0001 \leqslant \delta \leqslant .7$.

A.8. *Convergence on the Finest Grid.* On the finest grid G^M the solution is usually considered converged when the (static) residuals are of the order of the truncation error, in some appropriate norm. One way to estimate the truncation error is to measure them on coarser grids by (5.7), and extrapolate (taking into account that they are $O(h^p)$). Another, related but more straightforward criterion is to detect when the G^M solution has contributed most of its correction to the G^{M-1} solution. In the FAS algorithm the natural place to check is when a new $\overline{F}^{M-1}$ is computed, the convergence test being

$$\|\overline{F}^{M-1} - \overline{F}^{M-1}_{\text{previous}}\| \ll \|\overline{F}^{M-1} - I_M^{M-1} F^M\|. \tag{A.17}$$

The norm here may be any (L_2, L_∞, etc.), but the most relevant one is the discrete version of the norm (cf. Section 8.1)

$$\|f\| = \int G(x)|f(x)|\, dx. \tag{A.18}$$

A.9. *Partial Relaxation Sweeps.* A partial relaxation sweep over G^k is a relaxation sweep that may skip some subdomains of G^k. (Unlike "selective" relaxation sweeps, which in principle pass through all the grid points, although corrections may not be introduced in some of them. Cf. Section 3.2. A partial sweep may be selective, too.)

Partial sweeps are not used much in standard relaxation calculations. Usually, a slow-to-converge subdomain is coupled to other subdomains and therefore cannot be relaxed separately. In the multi-grid process, however, only high-frequency error components are to be reduced by relaxation, and this can be done separately in subdomains: *With regard to high frequencies, subdomains are practically decoupled.* Hence, in the multi-grid process, partial sweeps are potentially very important. In fact, high-frequency amplitudes may vary greatly over the domain, especially if $\bar{\mu}$ and $\bar{\sigma}$ vary much, or if high-frequency error components are introduced at boundaries, making partial sweeping there very desirable.

Partial sweeping may be performed by applying a criterion for slow convergence (Section A.6) separately in subdomains. A subdomain may be excluded from subsequent relaxation sweeps if slow convergence is shown simultaneously on that subdomain and on all neighboring subdomains. Under-relaxation may be used to phase-out the relaxed region (cf. [3], Section 4.6.4). The subdomains may be chosen quite arbitrarily, but each of them should be large enough (at least 4×4) to allow for separate smoothing.

A.10. *Convergence Criteria on Nonuniform Grids.* When G^k and G^{k-1} are not coextensive (i.e., the domain covered by G^k is only part of the G^{k-1} domain; cf. Section 7.2), the convergence criteria (Sections A.7–A.8) should be slightly modified. First, in (A.15), $\|r^k\|$ is not a comparable norm, since it may be measured on a much smaller subdomain. Instead, one can use the test

$$\|r^{k-1}\| < \delta \|r_1^{k-1}\|/\eta, \tag{A.19}$$

where $\|r_1^{k-1}\|$ is the residual norm computed on G^{k-1} at the first relaxation sweep after switching from G^k. The division by η in (A.19) is designed to compensate for the fact that $\|r_1^{k-1}\|$ is computed a sweep later than $\|r^k\|$.

The other modification is in (A.17), where it was assumed that G^M is the finest level everywhere. Generally, the convergence test can be, for example,

$$\|\overline{F}^k - \overline{F}^k_{\text{previous}}\| \ll \|\overline{F}^k - I^k_{k+1}\overline{F}^{k+1}\| \quad \text{for all } k = (0, 1, \ldots, M-1), \tag{A.20}$$

where the norms are taken over G^k_{k+1} (or, more precisely, over $G^k_{k+1} - G^{k+1}_{k+2}$).

Appendix B: Sample Multi-Grid Program and Output. This simple program of Cycle C (written in 1974 by the author at the Weizmann Institute) illustrates multi-grid programming techniques and exhibits the typical behavior of the solution process. For a full description of Cycle C, see Section 4 or the flowchart in Figure 1.

The program solves a Dirichlet problem for the Poisson equation on a rectangle. The same 5-point operator is used on all grids. The I_k^{k-1} residuals transfer is the trivial one (injection), the I_{k-1}^k interpolation is linear. The higher interpolation (A.7) and the special stopping criterion (A.16), recommended for the first $\lceil q/p \rceil$ cycles, are not implemented here.

For each grid G^k we store both v^k and f^k ($k = 1, 2, \ldots, M$). For handling these arrays f^k is also called v^{k+M}. The coarsest grid has $NX0 \times NY0$ intervals of length $H0$ each. Subsequent grids are defined as straight refinements, with mesh-sizes $H(k) = H0/2^{**}(k-1)$. The function $F(x, y)$ is the right-hand side of the Poisson equation. The function $G(x, y)$ serves both as the Dirichlet boundary condition (Φ^M) and as the first approximation (u_0^M). The program cycles until the L_2 norm of the residuals on G^M is reduced below TOL, unless the work WU exceeds WMAX. After each relaxation sweep on any grid G^k, a line is printed out showing the level k, the L_2 norm of the ("dynamic") residuals computed in course of this relaxation, and WU which is the accumulated relaxation work (where a sweep on the finest grid is taken as the work unit).

Note the key role of the GRDFN and KEY subroutines. The first is used to define an $M \times N$ array v^k, i.e., to allocate for it space in the general vector Q (where IQ points to the next available location), and to store its parameters. To use array v^k,

CALL KEY(k, IST, M, N, H)

retrieves the grid parameters (dimension $M \times N$ and mesh-size H) and sets the vector IST(i) so that $v_{ij}^k = Q(\text{IST}(i) + j)$. This makes it easy to write one routine for all arrays v^k; see for example, Subroutine PUTZ(k). Or to write the same routines (RELAX, INTADD, RESCAL) for all levels.

CYCLE C

```
      PROGRAM CYCLE C
      EXTERNAL G,F
      CALL MULTIG(3,2,1.,6,.01,30.,G,F)
      STOP
      END

      FUNCTION F(X,Y)                                  Right-hand side of the equation
      F=SIN(3.*(X+Y))
      RETURN
      END

      FUNCTION G(X,Y)                          Boundary values and first approximation
      G=COS(2.*(X+Y))
      RETURN
      END


      SUBROUTINE MULTIG(NX0,NY0,H0,M,TOL,WMAX,U1,F)
      EXTERNAL U1,F
      DIMENSION EPS(10)                        Multi-grid algorithm (see Fig. 1)
      DO 1 K=1,M
      K2=2**(K-1)
      CALL GRDFN(K,NX0*K2+1,NY0*K2+1,H0/K2)
    1 CALL GRDFN(K+M,NX0*K2+1,NY0*K2+1,H0/K2)
      EPS(M)=TOL
      K=M
      WU=0
      CALL PUTF(M,U1,0)
      CALL PUTF(2*M,F,2)
    5 ERR=1.E30
    3 ERRP=ERR
      CALL RELAX(K,K+M,ERR)
      WU=WU+4.**(K-M)
      WRITE(6,4)K,ERR,WU
    4 FORMAT(' LEVEL',I2,'   RESIDUAL NORM=',1PE10.3,'    WORK=',0PF7.3)
      IF(ERR.LT.EPS(K))GOTO 2
      IF (WU.GE.WMAX)RETURN
      IF(K.EQ.1.OR. ERR/ERRP.LT. .6)GOTO 3                          η=.6
      CALL RESCAL(K,K+M,K+M-1)
      EPS(K-1)=.3*ERR                                               δ=.3
      K=K-1
      CALL PUTZ(K)
      GOTO 5
    2 IF (K.EQ.M)RETURN
      CALL INTADD(K,K+1)
      K=K+1
      GOTO 5
      END


      SUBROUTINE GRDFN(N,IMAX,JMAX,HH)                 Define an IMAX × JMAX
      COMMON/GRD/NST(20),IMX(20),JMX(20),H(20)               array v^N .
      DATA IQ/1/
      NST(N)=IQ
      IMX(N)=IMAX
      JMX(N)=JMAX
      H(N)=HH
      IQ=IQ+IMAX*JMAX
      RETURN
      END


      SUBROUTINE KEY(K,IST,IMAX,JMAX,HH)
      COMMON/GRD/NST(20),IMX(20),JMX(20),H(20)       Set IST such that
      DIMENSION IST(1)                                v^k(I,J)= Q(IST(I) + J),
      IMAX=IMX(K)
      JMAX=JMX(K)                                    and set IMAX = IMX(K)
      IS=NST(K)-JMAX-1                                       JMAX = JMX(K)
      DO 1 I=1,IMAX
      IS=IS + JMAX                                             HH = H(K)
    1 IST(I)=IS
      HH=H(K)
      RETURN
      END
```

```
      SUBROUTINE PUTF(K,F,NH)
      COMMON Q(18000),IST(600)
      CALL KEY (K,IST,II,JJ,H)
      H2=H**NH
      DO 1 I=1,II
      DO 1 J=1,JJ
      X=(I-1)*H
      Y=(J-1)*H
    1 Q(IST(I)+J)=F(X,Y)*H2
      RETURN
      END
```

$v^K \leftarrow H(K)^{NH} \cdot F^K$

```
      SUBROUTINE PUTZ(K)
      COMMON Q(18000),IST(200)
      CALL KEY(K,IST,II,JJ,H)
      DO 1 I=1,II
      DO 1 J=1,JJ
    1 Q(IST(I)+J)=0.
      RETURN
      END
```

$v^K \leftarrow 0$

```
      SUBROUTINE RELAX(K,KRHS,ERR)
      COMMON Q(18000),IST(200),IRHS(200)
      CALL KEY(K,IST,II,JJ,H)
      CALL KEY(KRHS,IRHS,II,JJ,H)
      I1=II-1
      J1=JJ-1
      ERR=0.
      DO 1 I=2,I1
      IR=IRHS(I)
      IO=IST(I)
      IM=IST(I-1)
      IP=IST(I-1)
      DO 1 J=2,J1
      A=Q(IR-J)-Q(IO+J+1)-Q(IO+J-1)-Q(IM+J)-Q(IP+J)
      ERR=ERR+(A+4.*Q(IO+J))**2
    1 Q(IO+J)=-.25*A
      ERR=SQRT(ERR)/H
      RETURN
      END
```

A Gauss-Seidel Relaxation sweep on the equation

$$\Delta_h v^K = v^{KRHS}$$

giving

$$ERR = ||\text{residuals}||_{L_2}$$

```
      SUBROUTINE INTADD(KC,KF)
      COMMON Q(18000),ISTC(200),ISTF(200)
      CALL KEY(KC,ISTC,IIC,JJC,HC)
      CALL KEY(KF,ISTF,IIF,JJF,HF)
      DO 1 IC=2,IIC
      IF=2*IC-1
      JF=1
      IFO=ISTF(IF)
      IFM=ISTF(IF-1)
      ICO=ISTC(IC)
      ICM=ISTC(IC-1)
      DO 1 JC=2,JJC
      JF=JF+2
      A=.5*(Q(ICO+JC)+Q(ICO+JC-1))
      AM=.5*(Q(ICM+JC)+Q(ICM+JC-1))
      Q(IFO+JF) = Q(IFO+JF)+Q(ICO+JC)
      Q(IFM+JF) = Q(IFM+JF)+.5*(Q(ICO+JC)+Q(ICM+JC))
      Q(IFO+JF-1)=Q(IFO+JF-1)+A
    1 Q(IFM+JF-1) = Q(IFM+JF-1)+.5*(A+AM)
      RETURN
      END
```

Linear interpolation and addition

$$v^{KF} \leftarrow v^{KF} + I^{KF}_{KC} v^{KC}$$

```
      SUBROUTINE RESCAL(KF,KRF,KRC)
      COMMON Q(18000),IUF(200),IRF(200),IRC(200)
      CALL KEY(KF,IUF,IIF,JJF,HF)
      CALL KEY(KRF,IRF,IIF,JJF,HF)
      CALL KEY(KRC,IRC,IIC,JJC,HC)
      IIC1=IIC-1
      JJC1=JJC-1
      DO 1 IC=2,IIC1
      ICR=IRC(IC)
      IF=2*IC-1
```

Residuals injection

$$v^{KRC} \leftarrow I^{\text{coarse}}_{\text{fine}} (v^{KRF} - \Delta_h v^{KF})$$

```
      JF=1
      IFR=IRF(IF)
      IFO=IUF(IF)
      IFM=IUF(IF-1)
      IFP=IUF(IF+1)
      DO  1 JC=2,JJC1
      JF=JF+2
      S=Q(IFO+JF+1)+Q(IFO+JF-1)+Q(IFM+JF)+Q(IFP+JF)
    1 Q(ICR+JC)=4.*(Q(IFR+JF)-S+4.*Q(IFO+JF))
      RETURN
      END
```

```
LEVEL 6   RESIDUAL NORM= 2.614E+01   WORK=  1.000
LEVEL 6   RESIDUAL NORM= 2.764E+01   WORK=  2.000
LEVEL 5   RESIDUAL NORM= 2.659E+01   WORK=  2.250
LEVEL 5   RESIDUAL NORM= 2.555E+01   WORK=  2.500
LEVEL 4   RESIDUAL NORM= 2.317E+01   WORK=  2.563
LEVEL 4   RESIDUAL NORM= 2.095E+01   WORK=  2.625
LEVEL 3   RESIDUAL NORM= 1.649E+01   WORK=  2.641
LEVEL 3   RESIDUAL NORM= 1.285E+01   WORK=  2.656
LEVEL 2   RESIDUAL NORM= 7.626E+00   WORK=  2.660
LEVEL 2   RESIDUAL NORM= 3.840E+00   WORK=  2.664
LEVEL 3   RESIDUAL NORM= 5.058E+00   WORK=  2.680
LEVEL 4   RESIDUAL NORM= 8.006E+00   WORK=  2.742
LEVEL 4   RESIDUAL NORM= 2.545E+00   WORK=  2.805
LEVEL 5   RESIDUAL NORM= 9.736E+00   WORK=  3.055
LEVEL 5   RESIDUAL NORM= 2.464E+00   WORK=  3.305
LEVEL 6   RESIDUAL NORM= 1.064E+01   WORK=  4.305
LEVEL 6   RESIDUAL NORM= 2.442E+00   WORK=  5.305
LEVEL 6   RESIDUAL NORM= 2.399E+00   WORK=  6.305
LEVEL 5   RESIDUAL NORM= 2.351E+00   WORK=  6.555
LEVEL 5   RESIDUAL NORM= 2.303E+00   WORK=  6.805
LEVEL 4   RESIDUAL NORM= 2.173E+00   WORK=  6.867
LEVEL 4   RESIDUAL NORM= 2.043E+00   WORK=  6.930
LEVEL 3   RESIDUAL NORM= 1.739E+00   WORK=  6.945
LEVEL 3   RESIDUAL NORM= 1.453E+00   WORK=  6.961
LEVEL 2   RESIDUAL NORM= 9.889E-01   WORK=  6.965
LEVEL 2   RESIDUAL NORM= 6.183E-01   WORK=  6.969
LEVEL 1   RESIDUAL NORM= 2.760E-01   WORK=  6.970
LEVEL 1   RESIDUAL NORM= 5.170E-02   WORK=  6.971
LEVEL 2   RESIDUAL NORM= 2.292E-01   WORK=  6.975
LEVEL 3   RESIDUAL NORM= 5.465E-01   WORK=  6.990
LEVEL 4   RESIDUAL NORM= 7.710E-01   WORK=  7.053
LEVEL 4   RESIDUAL NORM= 1.163E-01   WORK=  7.115
LEVEL 5   RESIDUAL NORM= 8.657E-01   WORK=  7.365
LEVEL 5   RESIDUAL NORM= 1.058E-01   WORK=  7.615
LEVEL 6   RESIDUAL NORM= 9.059E-01   WORK=  8.615
LEVEL 6   RESIDUAL NORM= 1.052E-01   WORK=  9.615
LEVEL 6   RESIDUAL NORM= 1.012E-01   WORK= 10.615
LEVEL 5   RESIDUAL NORM= 9.759E-02   WORK= 10.865
LEVEL 5   RESIDUAL NORM= 9.452E-02   WORK= 11.115
LEVEL 4   RESIDUAL NORM= 8.710E-02   WORK= 11.178
LEVEL 4   RESIDUAL NORM= 7.960E-02   WORK= 11.240
LEVEL 3   RESIDUAL NORM= 6.389E-02   WORK= 11.256
LEVEL 3   RESIDUAL NORM= 4.931E-02   WORK= 11.271
LEVEL 2   RESIDUAL NORM= 2.916E-02   WORK= 11.275
LEVEL 2   RESIDUAL NORM= 1.622E-02   WORK= 11.279
LEVEL 2   RESIDUAL NORM= 1.017E-02   WORK= 11.283
LEVEL 3   RESIDUAL NORM= 1.949E-02   WORK= 11.299
LEVEL 4   RESIDUAL NORM= 3.128E-02   WORK= 11.361
LEVEL 4   RESIDUAL NORM= 8.843E-03   WORK= 11.424
LEVEL 5   RESIDUAL NORM= 3.710E-02   WORK= 11.674
LEVEL 5   RESIDUAL NORM= 8.486E-03   WORK= 11.924
LEVEL 6   RESIDUAL NORM= 4.007E-02   WORK= 12.924
LEVEL 6   RESIDUAL NORM= 9.051E-03   WORK= 13.924
```

OUTPUT

Error reduction by a factor greater than 10 per cycle.

Each cycle costs 4.3 WU

Insensitivity: Results would be practically the same for any $.005 \leq \delta \leq .5$ or any $0 \leq \eta \leq .65$

To solve on the same domain problems other than Poisson, the only subroutines to be changed are the relaxation routine **RELAX** and the residual injection routine **RESCAL**. The latter is always just a slight variation of the first.

For different domains, more general **GRDFN** and **KEY** subroutines should be written. A general **GRDFN** subroutine, in which the domain characteristic function is one of the parameters, has been developed, together with the corresponding **KEY**

routine. This essentially *reduces the programming of any multi-grid solution to the programming of a usual relaxation routine.* See [26].

Appendix C. Rigorous Bound to Model-Problem Convergence Rate. We consider the model problem: 5-point Poisson equation $\Delta_h U_h = F$ on a $(n_1 + 1) \times (n_2 + 1)$ rectangular grid G^M with Dirichlet boundary conditions. Let $n_j = 2^M N_j$ and let G^k be the $(2^k N_1 + 1) \times (2^k N_2 + 1)$ uniform grid on the same domain, with mesh-size $h_k = 2^{-k} h_0$ $(k = 0, 1, \ldots, M)$. We will estimate the convergence rate and work per one multi-grid cycle C^M.

The cycle C^M is defined inductively as follows: (i) Make r relaxation sweeps on the G^M approximate solution u^M. To facilitate the rigorous Fourier analysis we choose as our relaxation the Weighted Simultaneous Displacement (WSD, or "weighted Jacobi") method with weights $\omega_{00} = \omega_0$, $\omega_{01} = \omega_{0-1} = \omega_{10} = \omega_{-10} = \omega_1$, say (see Section 3.3). (ii) Inject (cf. Section A.4) the residual problem to G^{M-1}. (iii) Get an approximate solution v^{M-1} to this G^{M-1} problem by *two* C^{M-1} cycles, starting from the zero approximation. (iv) Correct $u^M \leftarrow u^M + I^M_{M-1} v^{M-1}$, where I^M_{M-1} is linear interpolation. (v) Make s more relaxation sweeps on G^M.

It is easily calculated that one WSD sweep amplifies the Fourier component $\exp(i\theta \cdot x/h_M)$ of the residual by the factor

$$\mu(\theta) = 1 - (2 - \cos\theta_1 - \cos\theta_2)(.5\omega_0 + \omega_1 \cos\theta_1 + \omega_1 \cos\theta_2).$$

Denote by $A(\theta)$ the amplitude, before the C^M cycle, of the $\theta = (\theta_1, \theta_2)$ component of the residual. Actually present on the grid G^M are only components of the form $\theta = (\alpha_1 \pi/n_1, \alpha_2 \pi/n_2)$ $(\alpha_j = \pm 1, \pm 2, \ldots, \pm(n_j - 1))$, and their amplitudes $A(\theta_1, \theta_2) = -A(\theta_1, -\theta_2) = -A(-\theta_1, \theta_2)$ are real (assuming two of the boundary lines to lie on the axes). Since $\mu(\theta_1, \theta_2) = \mu(\pm\theta_1, \pm\theta_2)$ is real, the r relaxation sweeps operate separately on each residual mode, transforming its amplitude $A(\theta)$ to $A'(\theta) = \mu(\theta)^r A(\theta)$.

For any component $\theta = (\theta_1, \theta_2)$ such that $|\theta| = \max(|\theta_1|, |\theta_2|) \leqslant \pi/2$, denote $\theta^1 = (\theta_1, \theta_2)$, $\theta^2 = (\theta_1 \pm \pi, \theta_2)$, $\theta^3 = (\theta_1, \theta_2 \pm \pi)$, $\theta^4 = (\theta_1 \pm \pi, \theta_2 \pm \pi)$, where each $\pm$ sign is chosen so that $|\theta^l| < \pi$ $(l = 1, 2, 3, 4)$. Of these four "harmonics", only the θ^1 mode appears on G^{M-1}, its amplitude there (in the right-hand side of the G^{M-1} residual problem formed in Step (ii)) being

$$A_\theta = A'(\theta^1) + A'(\theta^2) + A'(\theta^3) + A'(\theta^4). \tag{C.1}$$

Let ϵ_k denote an upper bound to the factors by which any C^k cycle reduces the L_2 norm of the residuals on G^k. In particular, the two C^{M-1} cycles (Step (iii)) are equivalent to solving a G^{M-1} problem with amplitudes a_θ instead of A_θ, where

$$\sum_{|\theta| \leqslant \pi/2} |a_\theta - A_\theta|^2 \leqslant \epsilon^4_{M-1} \sum_{|\theta| \leqslant \pi/2} A^2_\theta. \tag{C.2}$$

Interpolating the computed correction from G^{M-1} to G^M (Step (iv)), and then relaxing on G^M (Step (v)), the new residual amplitudes are easily calculated to be

$$\begin{aligned} \bar{A}(\theta^l) &= \mu(\theta^l)^s [A'(\theta^l) - S(\theta^l) a_\theta] \\ &= \mu(\theta^l)^s [\mu(\theta^l)^r A(\theta^l) - S(\theta^l) A_\theta + S(\theta^l)(A_\theta - a_\theta)] \quad (l = 1, 2, 3, 4), \end{aligned}$$

where

$$S(\theta) = \frac{(1 + \cos\theta_1)(1 + \cos\theta_2)(2 - \cos\theta_1 - \cos\theta_2)}{2 - \cos 2\theta_1 - \cos 2\theta_2}.$$

Hence

$$\sum_l \bar{A}(\theta^l)^2 \leqslant 2q^2 \sum_l A(\theta^l)^2 + 2\sum_l S(\theta^l)^2 (A_\theta - a_\theta)^2, \tag{C.3}$$

where q is an upper bound to the L_2 norms of the 4×4 matrices $Q(\theta)$, defined by

$$Q_{lm}(\theta) = \mu(\theta^l)^s (\delta_{lm} - S(\theta^l))\mu(\theta^m)^r \qquad (1 \leqslant l, m \leqslant 4).$$

Denoting $\beta_j = \sin^2\theta_j$, it is easy to check that

$$\sum_l S(\theta^l)^2 = 2 - \beta_1\beta_2 \frac{2 + \beta_1 + \beta_2}{(\beta_1 + \beta_2)^2} \leqslant 2. \tag{C.4}$$

Hence, summing (C.3) over the relevant range of θ, using (C.2) and (C.4) and then (C.1), we obtain

$$\sum_{|\theta| \leqslant \pi} \bar{A}(\theta)^2 \leqslant 2q^2 \sum_{|\theta| \leqslant \pi} A(\theta)^2 + 4\epsilon_{M-1}^4 \sum_{|\theta| \leqslant \pi/2} A_\theta^2$$
$$\leqslant (2q^2 + 4\gamma\epsilon_{M-1}^4) \sum_{|\theta| \leqslant \pi} A(\theta)^2,$$

where γ is any upper bound to all $\Sigma_l \mu(\theta^l)^{2r}$ $(0 < |\theta| \leqslant \pi/2)$. Thus, we have obtained the bound

$$\epsilon_M^2 = 2q^2 + 4\gamma\epsilon_{M-1}^4. \tag{C.5}$$

Choosing $r = s = 1$ and $\omega_0 = 1.07$, $\omega_1 = .155$ (to minimize q^2), straightforward computer calculations show that $q^2 \leqslant .015$ and $\gamma \leqslant 1.18$. From (C.5) it now follows, by induction on M, that $\epsilon_M \leqslant .19$.

The number of operations in the C^M cycle is $W_M \leqslant (12r + 12s + 4)n_1 n_2 + 2W_{M-1}$. Hence, by induction on M, $W_M \leqslant (24r + 24s + 8)n_1 n_2$. We thus have in summary

THEOREM. *The above C^M cycle reduces the L_2 error norm by a factor $\leqslant .19$ and costs 56 operations (additions and multiplications) per grid point.*

The theorem can be improved, in a sense, by defining the C^M cycle to consist of $r + M$ relaxation sweeps and only one C^{M-1} cycle, with sufficiently large r.

(Note, however, that employing arbitrarily large r pays only with simultaneous-displacement schemes on rectangular domains, where eigenfunctions of the coarse-grid correction cycle are also eigenfunctions of the relaxation.)

In practice, .1 reduction is obtained in less than 28 operations-per-point. (See

Appendix B. The Gauss-Seidel sweep employed there can be done in 5 operations-per-point. But for every 3 sweeps on G^k the interpolations I_k^{k-1} and I_{k-1}^k are also performed, respectively costing 6 and 10 operations per 4 grid points. Hence, a work unit in Appendix B should be considered as representing $(3 \times 5 + 6/4 + 10/4)/3 = 19/3$ operations per point.) These operations involve only additions and shifts.

Appendix D. Remarks on Initial-Value Problems. The Multi-Level Adaptive Techniques can be used in time-dependent problems. An obvious use could be made for implicit difference approximations, where a multi-grid algorithm (as in Sections 4 or 5) could be employed to solve, at each time step, the equations for the new time values. Usually, however, alternating-direction implicit (ADI) discretizations, when available, offer the same stability and greater efficiency. The multi-grid approach can be used in cases ADI is difficult to apply, such as nonuniform grids (space mesh-sizes $h^{(1)}, \ldots, h^{(d)}$, and possibly also time step $h^{(0)}$, being nonconstant), curved boundaries, certain nonlinear problems, etc. But note that if a nonuniform grid is constructed as a union of uniform subgrids (cf. Section 7.1), then an ADI procedure may sometimes still be applicable.

Whether the solution is advanced in time explicitly, or by ADI or implicit MG procedure, the automatic adaptive discretization techniques described above (Section 8) can be modified to operate with it, to optimize $h^{(0)}$, $h^{(1)}, \ldots, h^{(d)}$ and other discretization parameters (the approximation orders, both in space and time, the computational boundary for a problem on an unbounded domain, etc.). The main difference is of course that here we have the special coordinate, time, along which we march, usually without iterating.

Each parameter can be adapted locally including $h^{(0)}$, which may vary both in space and in time, provided we impose a structure as in Section 7, in which $h^{(j)}$ is restricted to the values $h_k^{(j)} = 2^{-k} h_0^{(j)}$. This may create time levels for which the boundary is not the natural boundary. The solution on such "internal" boundaries may be defined by extrapolation from past time levels.

Department of Mathematics
Weizmann Institute of Science
Rehovot, Israel

1. N. S. BAKHVALOV (BAHVALOV), "Convergence of a relaxation method with natural constraints on an elliptic operator," Ž. Vyčisl. Mat. i Mat. Fiz., v. 6, 1966, pp. 861–885. (Russian) MR 35 #6378.

2. A. BRANDT, "Multi-level adaptive technique (MLAT) for fast numerical solution to boundary value problems," *Proc. 3rd Internat. Conf. on Numerical Methods in Fluid Mechanics* (Paris, 1972), Lecture Notes in Physics, vol. 18, Springer-Verlag, Berlin and New York, 1973, pp. 82–89.

3. A. BRANDT, *Multi-Level Adaptive Techniques*, IBM Research Report RC6026, 1976.

4. A. BRANDT, "Elliptic difference operators and smoothing rates." (In preparation.)

5. R. P. FEDORENKO, "A relaxation method for solving elliptic difference equations," Ž. Vyčisl. Mat. i Mat. Fiz., v. 1, 1961, pp. 922–927. (Russian) MR 25 #766.

6. R. P. FEDORENKO, "On the speed of convergence of an iteration process," Ž. Vyčisl. Mat. i Mat. Fiz., v. 4, 1964, pp. 559–564. (Russian) MR 31 #6386.

7. J. M. HYMAN, "Mesh refinement and local inversion of elliptic partial differential equations," *J. Computational Phys.*, v. 23, 1977, pp. 124–134.

8. A. JAMESON, "Numerical solution of nonlinear partial differential equations of mixed type," *Numerical Solution of Partial Differential Equations,* III (SYNSPADE 1975) (Proc. Sympos., Univ. of Maryland, College Park, Md., 1975), Academic Press, New York.

9. E. M. MURMAN, "Analysis of embedded shock waves calculated by relaxation methods," *Proc. AIAA Conf. on Computational Fluid Dynamics* (Palm Springs, Calif., 1973), AIAA, 1973, pp. 27–40.

10. C. E. PEARSON, "On non-linear ordinary differential equations of boundary layer type," *J. Math. Phys.*, v. 47, 1968, pp. 351–358. MR 38 #5400.

11. Y. SHIFTAN, *Multi-Grid Method for Solving Elliptic Difference Equations*, M. Sc. Thesis, Weizmann Institute of Science, Rehovot, Israel, 1972. (Hebrew)

12. J. C. SOUTH, JR. & A. BRANDT, *Application of a Multi-Level Grid Method to Transonic Flow Calculations*, ICASE Report 76-8, NASA Langley Research Center, Hampton, Virginia, 1976.

13. R. V. SOUTHWELL, *Relaxation Methods in Engineering Science*, Oxford Univ. Press, New York, 1940. MR 3, 152.

14. R. V. SOUTHWELL, *Relaxation Methods in Theoretical Physics*, Clarendon Press, Oxford, 1946. MR 8, 355.

15. E. L. STIEFEL, "Über einige Methoden der Relaxationsrechnung," *Z. Angew. Math. Phys.*, v. 3, 1952, pp. 1–33. MR 13, 874; erratum, 13, p. 1140.

16. F. de la VALLÉE POUSSIN, "An accelerated relaxation algorithm for iterative solution of elliptic equations," *SIAM J. Numer. Anal.*, v. 5, 1968, pp. 340–351. MR 38 #1845.

17. E. L. WACHSPRESS, *Iterative Solution of Elliptic Systems, and Applications to the Neutron Diffusion Equations of Reactor Physics*, Prentice-Hall, Englewood Cliffs, N. J., 1966. MR 38 #2965.

18. E. L. WACHSPRESS, "Variational acceleration of linear iteration," Proc. Army Workshop Watervliet Arsenal, Albany, New York, 1974.

19. S. V. AHAMED, "Accelerated convergence of numerical solution of linear and non-linear vector field problems," *Comput. J.*, v. 8, 1965, pp. 73–76. MR 31 #5343.

20. I. BABUŠKA, W. RHEINBOLDT & C. MESZTENYI, *Self-Adaptive Refinements in the Finite Element Method,* Technical Report TR-375, Computer Science Department, University of Maryland, 1975.

21. P. O. FREDERICKSON, *Fast Approximate Inversion of Large Sparse Linear Systems*, Math. Report 7-75, Lakehead University, Ontario, Canada, 1975.

22. M. LENTINI & V. PEREYRA, *An Adaptive Finite Difference Solver for Nonlinear Two Point Boundary Problems with Mild Boundary Layers*, Report STAN-CS-75-530, Computer Science Department, Stanford University, Stanford, California, 1975.

23. R. A. NICOLAIDES, "On multiple grid and related techniques for solving discrete elliptic systems," *J. Computational Phys.*, v. 19, 1975, pp. 418–431.

24. A. SETTARI & K. AZIZ, "A generalization of the additive correction methods for the iterative solution of matrix equations," *SIAM J. Numer. Anal.*, v. 10, 1973, pp. 506–521. MR 48 #10148.

25. R. V. SOUTHWELL, "Stress calculation in frameworks by the method of systematic relaxation of constraints. I, II," *Proc. Roy. Soc. London Ser. A*, v. 151, 1935, pp. 56–95.

26. A. BRANDT, "Multi-level adaptive solutions to partial differential equations–ideas and software," *Proc. Sympos. on Math. Software* (Math. Research Center, Univ. of Wisconsin, March 1977), Academic Press. (To appear.)

27. C. WILLIAM GEAR, *Numerical Initial Value Problems in Ordinary Differential Equations,* Prentice-Hall, Englewood Cliffs, N. J., 1971. MR 47 #4447.

28. W. HACKBUSH, *Ein Iteratives Verfahren zur Schnellen Auflösung Elliptischer Randwertprobleme,* Math. Inst., Universität zu Köln, Report 76-12 (November 1976). A short English version: "A fast method for solving Poisson's equation in a general region," *Numerische Behandlung von Differentialgleichungen* (R. Bulirsch, R. D. Grigorieff & J. Schröder, Editors), Lecture Notes in Math., Springer-Verlag, Berlin and New York, 1977.

29. R. A. NICOLAIDES, "On the l^2 convergence of an algorithm for solving finite element equations," *Math. Comp.* (To appear.)

30. R. D. RICHTMYER & K. W. MORTON, *Difference Methods for Initial-Value Problems,* 2nd ed., Interscience, New York, 1967. MR 36 #3515.