ICASE

MULTI-LEVEL ADAPTIVE TECHNIQUES (MLAT) FOR SINGULAR-PERTURBATION PROBLEMS

Achi Brandt

Report Number 78-18
October 12, 1978

INSTITUTE FOR COMPUTER APPLICATIONS IN SCIENCE AND ENGINEERING
NASA Langley Research Center, Hampton, Virginia

Operated by the

UNIVERSITIES SPACE USRA RESEARCH ASSOCIATION

MULTI-LEVEL ADAPTIVE TECHNIQUES (MLAT) FOR SINGULAR-PERTURBATION PROBLEMS

Achi Brandt

Weizmann Institute of Science, Rehovot, Israel

ABSTRACT

The multi-level (multi-grid) adaptive technique is a general strategy of solving continuous problems by cycling between coarser and finer levels of discretization. It provides very fast general solvers, together with adaptive, nearly optimal discretization schemes. In the process, boundary layers are automatically either resolved or skipped, depending on a control function which expresses the computational goal. The global error decreases exponentially as a function of the overall computational work, in a uniform rate independent of the magnitude (ε) of the singular-perturbation terms. The key are high-order uniformly stable difference equations, and uniformly smoothing relaxation schemes.

1. INTRODUCTION

The Multi-Level Adaptive Technique (MLAT) is a general numerical strategy for solving continuous problems such as differential and integral equations and functional minimization problems. It will be discussed here mainly in terms of the numerical solution of partial differential boundary-value problems, with special emphasis on singular-perturbation problems.

The work reported here was performed under NASA Contract No. NAS1-14101 while the author was in residence at ICASE, NASA Langley Research Center, Hampton, VA 23665.

The usual approach is first to discretize the boundary-value problem in some preassigned manner (e.g., finite-element or finite-difference equations on a fixed grid), and then to submit the resulting discrete system to some numerical solution process. In MLAT, however, discretization and solution processes are intermixed with, and greatly benefit from, each other. A sequence of uniform grids (or "levels"), with geometrically decreasing mesh-sizes, participates in the process. The cooperative solution process on these grids involves relaxation sweeps over each of them, coarse-grid-to-fine-grid interpolations of corrections and fine-to-coarse transfers of residuals. This process has two important basic benefits. On one hand it acts as a very fast general solver of the discrete system of equations (including the equations on the finest grid). On the other hand it provides, in a natural way, a flexible, adaptive discretization. For convenience, we discuss these aspects one by one.

1.1. The Fast Solver

In Section 2 of this paper we portrait the multi-level process as a fast solver, i.e., regarding the coarser grids as nothing but auxiliaries for solving the finest-grid equations. The description has appeared before (Brandt (1972), (1977a), (1977b)), but here we add more detailed examples of the solution process (Sec. 2.2), and emphasize some new important aspects. In particular, the "fine-to-coarse correction function" (or the "local relative truncation error") τ_h^H is discussed, together with some of its usages.

One usage, developed in collaboration with N. Dinar, is the so called τ-*extrapolation*. It amounts to a trivial additional operation in the multi-grid program, and costs

negligible amount of extra computing time. But, as shown in Sec. 2.3, it improves the solution, sometimes by very much. With it, at a total computational cost of 4 to 7 work-units (a unit being the work equivalent of one Gauss-Seidel relaxation sweep over the finest grid), a solution u^h is always obtained which is better (i.e., a closer approximation to the true *differential* solution U) than the exact solution U^h of the difference equations. Furthermore, in case some extra smoothness is present, u^h will be some orders-of-magnitude better than U^h; or, alternatively, it may be obtained at a much smaller computational cost.

As other usages of the fine-to-coarse correction function τ_h^H we list, in Section 2.4, some very efficient *methods for making nonlinear continuation (e.g., for bifurcation problems), methods for optimal-control problems, ill-posed boundary-value problems and parabolic time-dependent problems*, as well as fast solution methods that can operate with a limited computer storage. Also mentioned in Section 2.4 are new numerical experiments, made in collaboration with N. Dinar, for the *steady-state incompressible Navier-Stokes equations*, including the singular-perturbation case of large Reynolds numbers. These, and many other experiments briefly referred to, clearly indicate that the above-mentioned multi-grid efficiency (solution in just few work-units) is obtained for general elliptic and non-elliptic systems on general domains.

The fast-solver aspect of the multi-level techniques was studied by various other workers, starting, perhaps, with the "group relaxation" of Southwell (1935). See references in Brandt (1977a), and more recent references in Nicolaides (1978) and Hackbusch (1978b). Most of this work is very theoretical. That is, rigorous asymptotic bounds are

derived for the multi-grid efficiency. The price of rigorosity, of course, is that the results are far from realistic: The proofs hold only for extremely small mesh sizes, and, even for those, the work estimates are orders-of-magnitude too large. (Cf. Section 10 of Brandt (1977a).) The rigorous estimates are too crude, in fact, to yield any useful information; e.g., they cannot resolve the difference between more efficient and less efficient multi-grid processes. For singular-perturbation problems, especially, the rigorous proofs hold only for mesh-sizes which are microscopic compared with the practical ones. For this reason, and since the quantity we try to estimate here is actually nothing but the computer-time, which of course we know anyway (at least aposteriori), a different type of theoretical studies are preferred by the present author. Briefly, discarding rigorosity, it is observed that the important multi-grid processes are of local nature, since long-range convergence is obtained by coarse-grid processes, which cost very little. One can therefore analyze the crucial aspects of multi-grid processes by employing a *local mode (Fourier) analysis*, ignoring for instance far boundaries and changes in the (possibly nonlinear) equations. Experiments with various types of equations (see Dinar (1978) and also Poling (1978)) shows that this analysis (which is much simpler than the rigorous theorems) precisely predicts the multi-grid efficiency. It is therefore very useful in selecting efficient algorithms (see, e.g., Appendix A in Brandt (1977a)), in understanding the numerical results, and in debugging multi-grid programs. It led, in fact, to the efficient algorithms mentioned above, which solve, for example, Navier-Stokes equations in about 7 work-

units. Its mechanized use if mentioned in Section 2.4 and, for singular-perturbation problems, in Section 6.

About the *difficulties in implementing* multi-grid procedures, and what to do about them, some general comments are made toward the end of Section 2.4.

1.2. *Non-Uniform, Adaptible Structures*

Section 3 of this paper (following and adding to Sections 2 and 3 in Brandt (1977b)) discusses the special capability of the multi-level structure to create non-uniform, flexible discretization patterns, especially such patterns as required by singular-perturbation problems, where thin layers should sometimes be resolved, either near or away from boundaries. This capability is obtained by observing that the various grids (levels) need not all extend over the same domain. Finer levels may be confined to increasingly smaller subdomains, so as to provide higher resolution only where desired. Moreover, we may attach to each of these localized finer grids its own local system of coordinates, to fit curve boundaries or to approximate directions of interior interfaces and thin layers. Unlike global coordinate transformation, these local coordinates do not complicate the difference equations throughout the domain (hence do not turn the one-dimensional trouble of boundary approximations into a two-dimensional trouble of complicated equations). All these patches of local grids interact with each other through the multi-grid process, which, at the same time, provides fast solutions to their difference equations (an important advantage over other methods of patching grids or using transformations).

This structure, in which non-uniform discretization is produced through the sequence of uniform grids, is highly flexible. Discretization parameters, such as (finest)

mesh-sizes and approximation orders, can be locally adjusted in any desired pattern, expending negligible amounts of book-keeping work and storage.

In particular, since in this structure only equidistance differencing is needed (much less expensive than differencing on variable grids), it becomes feasible to employ high-order difference approximations, even in singular-perturbation cases (see Section 5).

The discretization can thus be progressively refined and adapted. The actual adaptive solution process is governed by certain criteria, described in Section 3.6. Derived from optimization considerations, these are local criteria which automatically decide where and how to change the local discretization parameters. Furthermore, these criteria are controlled by the user through a certain function G (the error-weighting function), which, in effect, expresses the purpose of the numerical calculations, i.e., the sense (or the error norm) in which approximations to the true solution are to be measured. The resulting discretization will be of high order wherever the evolving solution is suitably smooth. Singularities of all kinds will automatically be detected and treated, usually by introducing increasingly finer levels on increasingly smaller neighborhoods of the singularity.

1.3. MLAT Solutions to Singular Perturbation Problems

The discretization patterns produced by this general adaptive process for singularly-perturbation cases are studied in Section 4. It turns out that boundary layers are sometimes resolved by the adaptive process, and in other cases they are completely "skipped", depending on the choice of the control function G. The decision whether and how to resolve the layer is automatically taken by the adaptive

process itself. In any case, the convergence rate in the suitable sense (i.e., in the error norm corresponding to G) is always fast.

Rates of convergence of adaptive processes are measured by the rate of decrease of the error $E = \|u - U\|$ as a function of the computational work W, where $u = u(W)$ is the evolving numerical solution, U is the true differential solution, and $\|\cdot\|$ is the appropriate error norm. Since the grid is not uniform, nor does it have any fixed number of grid-points, the work-unit in this context must be different from the one mentioned above (which was defined in terms of the finest grid). It can be defined, for instance, as the work of applying the lowest-order difference equations at one grid-point. Thus, for example, in the conventional case, where regular grids are applied for solving a d-dimensional regular differential problem, employing $O(h^p)$ difference approximations, if a fast solver (e.g., a multi-grid algorithm) is used which solves the algebraic equations in $O(h^{-d})$ arithmetic operations, then the rate of convergence can be expressed as $E \leq C(p)W^{-p/d}$. The constant C depends not only on p but also on the solution U.

We show that, *using adaptive discretization, the same p-order convergence rate* $E = O(W^{-p/d})$ *is obtained uniformly in the size* (ε) *of the singular perturbation; that is,* $C(p)$ *does not depend on* ε. *Moreover, if the order-of-approximation* p *is adaptible too, the rate of convergence is uniformly exponential.* More precisely, for cases requiring boundary-layer resolution it is shown that $E \approx cW^{\beta}\exp(-cW^{\alpha})$, with α and c independent of ε. We assume, of course, that the convergence rate for the reduced problem would not be slower. In cases the boundary-layer is

skipped, the rate depends solely on the rate of the reduced problem. We also show that for this type of results it is not necessary to adapt p locally; p *may be an adaptible constant*.

These convergence rates are *uniform*; they are obtained for any ε, small, moderate or large. Nothing actually should be known in advance about the value of ε. It is not even required at all to know that this is a singular-perturbation problem. Most other solution methods, by contrast, solve either the regular case ($\varepsilon = 0(1)$) or the asymptotic case (ε very small), but not the whole range. (Quite often, intermediate values of ε are the most difficult to solve). Here, no special analyses are required, no need to separate the reduced problem from the singular-perturbation, and, in particular, no need to compute the proper reduced boundary conditions. No matching procedures are employed. The method works similarly for interior singular layers, e.g., for ODE problems with turning points, even when (as in nonlinear problems) the location of the singularity is not known in advance.

Although no apriori knowledge is needed about the size of the singular-perturbation, some rules should be kept in dealing with potentially singularly-perturbed problems. As is well known, difference schemes should be constructed which are uniformly stable. This aspect is discussed in Section 5, including the construction of *high-order uniformly-stable difference operators*. Similarly, in the multigrid processing of potentially singular problems, relaxation schemes with *uniform smoothing rates* should be employed. Such schemes are described in Section 6.

Remark. Sections 5 and 6 are abbreviated here. For their full text, see Brandt (1978b). It will include

further discussion of ellipticity and uniform well-posedness of finite-difference systems, as well as remarks on the uniformly well-posed approximation of singular-perturbation problems with highly-oscillating solutions.

1.4. Table of Contents

1. Introduction
   - 1.1. The fast solver
   - 1.2. Non-uniform, adaptible structures
   - 1.3. MLAT solutions to singular-perturbation problems
2. Multi-grid fast solvers
   - 2.1. Basic multi-grid processes
   - 2.2. Full multi-grid run: An example
   - 2.3. Relative local truncation errors and τ-extrapolation
   - 2.4. General properties of multi-grid solutions
3. Non-uniform and adaptive discretizations
   - 3.1. Non-uniformity organized by uniform levels
   - 3.2. Unisotropic refinements
   - 3.3. Difference equations and multi-grid processing
   - 3.4. Remark on high-order approximations
   - 3.5. Local coordinate transformations
   - 3.6. Adaptation techniques
4. Multi-level adaptive solutions to singular-perturbation problems: Case studies
   - 4.1. Optimal discretization of one-dimensional case
   - 4.2. Boundary layer resolution
   - 4.3. Fixed-order and constant-order discretizations
   - 4.4. A case of skipping the boundary layer
   - 4.5. Remarks on more general problems

5. Uniformly stable, high-order discretizations
   5.1. General remarks
   5.2. Examples
6. Relaxations with uniform smoothing rates

2. MULTI-GRID FAST SOLVERS

To understand the basic numerical processes of MLAT, consider first the usual situation where a partial differential problem

$$LU(x) = F(x), \quad \text{for } x = (x_1,\ldots,x_d) \text{ in a domain } \Omega \subseteq R^d, \tag{2.1a}$$

$$AU(x) = \phi(x), \quad \text{for } x \text{ on the boundary of } \Omega, \tag{2.1b}$$

is discretized in a preassigned manner on a given uniform grid G^h, with mesh-size h, yielding the finite-difference equations

$$L^h U^h(x^h) = F^h(x^h), \qquad (x^h \in G^h). \tag{2.2}$$

Here $U = (U_1,U_2,\ldots,U_q)$ and its discrete approximation U^h are q-dimensional vectors of unknown functions, L and A are linear or nonlinear differential operators and $L^h U^h(x^h)$ is, correspondingly, a linear or nonlinear expression involving values of U^h at x^h and at neighboring grid points. At various instances of the solution process, we have on G^h an approximation to U^h, which we will generally denote by u^h.

In this section multi-level techniques for the fast solution of (2.2), with coarser grids serving as auxiliaries, will be described. In this context the term multi-grid, rather than multi-level, can be used. The difference between "grid" and "level" arises only in the more general situation (see Section 3 below).

2.1. Basic Multi-Grid Processes

To obtain a fast solution to equation (2.2) via the multi-grid method, we add to G^h a sequence of coarser uniform grids. Let G^H be such a coarser grid; e.g., let the grid-lines of G^H be every other grid-line of G^h, so that its meshsize is $H = 2h$.

One way of inexpensively obtaining an approximate solution u^h to (2.2) is first to obtain an (approximate) solution u^H to the corresponding coarser problem

$$L^H U^H(x^H) = F^H(x^H), \qquad (x^H \in G^H), \tag{2.3}$$

(which is much less expensive to solve since it contains far fewer unknowns) and then to interpolate u^H to the fine grid:

$$u^h = I_h^H u^H . \tag{2.4}$$

The symbol I_h^H stands for the operation of interpolating from G^H to G^h. Polynomial interpolations of any order can be used. (The optimal order is discussed in Section A.2 of Brandt (1977a). Generally, if m_j is the highest order of derivatives of U_j in L and p is the order of approximation, then an interpolation of order at least $m_j + p$ (i.e., polynomials of degree at least $m_j + p - 1$) should be used for u_j^H to ensure full multi-grid efficiency. In some particular situations, even greater efficiency can be achieved by still higher interpolation; see footnotes 1 and 5 below.)

How good the approximation (2.4) is depends on the smoothness of the solution U^h. In some cases U^h is so smooth that, if the interpolation I_h^H and the coarse grid operator L^H are of order high enough to exploit that smoothness, then u^h obtained by (2.4) satisfies

$$\|u^h - U\| \leq \|U^h - U\| , \qquad (2.5)$$

in some suitable norm. This means that u^h solves (2.2) "to the level of the truncation error", which is all we can meaningfully require in solving (2.2). In such cases, however, the fine grid G^h is not really needed: the coarser grid G^H already yields a solution with the required accuracy. If G^h is at all needed. our first approximation (2.4) will require a considerable improvement.

Can we compute a correction to u^h again by some interpolation from the coarse grid G^H? Namely, can we somehow approximate the error $V^h = U^h - u^h$ by some V^H computed on G^H? Normally[1)], the answer is no. If u^H in (2.4) is a good enough approximation to U^H, then V^h will be a rapidly oscillating function that cannot meaningfully be described on the coarse grid G^H. Therefore, before we can reuse coarse grids, the error V^h should be smoothed out.

An efficient smoothing is obtained by *relaxation sweeps*. A standard example is the Gauss-Seidel relaxation sweep. This is a process in which all points x^h of G^h are scanned one by one in some prescribed order. At each point the old value $u^h(x^h)$ is replaced by a new value, which is computed so that (2.2) is satisfied at that particular point x^h (or nearly satisfied, in case (2.2) is nonlinear at that point; one Newton step of changing $u^h(x)$ is enough). Having completed such a sweep, the system (2.2) is not yet solved, because its equations are coupled to each other; but the new approximation u^h is hopefully "better" than the old one.

In fact, a well known, and extensively used, method for solving (2.2) is by a long sequence of relaxation sweeps. When the system (2.2) is linear, convergence of u^h to U^h is obtained by a sequence of relaxation sweeps if and only

if the system is definite. But the rate of convergence is very slow. Typically, if m is the order of L and N_i is the number of grid intervals in the x_i direction, then the number of sweeps required for convergence is proportional to $(\min[N_1,\ldots,N_d])^m$.

A closer examination, e.g., by Fourier analysis, of the error V^h, shows that the components slow to converge are those whose wavelength is large compared with the mesh-size of h. The high-frequency components, however, converge very fast; they practically disappear after a few sweeps. For example, in Gauss-Seidel relaxation for the 5-point Laplace operator

$$L^hU^h(x,y) \equiv \Delta_h U^h(x,y) \equiv \frac{1}{h^2}\{U^h(x+h,y) + U^h(x-h,y) + U^h(x,y+h) + U^h(x,y-h) - 4U^h(x,y)\}, \qquad (2.6)$$

the convergence factor of the Fourier component $\exp[i(\theta_1 x + \theta_2 y)/h]$ (i.e., the factor by which the magnitude of its amplitude in the error expansion is reduced by one sweep) is

$$\mu(\theta_1,\theta_2) = \left|\frac{e^{i\theta_1} + e^{i\theta_2}}{4 - e^{-i\theta_1} - e^{-i\theta_2}}\right| . \qquad (2.7)$$

For the longest components $\theta_j = O(N_j^{-1})$, and hence $\mu = 1 - O(N_1^{-2} + N_2^{-2})$. But for high-frequency components, say with $\max|\theta_j| \geq \frac{\pi}{2}$, we have $\mu \leq .5$, so that in three relaxation sweeps these components are reduced by almost an order of magnitude.

This means that *relaxation sweeps, inefficient as they are in solving problems, are very efficient in smoothing out the error* V^h. This property, which is extensively used in multi-level algorithms, is very general. It holds for

Gauss-Seidel relaxation of any uniformly elliptic scalar ($q = 1$) difference operator, whether linear or nonlinear. For elliptic *systems* ($q > 1$), efficient smoothing is obtained by suitable variants of the Gauss-Seidel relaxation. Even degenerate and singularly-perturbed elliptic operators are smoothed out with similar efficiency, provided more sophisticated variants are used, such as line relaxations in suitable directions, or "distributed" Gauss-Seidel relaxations. Moreover, some of these variants are very efficient even for non-elliptic systems. (See Section 3 in Brandt (1976), Section 3 in Brandt (1977a), Lectures 5, 6 and 7 in Brandt (1978a) and Section 6 below.) It is also important to note that fortunately the smoothing efficiency does not depend on some sensitive relaxation parameters. Such parameters are sometimes needed (e.g., a relaxation factor is required in simultaneous-displacement relaxation schemes, which are used in conjunction with vector or parallel processing); but since smoothing is a local process, the optimal values of the parameters depend on the local operator only, and can easily be calculated by local Fourier analysis. Large deviations from the optimal values have only mild effect.

Thus, after a couple of relaxation sweeps, the error v^h is smooth, and a good approximation to it can inexpensively be computed on the coarser grid G^H. To see how this is done in the general nonlinear case, observe that on G^h the equation satisfied by v^h is the "*residual equation*"

$$\hat{L}^h v^h(x^h) = r^h(x^h), \qquad (x^h \in G^h), \tag{2.8a}$$

where[10]

$$\hat{L}^h v^h \equiv L^h(u^h + v^h) - L^h u^h, \tag{2.8b}$$

$$r^h \equiv F^h - L^h u^h. \tag{2.8c}$$

In the Linear case $\hat{L}^h = L^h$, and on first reading one may like to keep this case in mind. (2.8) is of course fully equivalent to (2.2), but we are interested in this form because V^h, not U^h, is the smooth function which we like to approximate on the coarser grid G^H. r^h is the *"residual function"*, and, like V^h, it is smoothed-out by relaxation. The approximation to (2.8) on the coarse grid is

$$L^H(I_h^H u^h + V^H)(x^H) - L^H(I_h^H u^h)(x^H) = \bar{I}_h^H r^h(x^H), \qquad (x^H \in G^H), \qquad (2.9)$$

where V^H is designed to be the coarse-grid approximation to V^h, and I_h^H and $\bar{I}_h^H$ are interpolation operators (not necessarily the same) from G^h to G^H. Since the points of G^h are often a subset of the points of G^H, one can actually use direct "injection", i.e., $I_h^H u^h(x^H) = u^h(x^H)$. In many cases, however, it is preferrable to use "full weighting", i.e., to use $I_h^H u^h(x^H)$ which is a weighted average of values $u^h(x^h)$ at points $x^h \in G^h$ near x^H, in such a way that all values $u^h(x^h)$ equally contribute to the coarse-grid values.

Observe that at this stage we could *not* approximate the equation $L^h(u^h + V^h) = f^h$ on the coarse grid by the simpler approximation

$$L^H(I_h^H u^h + V^H) = I_h^H f^h , \qquad (2.9')$$

since the error of this approximation depends on the rapidly-oscillating part of u^h, which may be large compared with the function V^h we seek to approximate. In (2.8), by contrast, even if V^h is small, the left-hand side is still approximately a linear operator in V^h, and the left-hand side of (2.9) nicely approximates[2] that linear operator. In fact, the coefficients of that quasi-linear operator on

the fine grid depend on u^h, while on the coarse grid they have similar dependence on $I_h^H u^h$. Hence, if I_h^H is a proper averaging operator, the coarse grid coefficients will automatically be averages of the fine grid coefficients, so that even if u^h is highly oscillatory, the coarse-grid equation is a proper "homogenization" of the fine-grid equation. (For discussions of homogenization see, e.g., Babuška (1975) and Spagnolo (1975).)

Observe also that residuals r^h are defined, and are transferred (with some averaging) to the coarser grid, not only with respect to the interior equations, but also with respect to the boundary conditions. In order that such transfers are done in the right scale, it is important that (i) the difference equations (2.2), and similarly (2.3), approximate (2.1) without change of scale (e.g., without multiplying through by h^m. Equations (2.8)-(2.9) refer to the *divided* form of the difference approximations. Keeping this in mind, one can of course write the *program* with differently scaled equations, provided r^h is multiplied by a suitable constant when transferred to the coarser grid.) (ii) Difference-equations approximating different differential equations should be clearly separated. For example, do not scramble together equations approximating (2.1a) with equations approximating (2.1b). Do not incorporate the boundary condition into the neighboring interior equation. Failure to observe these rules is a common error in multilevel programming.

To avoid complicated linearization in solving (2.9), a new unknown function

$$U_h^H = I_h^H u^h + v^H \tag{2.10}$$

is introduced (instead of v^H) on G^H, in terms of which

(2.9) becomes

$$L^H U_h^H(x^H) = F_h^H(x^H), \qquad (x^H \in G^H), \qquad (2.11a)$$

where[10]

$$F_h^H = L^H(I_h^H u^h) + \bar{I}_h^H r^h . \qquad (2.11b)$$

The advantage of this new form is that it is the same equation as (2.3), except for a different right-hand side. (The difference between the two right-hand sides is an important quantity which will be exploited below. See Section 2.3.) Moreover, (2.11) and (2.3) has the same form as (2.2). Hence, the same routines (e.g., the same relaxation routine) can be used in treating all of them. (See for example the simple sample program in Appendix B of Brandt (1977a).)

It is also worth noting that our new unknown (2.10) represents, on the coarse grid, the sum of the basic approximation u^h and its correction V^h. Thus, u_h^H is the full current approximation, represented on G^H. The scheme of using u_h^H is therefore called the *Full Approximation Scheme (FAS)*. To be distinguished from the Correction Scheme (CS), which directly uses V^H. (The Correction Scheme is messy in nonlinear problems, and cannot be applied on composite grids (see Section 3). We therefore continue our discussion here only in terms of the more general scheme FAS.) At convergence, when $u^h = U^h$ and $V^h = 0$, we have $U_h^H = I_h^H U^h$. Thus U_h^H is a coarse-grid function which coincide with the fine-grid solution - a fact which will also be very useful below (Sections 2.3 and 2.4).

Once (2.11) is solved (or approximately solved), we want to use its solution to correct the basic approximation u^h. In doing so we should keep in mind that $V^H = U_h^H - I_h^H u^h$, and not u_h^H itself, is the function which approximates a

smooth fine-grid function, and hence v^H (or its computed approximation) is what we should interpolate to the fine grid and use it there as a correction to u^h. Thus, denoting our approximate solution to (2.11) by u_h^H, the corrected approximation on the fine grid should be

$$u_{NEW}^h = u_{OLD}^h + I_H^h(u_h^H - I_h^H u_{OLD}^h) \ . \tag{2.12}$$

Observe that this interpolation is *not* equivalent to

$$\bar{u}_{NEW}^h = I_H^h u_h^H \ , \tag{2.13}$$

since $I_H^h I_h^H$ is not the identity operator. The important difference is that (2.12) preserves the high-frequency information contained in u_{OLD}^h, while (2.13) does not use u_{OLD}^h, and thus loses this information. Interpolation of the type (2.12) is called *FAS interpolation*.

The order of the interpolation I_H^h in (2.12) need not be as high as in the first coarse-to-fine interpolation (2.4). Order m_j is enough (cf. the discussion following (2.4)). For example, if the differential equation is of second-order, then linear interpolation is enough.

We summarize the basic processes above: To solve the fine-grid equations (2.2), an initial approximation (2.4) is obtained from an approximate solution u^H to the coarse grid equation (2.3). Then the approximation is improved by a "multi-grid cycle". This cycle includes a couple of relaxation sweeps followed by the "coarse-grid correction" (2.12), in which u_h^H is an approximate solution to the coarse-grid correction equations (2.11).

In most cases, at the end of the multi-grid cycle the approximation u^h will satisfy (2.5) and therefore require no further improvement. This is because the relaxation sweeps effectively liquidate the high-frequency

components of the error, while the coarse-grid correction liquidates the lower components. In fact, we will see below (Section 2.3) that, with some modification in the algorithm, at the end of one multi-grid cycle $\|u^h - U\|$ may be much smaller than $\|U^h - U\|$. If, however, for any reason, a greater accuracy in solving (2.2) is desired, additional multi-grid cycles can be performed. Typically, each cycle which includes three sweeps of a suitable relaxation scheme will reduce $\|u^h - U^h\|$ by a factor of .2 to .08.

We still have to specify how *the coarse-grid equations,* first (2.3) and later (2.11a), are actually solved. They *are solved in the same way that (2.2) is solved, namely, by a combination of relaxation sweeps and coarse-grid corrections, using a grid still coarser than* G^H. More precisely, (2.3) is solved by a first approximation obtained from a still-coarser grid (grid G^{2H}, for example), and then a multi-grid improvement cycle (using G^{2H} again). For solving equation (2.11a) the first approximation is $I_h^H u^h$; one multi-grid cycle (using G^{2H}) is enough for improving this approximation to the required level of accuracy.

The full algorithm has several variations. One is flow charted here as Figure 1. This is essentially the same algorithm as described in Section 1.3 of Brandt (1977b). Sample runs of it can be produced by the MUGTAPE (1978a) program FASFMG. It contains three *switching parameters*: α, δ and η. Usually $\alpha = 2^{-p}$, where p is the approximation order. Optimal values for δ and η are discussed in Sections A.6 and A.7 of Brandt (1977a). In practice the precise optimization is not important. One can take $\eta = \bar{\mu}$ and $\delta = \bar{\mu}^r$, where $\bar{\mu}$ is an estimate for the smoothing factor (computed for example by SMORATE; cf. Section 6),

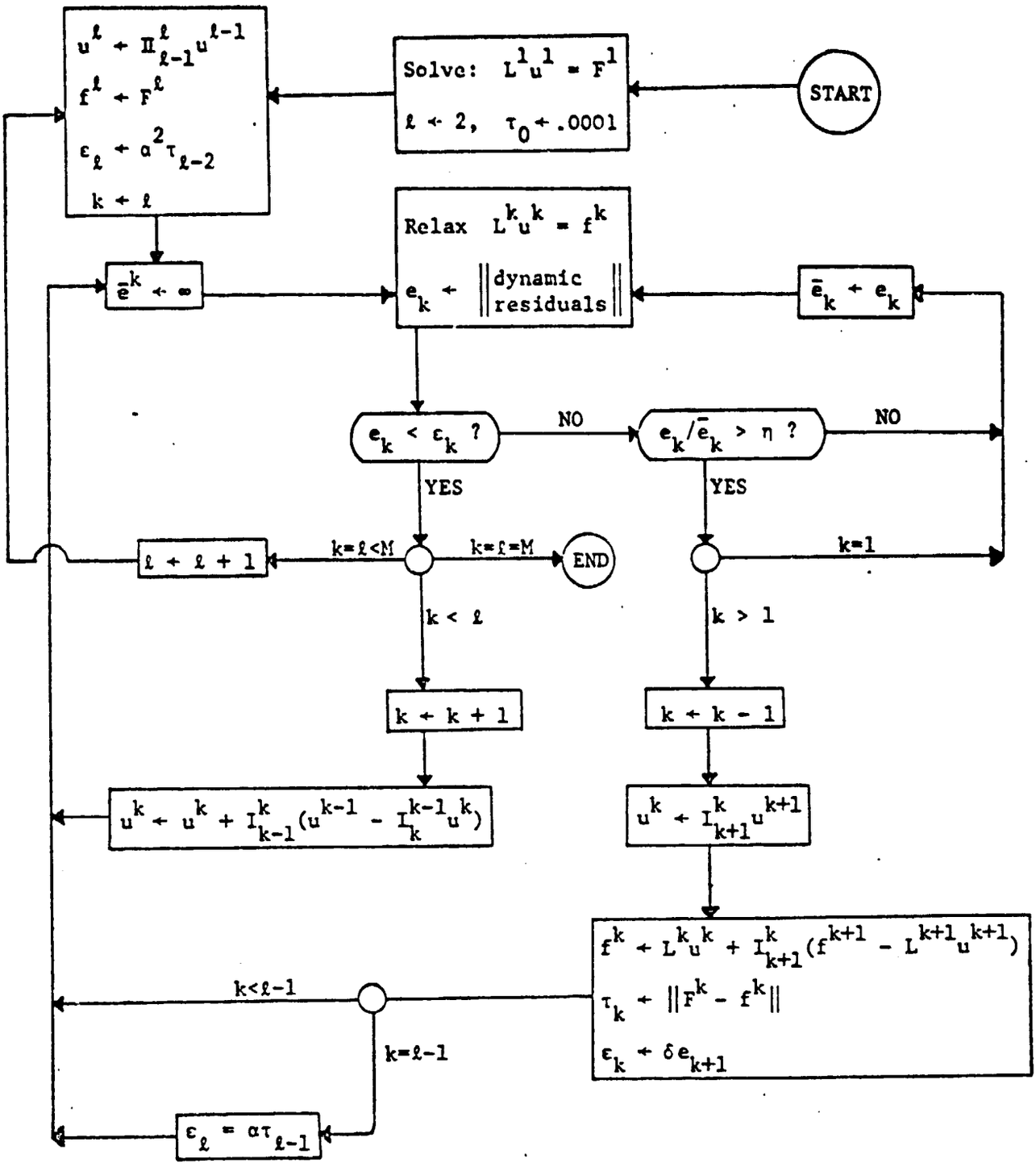


Fig. 1. FAS Full Multi-Grid (FAS FMG) Algorithm.

(See Legend on next page.)

Fig. 1. FAS Full Multi-Grid (FAS FMG) Algorithm. In this flowchart, as in the program itself, the different levels (grids) are not labelled by their mesh-size, as in the text, but by a positive integer k. $k = 1$ *is the coarsest level,* $k = M$ *is the ultimately finest level, and* $k = \ell$ *is the currently finest one (i.e., the finest so far used by the algorithm).*

Thus, the original finite-difference equation on level k *(before being changed to serve as a correction equation like (2.11)) is* $L^k u^k = f^k$. u^k *denotes the current approximation, and* f^k *the current right-hand side, on level* k. I^k_{k-1} *denotes interpolation of order* m *(the order of the differential equation) from level* $k-1$ *to the next finer level* k. $\Pi^\ell_{\ell-1}$ *denotes higher-order (*$m + p$ *order) interpolation.* I^k_{k+1} *denotes transfer (by some averaging) from level* $k+1$ *to the next coarser level* k. τ_k *denotes an approximate measure of the local truncation errors on level* k *(cf. Section 2.3).* e_k *is a measure of the residuals, taken during the relaxation sweep on level* k. $\bar{e}_k$ *is the value of* e_k *at the previous sweep, so that* $e_k / \bar{e}_k > \eta$ *signals slow convergence.* ε_k *is a tolerance designed so that* $e_k < \varepsilon_k$ *signals convergence of the current k-level problem. For* $k < \ell$ *the k-level problem is the correction problem to level* $k+1$, *hence* $\varepsilon_k = \delta e_{k+1}$. *On the currently finest level* $(k = \ell)$ *we need convergence to within the estimated size of the truncation error. Hence* $\varepsilon_\ell = \alpha\tau_{\ell-1}$, *and before* $\tau_{\ell-1}$ *is known* $\varepsilon_\ell = \alpha^2\tau_{\ell-1}$. *The parameters* η, δ *and* α *are discussed in the text.*

and r is the number of relaxation sweeps per multi-grid cycle. With good relaxation schemes $\bar{\mu} \approx .5$ and $r \approx 3$.

A slightly different algorithm is shown step-by-step in Table 1, in the next section. That algorithm can also be run through program FASFMG (using its FASFIX subroutine). The difference between the two is that the algorithm in Fig. 1 is "accommodative", its flow depends on some internal checks, while that in Table 1 is "fixed", its entire flow is prescribed in advance, depending only on some input parameters.

2.2. Full Multi-Grid Run: An Example

Table 1 shows the steps and the results of a multi-grid solution process for the 5-point Poisson equation (cf. (2.6))

$$\Delta_h U^h(x^h) = F(x^h), \qquad (x^h \in G^h), \qquad (2.14)$$

where G^h is a 97×65 grid with mesh-size $h = 1/32$, covering the rectangle $\{0 \le x_1 \le 3,\ 0 \le x_2 \le 2\}$. Dirichlet boundary conditions are given on the boundary of the rectangle. In this particular run the boundary conditions and $F = \Delta U$ were chosen so that the exact solution to the differential problem is $U = \sin(3x_1 + 3x_2)$.

The flow of the algorithm can be seen from the first column of the table. It tells us the mesh-size H of the grid on which a Gauss-Seidel (GS) relaxation sweep is made. Thus, the process starts with 5 sweeps on G^{16h}, a 7×5 grid with mesh-size $H = 16h = 1/2$, starting with the approximation $u^{16h} \equiv 0$. Since the grid is very coarse, after 5 sweeps u^{16h} solves (2.3) well below the truncation level. This last u^{16h} is then *cubic*-interpolated (i.e., interpolation of order 4) to the finer grid G^{8h} to serve there as the first approximation u^{8h}, as indicated

TABLE I

Output of Multigrid Runs for a Poisson Problem, Produced by the MUGTAPE (1978) Program FASFMG

Level H	$\|r^H\|_G$	Relaxation Work	$\|u^H - U\|_\infty$ Usual	$\|u^H - U\|_\infty$ τ-extrap.	$\|U^H - U\|$
16h	2.11	.0039			
16h	.750	.0078			
16h	.270	.0117			
16h	.104	.0156			
16h	.0417	.0195	.25	.25	.249
Cubic interpolation $u^{8h} = I_{16h}^{8h} u^{16h}$					
8h	3.73	.0351			
8h	1.34	.0508	.150	.150	
				$\tau_{8h}^{16h} \leftarrow 4\tau_{8h}^{16h}/3$	
16h	.570	.0547			
16h	.269	.0589	.0611	.0600	
32h	.203	.0596			
32h	.0262	.0605			
32h	.00164	.0615	.0586	.0538	
16h	.145	.0654	.0765	.0600	
8h	.492	.0811	.0798	.0499	.0572
Cubic interpolation $u^{4h} = I_{8h}^{4h} u^{8h}$					
4h	1.01	.144			
4h	.602	.206	.0645	.0407	
				$\tau_{4h}^{8h} \leftarrow 4\tau_{4h}^{8h}/3$	
8h	.418	.222			
8h	.288	.237	.0398	.0246	
16h	.150	.241			
16h	.0473	.245	.0117	.0517	
32h	.0114	.246			
32h	.000095	.247			

TABLE I - Continued

32h	.000006	.248	.00713	.00458	
16h	.0145	.252	.0123	.00565	
8h	.0988	.266	.0162	.00731	
4h	.201	.330	.0180	.00525	.0138
Cubic interpolation $u^{2h} = I_{4h}^{2h} u^{4h}$					
2h	.267	.580			
2h	.205	.830	.0168	.00467	
			$\tau_{2h}^{4h} + 4\tau_{2h}^{4h}/3$		
4h	.177	.893			
4h	.158	.955	.0146	.00387	
8h	.109	.971			
8h	.0752	.986	.00893	.00242	
16h	.0378	.990			
16h	.0139	.994	.00206	.000870	
32h	.00551	.995			
32h	.000715	.996			
32h	.000045	.997	.00223	.000109	
16h	.00435	1.00	.00254	.000253	
8h	.0274	1.02	.00364	.000226	
4h	.0504	1.08	.00404	.000675	
2h	.0665	1.33	.00419	.000472	.00344
Cubic interpolation $u^{h} = I_{2h}^{h} u^{h}$					
h	.0680	2.33			
h	.0569	3.33	.00412	.00448	
			$\tau_{h}^{2h} + 4\tau_{h}^{2h}/3$		
2h	.0540	3.58			
2h	.0513	3.83	.00394	.000411	
4h	.0455	3.89			
4h	.0393	3.95	.00340	.000313	
8h	.0278	3.97			
8h	.0192	3.99	.00206	.000167	
16h	.00945	3.99			
16h	.00355	3.99	.000562	.0000594	
32h	.00138	3.99			
32h	.000123	3.99			
32h	.000008	4.00	.000540	.0000231	

TABLE I - Continued

16h	.00113	4.00	.000625	.0000261	
8h	.00684	4.02	.000879	.0000176	
4h	.0127	4.08	.000984	.0000337	
2h	.0163	4.33	.00102	.0000438	
h	.0182	5.33	.00103	.0000518	.00086

in the table. Two GS sweeps are then made on G^{8h}. Then, the table shows, a switch is made back to the coarser grid G^{16h}. Such a "switch to the coarser grid" means that coarse-grid correction equations such as (2.11) are set up on G^{16h}, namely, their right-hand side is calculated and an initial approximation $u^{16h} = I_{8h}^{16h} u^{8h}$ is introduced. Then, as shown, 2 relaxation sweeps are made on those G^{16h} equations starting with that approximation. Then a switch is made to the still-coarser grid G^{32h} (our coarsest grid here, which is 4×3 and has therefore only 2 interior points), where 3 sweeps are made. Then a switch back to the finer grid G^{16h} is made. Such a "switch to the finer grid" means that a FAS interpolation, like (2.12), is made from G^{32h} to G^{16h}. These interpolations are all of order 2, that is, *linear* interpolations. One relaxation on G^{16h}, then switch back to G^{8h} and one sweep on it, and a multigrid cycle for G^{8h} has been completed. At this point u^{8h} solves (2.3) to the level of its truncation error, so we can already use it, with *cubic* interpolation, as the initial approximation on G^{4h}. Etc., etc., the first column in Table 1 shows the flow all the way to a final approximation on the finest grid ($\tilde{h} = h$).

At each relaxation sweep over G^H the residuals $r^H(x^H)$ are measured and their norm is accumulated. This norm is shown on the second column of Table 1. The precise

definition of the measured quantity $\|r^H\|_C$ is not important at this point. In this specific run we chose the following definition: r^H are the *dynamic* residuals, i.e., $r^H(x^H)$, as defined by (2.8c), is calculated using values of u^H as they stand immediately before the point x^H is scanned by the relaxation sweep, that is, immediately before the value $u^H(x^H)$ is changed. This type of residual is least expensive to calculate, since it is (almost) calculated anyhow in the course of computing the new value of $u^H(x^H)$. The norm $\|\cdot\|_C$ used in the table is the grid analog of the continuous norm

$$\|r\|_C = \frac{\int G(x)|r(x)|dx_1dx_2}{\int G(x)dx_1dx_2} \tag{2.15}$$

where G is some function related to the error functional we are interested in (see Section 3.6). In this particular run we had $G(x) = C_x^2$, where C_x is the distance of x from the nearest corner.

The third column of Table 1 shows the accumulated work, measured in *work units*. A work unit is the work of one relaxation sweep on the finest grid G^h. Hence, a relaxation sweep on G^{Nh} is counted as N^{-2} work units, since it contains about N^{-2} the number of grid-points contained in G^h. This work count neglects any other work except that of relaxation sweeps, because (i) Relaxation sweeps account for at least 70% of the total actual work. (ii) The other work, mostly that of interpolations, is not directly expressible in terms of work units. We could measure running time, but this depends too much on the particular hardware, software and program being used. (iii) The theoretical prediction of

convergence rates is made, too, in terms of these work units, so it is convenient to use them for comparing experiments with theory.

Observe how little is the work accumulated on the coarser grids. In sum, this solution algorithm requires only 5.33 work units. A more precise count of *all* operations (including interpolations) shows that, if $\|r^H\|$ is not computed[3], *this algorithm requires less than 42*n *arithmetic operations, where* n *is the total number of points in the finest grid.*

Incidentally, in this particular problem (2.14), most of these arithmetic operations are additions. In fact, one can arrange it so that the only multiplications (and divisions) involved are by factors 4, 2, 1/2 or 1/4, which in binary floating-point arithmetic can be performed as additions. (For this purpose, cubic interpolation should be performed through the difference operator itself, as for example in Hyman (1977).) Thus, *no multiplications or divisions are required.*

In experiments at ICASE made by Craig T. Poling, the time measured for this algorithm on a CDC 6600 was .083 seconds for a 33 × 33 grid, and .303 for a 65 × 65 grid. A similar algorithm (using another kind of relaxation) on CDC STAR-100 computer required .011 seconds for a 65 × 65 grid and .0347 seconds for a 129 × 129 grid, i.e., about 2 microseconds per grid point.

The first three columns in Table 1 exhibit the standard output of a multi-grid run. The last three columns can be produced only in experimental runs made for cases in which the exact differential solution U is known. The purpose of these columns is to show the quality of the computed approximations u^H. Here, u^H denotes the current approximation on G^H. Thus, in the coarse of a correction cycle

for level $h_1 < H$, u^H denotes the full approximation on level G^H, similar to (2.10). The exact solution of the difference equations (2.3) on grid G^H is denoted by U^H.

The fourth column of Table 1 shows, at various stages, the maximum difference between the exact differential solution U and the computed u^H, where the values of u^H at each stage are those obtained at the *end* of the G^H relaxation sweep corresponding to that row of the table. The fifth column of the table will be explained in Section 2.3. The sixth column gives, for each level H, the maximum discretization error $|U^H - U|$. It is shown on the row corresponding to the end of the multi-grid correction cycle for level H. The main observation is, of course, that at this stage $\|u^H - U\|$ is not considerably bigger that $\|U^H - U\|$, so *the algorithm indeed solves the discrete problem to the level of the discretization error.*

This performance is predictable, and will be the same for any other data. The local mode analysis shows that the multigrid cycle used here reduces $\|u^h - U^h\|$ by a factor .08. A factor .25 would in fact suffice, since all we need in this cycle is to reduce the error from approximately $\|U^{2h} - U\|$ to approximately $\|U^h - U\|$.

Moreover, observe the row one before the last in Table 1. It shows that already at this stage u^{2h} solves the problem to the level of the h discretization error, i.e., $\|u^{2h} - U\|$ is not much bigger than $\|U^h - U\|$. Hence, *the level of the* h *discretization error is actually obtained in only* 4.3 *work-units. The total number of arithmetic operations (additions and subtractions) required is only about* 35n. The full 42n operations are required only if we need the solution, to the same accuracy, at all points of G^h, not only at points of G^{2h}.

Another interesting comment which follows concerns the storage requirement for this algorithm. If the algorithm indeed terminates at 4.33 work-units, no FAS interpolation to G^h is made. Hence the values of u^h need not be stored. All the operations made on G^h are the initial cubic interpolation from G^{2h}, followed by two relaxation sweeps and the calculation of F_h^{2h}. All these operations can be made by one pass over G^h, requiring in storage only 5 columns of the grid at a time. (The first sweep of relaxation is made on the fourth of these columns, then the second sweep can already be made for the third column and the residual transfer can then be made for the second column.) Thus, *the algorithm requires no storage (not even external storage) for the finest grid.* The storage required for the coarser grids is only $\frac{1}{4} + \frac{1}{16} + \cdots \leq \frac{1}{3}$ that of the finest grid. A modified multi-grid algorithm can work with even smaller storage (see Section 2.4).

2.3. *Relative Local Truncation Errors and τ Extrapolation*

To realize further aspects of the multi-grid method, we now slightly shift our point of view. Going back to the coarse-grid correction equations (2.11), we rewrite them, using (2.8c), in the form

$$L^H U_h^H = F^H + \tau_h^H \tag{2.16}$$

where $F^H = \bar{I}_h^H F^h$ and

$$\tau_h^H = L^H(I_h^H u^h) - \bar{I}_h^H(L^h u^h) \ . \tag{2.17}$$

At convergence, where $u^h = U^h$, we have $V^h = 0$ and, by (2.10), $U_h^H = I_h^H U^h$. Hence, τ_h^H actually represents the *local truncation error of* G^H *relative to* G^h, i.e., the error which arises when the fine-grid solution U^h is

substituted in the coarse-grid equation (2.3). Compare this to the usual local truncation error, i.e., to the error τ^h which arises when the true *differential* solution U is substituted into the G^H equations (2.3); namely

$$\tau^H = L^H U - LU \ . \tag{2.18}$$

We can now reverse our viewpoint. Instead of regarding the coarse-grid as a devise for calculating the correction (2.12) to the fine grid solution, we can view the fine grid as a devise for calculating the correction τ_h^H to the coarse-grid equations, a correction which will make the solution of these coarse-grid equations to coincide (up to interpolation) with the fine-grid solution. τ_h^H is therefore called the *fine-to-coarse correction function*. It is a kind of defect correction (cf. Sec. 3.4). This point of view open many more algorithmic possibilities, such as the multi-grid method for non-uniform discretization (Section 3.3 below), continuation techniques, methods for ill-posed, optimal-control, and time-dependent problems, and small-storage procedures (Section 2.4).

The quantity τ_h^H itself is very useful. First, it is an approximation to the local truncation error τ^H. More precisely, we see from (2.17)-(2.18) that

$$\tau_h^H \approx \tau^H - \tau^h \ . \tag{2.19}$$

This relation will be used in the adaptive processes (Section 3.6). Here we show how to use it to improve the multi-grid solution.

The local truncation error can be expanded in Taylor series, yielding always a relation of the form

$$\tau^h(x) = C(x)h^p + O(h^{\bar{p}}), \qquad (\bar{p} > p) \ , \tag{2.20}$$

where $C(x)$ depends on the (unknown) solution, but does not depend on h. Applying (2.20) also to τ^H, and using

(2.19), we find

$$\tau^H \approx \frac{1}{1-\eta^p}\,\tau_h^H + O(H^{\bar{p}}), \qquad (\eta = \frac{h}{H}, \text{ hence usually } \eta = \frac{1}{2}) \quad (2.21)$$

Thus, if we replace τ_h^H in the coarse-grid correction equations (2.16) by $\tau_h^H/(1-\eta^p)$, we effectively introduce the *true* truncation error (up to order $H^{\bar{p}}$) instead of the *relative* one, and the solution U_h^H should become a much better approximation to the true differential solution U. We call this replacement a *local truncation extrapolation*, or, briefly, τ*-extrapolation*.

Note that τ_h^H is defined with respect to both the interior difference equations and the boundary conditions, hence τ-extrapolation can be applied for both.

The τ-extrapolation costs very little. Only one operation (multiplication by $(1-\eta^p)^{-1}$) is added, and only at coarser - grid points, so it amounts to less than $\frac{1}{3}$ operation per fine-grid-point. The stages in the algorithm where this extrapolation is made are shown in the fifth column of Table 1. Also shown in that column are the results of this operation in terms of the error $\|u^H - U\|_\infty$ at various stages. We see that with exactly the same flow[4], the algorithm produces now much smaller errors; in fact u^h *now, after* 4.3 *work-units, is a much better approximation (to* U*) than the exact finite-difference solution* U^h.
With more work and some changes in the algorithm (quintic instead of cubic interpolations, and two instead of one multi-grid cycles for each level, τ-extrapolation being made on the second of the two), $\|u^h - U\|_\infty$ could be made much smaller yet[5].

The impressive improvement depends of course on the smoothness of the solution. Similar improvements could be

obtained, in some such cases, by using Richardson extrapolation (extrapolating from the *solutions* u^H and u^h). Indeed, in case the solution is *not* smooth, e.g., if it oscillates wildly on the scale of G^h, the τ-extrapolation does not considerably improve u^H. But exactly in this case the one multi-grid cycle is enough to reduce $\|u^h - U^h\|$ well below $\|U^h - U\|$, since, exactly in this case, τ^h is not considerably smaller than τ^H. So the point of the τ-extrapolation is that it can always be used, for negligible extra cost in either programming or computer time, and it produces u^h that, at 4.3-5.3 work-units, is guaranteed to be no worse than U^h, the full solution of the difference equations, with the nice additional feature that any available smoothness is automatically exploited to improve u^h even further.

It should also be pointed out that the τ-extrapolation can improve the solution in many cases where the Richardson extrapolation cannot. The τ-extrapolation depends on Taylor expansion for the *local* truncation error, while the Richardson extrapolation requires such an expansion for the *global* discretization error $U^h - U$. In many cases the later expansion does not exist; for example, no such expansion is possible when the local truncation error is not uniform. Another nice feature of the τ-extrapolation procedure is that it produces the improved solution at all points of the fine-grid, while Richardson extrapolation gives it only at points common to the fine and the coarse grid.

A remark on both types of extrapolations: When the extrapolated solution is considerably better than the fine-grid solution, its accuracy is actually comparable to the accuracy of a higher-order solution on the *coarse* grid. (Because the coarse-grid higher-order error term is not removed by the extrapolation.) That higher-order coarse-grid solution is

in principle less expensive in computer resources than the extrapolated solution, since it does not use the fine grid at all. A fully adaptive procedure (cf. Section 3.6) would probably prefer in such a situation to use the higher-order scheme on the coarser grid.

2.4. General Properties of Multi-Grid Solutions (Advertisement)

We have discussed at length the multi-grid solution of one particular problem. The algorithm, however, does not depend on any of the particular features of that problem.

The *shape of the domain* is immaterial. Only low-frequency error components are affected by it, and such components are always liquidated on the coarsest grids for negligible amount of work. The only effect of complicated boundaries is to complicate the difference equations at adjacent points and thus to make each relaxation sweep somewhat more expensive. In terms of work-units as defined above, the efficiency remains the same. Many experiments (reported in Brandt (1972), with many more examples in Shiftan (1972)) confirm this.

Variations in the coefficients, or nonlinearities, in the differential equations usually also affect only low-frequency components, and are therefore still treated at the same multi-grid efficiency. When these variations are wild, i.e., when the coefficients change significantly over the scale of the grid, attention should be given to the proper choice of residual-weighting ($\bar{I}_h^H$ in (2.9) or (2.11)), since the residual function r^h is considerably less smooth than the error v^h. (See discussions in Section A.4 of Brandt (1977a). More precise rules of residual-weighting are given in Lecture 17 of Brandt (1978a). The weights near boundaries

depend on the type of boundary conditions.) It is also important for such cases that the coefficients of the coarse-grid difference equations represent local averages of the fine-grid coefficients. This, in fact, is automatically obtained if (for variable-coefficients linear problems) the difference equations on each grid are derived by suitably averaging the differential equations, or if (for nonlinear problems, where the coefficients depend on the solution) the solution-weighting (I_h^H in (2.9) or (2.11)) is a full weighting operator.

Numerical experiments (Brandt (1978a) p. 17-7, and more details in Ophir (1978)) were conducted with difference equations of the form

$$L^h U^h_{\alpha,\beta} \equiv a^h_{\alpha\beta} \frac{U^h_{\alpha-1,\beta} - 2U^h_{\alpha,\beta} + U^h_{\alpha+1,\beta}}{h^2}$$

$$+ c^h_{\alpha\beta} \frac{U^h_{\alpha,\beta-1} - 2U^h_{\alpha,\beta} + U^h_{\alpha,\beta+1}}{h^2} = F^h_{\alpha,\beta} , \qquad (2.22)$$

where *the coefficients* a^h *and* c^h *vary wildly*; e.g., a^h and c^h are random; or $a^h_{\alpha\beta} = c^h_{\alpha\beta} = 1$ at even points ($\alpha + \beta$ even) but $a = .01$ and $c = 1$ otherwise; etc. When, and only when, the algorithm (with proper line relaxation) used full weighting in the transfer to the coarse grid of both the residuals and the coefficients, the solution was almost as efficient as in corresponding constant-coefficient cases.

Many experiments were also made for various nonlinear problems. In 1975-76 Multi-grid programs were developed for the steady-state small-disturbance *transonic flow* problems (see South and Brandt (1976), and Section 6.5 in Brandt (1977a)), in which the differential equations are mixed

elliptic-*hyperbolic*, and the solutions contain shock discontinuities. Although these programs are somewhat obsolete (with the present stage of multi-grid experience and knowledge they could be improved in various ways), they do clearly show that the typical multi-grid efficiency can be obtained for this type of problems.

Recently, multi-grid codes for *steady-state incompressible Navier-Stokes problems* have been developed (see Lecture 7 in Brandt (1978a), Brandt (1978c) and Dinar (1978)). The algorithm is similar to the one shown in Section 2.2 above, except for the more elaborated "distributed" relaxation scheme which is required here (a) because it is an elliptic *system*, not a scalar equation, and (b) because, for large Reynolds numbers R the system is *singularly perturbed*. Cavity and pipe problems were solved, with R ranging from 0 to large values (but below the values causing instabilities). For any such R, the process required 6 to 7 work-units, and produced solutions closer to the true differential solutions (in cases those were known) than the exact solutions of the finite-difference equations.

Successful multi-grid applications are also reported for the *minimal-surface equations* (D. J. Jones, Lecture 15 in Brandt (1978a)), for the *pressure iteration in Eulerian and Lagrangian hydrohynamics* (Brandt, Dendi and Ruppel (1978)), and for some simple problems in *finite-element formulations* (Nicolaides (1978) and Poling (1978)). Not listed here are some other multi-grid applications, which seem not to have realized the true multi-grid efficiency[6].

In some nonlinear problems *a continuation (or embedding) process* should be made, either because there are several solutions to the differential equations or because an initial approximation to the solution cannot otherwise be

obtained. In such cases 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 $\gamma_0 \le \gamma \le \gamma_*$, where $P(\gamma_0)$ is easily solvable (e.g., it is linear) and $P(\gamma_*)$ is the target (the given) problem. The continuation method 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 $P(\gamma - \delta\gamma)$ (or extrapolation from several previous steps) as an initial approximation in an iterative process for solving $P(\gamma)$. The main purpose is to ensure that the approximations we use are all "close enough" to the respective solution, so that some desirable properties are maintained and convergence is ensured. The process should of course be made carefully in case of *bifurcation*, when several solutions branch off at a certain value of γ. (See, e.g., Keller (1977).)

With the multi-grid solution method the continuation process may cost very little. First, because it can be made on coarser grids. Sometimes, however, too coarse grids do not retain enough of the solution features, and the continuation may not accomplish its purpose. This means that oscillations on the scale of a certain mesh-size h cannot be ignored. These oscillations, however, do not usually change much at each $\delta\gamma$ step. The trick then is to use form (2.16) of the coarse-grid equations. We can make several continuation steps on the coarse grid G^H only, *retaining the values of* τ_h^H *fixed*. By doing this we effectively freeze the high-frequency part of the solution, and retain its influence on the coarse-grid equations. Only once in several (sometimes many) steps we need to calculate on the finer grid too, in order to update the fine-to-coarse correction τ_h^H. This of course may be carried further: the G^h equations themselves

may include a $\tau_{h/2}^{h}$ correction term, and once in several visits to G^h we may go to calculate on $G^{h/2}$ in order to update that term; etc.

A similar technique can be used in *optimal control problems*. Here, some parameters controlling the partial-differential problem should iteratively be adjusted so as to achieve some optimal condition (minimal "cost"). In such problems we can again make most of the iterations on coarse grids, with frozen values of the fine-to-coarse corrections τ_h^H, making only infrequent visits to finer grids for updating τ_h^H. (Together with τ_h^H we should also freeze the fine-to-coarse correction of the cost functional.) If, for example, the control itself is a grid function (e.g., a term in the right-hand side of the equations), a change in the control value at a point will introduce only smooth changes in the solution in regions away from that point. It is therefore enough to keep refined only a small portion of the domain at a time, while at other regions the coarse level is used with frozen fine-grid corrections. This technique should combine with the usual multi-grid approach of obtaining the first approximation on each finer grid by interpolating from a solution to a similar problem on the next coarser grid.

Such techniques can also serve some *ill-posed problems*, where the solution should fit some data which are not the normal, well-posed boundary conditions. In such situations only smooth components of the solution can be meaningfully fitted to the data (or to smooth averages of data). The data-fitting can therefore be made on a coarse grid, G^H say, but the coarse-grid equations should have the fine-to-coarse correction (2.17), so that they represent a reasonable approximation to the differential equations.

Small-storage multi-grid algorithms are also based on the form (2.16) of the coarse-grid equations. Observe that, once τ_h^H is known, the fine grid is no longer needed. But τ_h^H depends mainly on high-frequency components of u^h, which can be computed by having in storage only a small segment of G^h. An algorithm based on this idea ("segmental refinement", Section 7.5 in Brandt (1977a)) requires in principle only some $15^d \log n$ storage locations, where d is the problem dimension and n is the number of points in the finest grid. No external memory is assumed.

Time-dependent problems, especially of parabolic type, usually require the use of implicit difference equations. The system of equations to be solved at each time step is similar to the steady-state equations, and can be solved usually by one multi-grid cycle, starting from the previous-time solution as the first approximation. Moreover, in some important cases (e.g., the heat equation) after a short time $t_0 = O(h^2)$ the high-frequency components of the solution practically reach their steady state, and further changes occur only in the smooth components. Hence, for many time-steps, the values of τ_h^H hardly change. Freezing τ_h^H for several time-steps we can then solve the coarse-grid equation (2.16) without using the fine grid at all. Only infrequently should we make a time-step with fine grid calculations, to update the high-frequencies of the solution and the values of τ_h^H. In this way *we retain fine-grid accuracy but each implicit time step of this kind costs on the average much less than an explicit time step on the finest grid*. This kind of techniques for parabolic time-dependent problems, are discussed in Lecture 9 of Brandt (1978a), and some preliminary experiments are reported by Dinar (1978).

Parallel or vector processing can be fully exploited by the multi-grid algorithm. The main processes, namely, interpolations and relaxation sweeps, are completely parallelizable, although it requires the use of a slightly different type of relaxation, with smoothing rates (per sweep) somewhat slower than the (sequential) Gauss-Seidel relaxation. (See Section 3.3 in Brandt (1977a).)

Implementation difficulties. A fair amount of knowledge is required in implementing multi-grid algorithms, including some general knowledge common to all multi-grid programs, plus particular expertise related to the specific type of problem at hand.

Generally, one has to be familiar with the basic rules of interpolation and residual-weighting, with the normal flow of multi-grid runs[7)], and, last but not least, with the *data structure* used in multi-grid programs. Without a suitable data structure the program will become complicated, with many unnecessary repetitions. It is advisable to follow the programming techniques exhibited in the simple Sample Program (Appendix B of Brandt (1977a)) and in the programs of MUGTAPE (1978). With this technique, each operation (such as relaxation, coarse-to-fine interpolation, fine-to-coarse residual weighting) is written once for all grids. Moreover, most operations can be written once for all programs; that is, the same code can be used by all the programs which use the same data structure. This includes the coarse-to-fine and fine-to-coarse transfer operations (e.g., interpolation), the operation of introducing the values of a given function into a given grid, operations of generating and manipulating grids (e.g., augmenting a grid, coarsening a grid, generating the interior part of a grid, or transposing a grid from row structure to column structure), displays of grids or

grid-functions, etc. Three different data structures are used in existing programs: One for rectangular grids, one for single-string grids (where the domain can be defined as $\{(x,y)|x_1 \le x \le x_2,\ f_1(x) \le y \le f_2(x)\}$), and one for general grids. The latter is called the QUAD structure, and is described in Brandt (1977b) and in Lectures 12 and 13 of Brandt (1978a). With these techniques, *the programming of a multi-grid solution for a new problem is essentially reduced to the programming of the relaxation routine*. (The residual-weighting routine should also be programmed anew for each problem, but its part that depends on the problem is a simple modification of a similar part in the relaxation routine.)

Particular expertise is required in designing the relaxation sweeps. For a uniformly elliptic scalar equations the simplest Gauss-Seidel is the best scheme (on sequential machines), but suitable modifications are required for degenerate-elliptic, non-elliptic, indefinite or singular-perturbation problems, as well as for cases of parallel or vector processing, and for problems involving more than one unknown function. Generally speaking, the particular knowledge required for designing relaxation is similar in each case to the specific knowledge required in discretizing the differential equations. Similar - but not identical. As in learning discretization methods, one should learn relaxation methods gradually, starting from simplest models, gaining some basic insights, and only then proceeding to complex real-world problems.

The design of relaxation is much facilitated by a *standard gauge we have for apriori measuring the relaxation efficiency*. The only role of relaxation in multi-grid programs is to smooth the errors. The efficiency of the entire algorithm

depends on (and can be approximately predicted by) the "smoothing factor" of the relaxation sweeps. Since smoothing is a local process, the smoothing factor can be calculated by a local mode analysis (cf. Section 6). For simple equations this can be done by hand (see examples in Sections 3.1 and 6.2 of Brandt (1977a), and Section 3 of Brandt (1976).). For general equations it is done by using the computer routine SMORATE, available on MUGTAPE (1978). The user of this routine supplies a description of the relaxation scheme and other parameters (in a format explained by comment cards in the routine). The output contains the smoothing factor and other useful information, including an estimate of the multi-grid convergence factor per work-unit. The routine can therefore be used to *optimize relaxation*, i.e., to select the best relaxation type and parameters from a given set of possibilities.

Some general orientation concerning the relaxation of singular-perturbation problems is given in Section 6 below.

A major advantage of the multi-grid solution process, in particular for singular-perturbation and other irregular problems, is its *full compatibility with adaptive processes*. The reason is that the multi-grid process in itself is adaptive: in adaptive processes mesh sizes are adapted to the computed solution; the multi-grid process goes one step further and employ mesh sizes adapted to the *error* in the computed solution. Let us now turn to these adaptive aspects of the multi-level techniques.

3. NON-UNIFORM AND ADAPTIVE DISCRETIZATIONS

3.1. Non-Uniformity Organized by Uniform Levels

In principle, the multi-grid solution process described in Section 2 could work equally well when the finest grid G^h is non-uniform and even non-rectangular, with grid points at arbitrary locations. A relaxation with good smoothing rates on a general grid is obtained by employing all line and marching directions. The main difficulty with general grids, however, is practical: Merely to formulate and use the difference equations, let alone solve them, is complicated. It requires lengthy calculations and large memories for storing geometrical information, such as the location of each grid point, its neighbors, the coefficients of its difference equation, etc. The multi-grid processing of such arbitrary grids generates additional practical difficulties since it requires the introduction of coarser grids and the grouping of grid points in grid lines (for line relaxation).

These arbitrary general grids, however, are not really needed. We will show below a method of organization which is less general but in which any desired refinement pattern can still be obtained, and easily changed, with negligible book-keeping and with difference equations always defined on equi-distant points. This flexible organization will naturally lend itself to multi-grid processing and to local transformations, and will lay the groundwork for efficient adaptation.

It is proposed to organize non-uniform grids as "*composite grids*". A composite grid is a union of uniform subgrids

$$\ldots, G^{4h}, G^{2h}, G^{h}, G^{h/2}, \ldots \tag{3.1}$$

where the superscripts denote the mesh-size. The grids are usually positioned so that every other grid-line of G^h is

a grid line of G^{2h}. Unlike the description in Section 2, however, the subgrids are not necessarily extended over the entire domain Ω: The domain of G^h may be only a proper part of the domain of G^{2h}, so that different degrees of refinement can be created at different subdomains. See Figure 2. Each G^h is extended, as a rule, over those subdomains where the desired mesh-size is roughly 1.5h or less. G^h may be thus disconnected, but its domain is always a subdomain of G^{2h}. The effective mesh-size at each neighborhood will be that of the finest grid covering that neighborhood. Clearly, any desired mesh-size $\bar{h}$ can be approximated by some effective mesh-size h', where $0.75\bar{h} \leq h' \leq 1.5\bar{h}$. Mesh-sizes never require better approximation.

The composite grid is very flexible, since local grid refinement (or coarsening) is done in terms of extending (or contracting) uniform subgrids, which is relatively easy and inexpensive to implement. A scheme for constructing, extending and contracting uniform grids, together with various service routines for such grids (efficient sweeping aids, interpolations, displays, etc.) is described in Brandt (1977b) and is partly available on MUGTAPE (1978a). One of its advantages is the efficient storage. The amount of logical information (pointers) for describing a uniform grid is proportional to the number of strings of points, and is therefore usually small compared with the number of points on the grid. Similarly, the amount of logical operations for sweeping over a grid is only proportional to the number of strings. Changing a grid is inexpensive, too.

Moreover, this composite structure will at the same time provide a very efficient solution process to its difference equations, by using its levels (3.1) also as the multi-grid sequence (as in Section 2). Each G^h will automatically

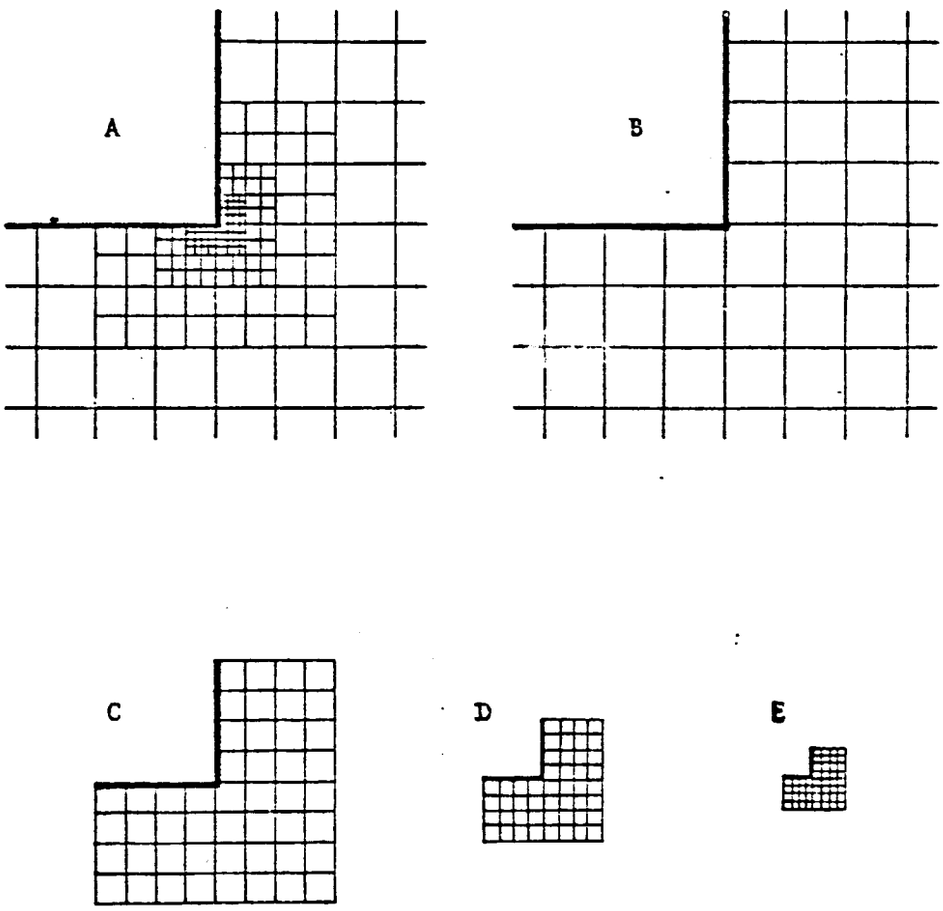


Fig. 2. A piece of non-uniform grid (A) and the uniform subgrids it is made of (B, C, D, E).

play the role of the correcting coarse-grid whenever the finer subgrid $G^{h/2}$ is present (see Section 3.3).

In adaptive procedures, the sequence of subgrids will be kept open-ended, so that we can add increasingly finer or coarser levels, as needed. (Increasingly coarser levels may be needed if the problem's domain Ω is unbounded and the bounded computational domain is chosen adaptively).

The coarsest subgrid should of course be kept coarse enough to have its system of difference equations relatively inexpensive to solve. Hence, there will usually be several coarse subgrids extending over the entire domain Ω. That is, they will not serve to produce different levels of refinement, but they are kept in the system for serving in its multi-grid processing.

There seems to be certain waste in the proposed system, because one function value may be stored several times when its geometrical point belongs to several subgrids G^k. This is not really the case: The extra values are exactly those needed for the multi-grid processing. In the process, the different subgrids may have different values at the same geometrical point. Moreover, it is only a small fraction (2^{-d}) of the points that are actually being repeated.

3.2. Unisotropic Refinements

For singular-perturbation and other problems, it is sometimes desired to have a grid which resolves a certain thin layer, such as a boundary layer. Very fine mesh-sizes are then needed in one direction, namely, across the layer, to resolve its thin width. Even when the required mesh-size is extremely small, not many grid points are needed, since the layer is correspondingly extremely thin. (See Section 4.)

Provided, of course, that the *fine mesh-size is used only in that one direction*. We need therefore a structure for mesh-sizes which get finer in one direction only.

In case the thin layer is along coordinate lines (x_j = constant), we resolve it again by using a sequence of uniform grids, except that their meshes are no longer square; h_i, the mesh-size in the x_i direction, may be very different from h_j. See Figure 3. We still require all mesh-sizes to be binary multiples of some basic size h_0, that is, on the k-th subgrid the mesh-size in the x_i direction is

$$h_i^k = 2^{n_{ik}} h_0; \qquad (n_{ik} \text{ integer}, \quad i = 1,\ldots,d) \; . \qquad (3.2)$$

For the multi-grid processing we require that for each such subgrid k, except for the very coarsest (k = 1), there exists in the scheme a coarser subgrid, number $\ell = \ell(k)$ say, such that for each $1 \leq i \leq d$ either $n_{i\ell} = n_{ik}$ or $n_{i\ell} = n_{ik} + 1$, and $\Sigma\, n_{i\ell} > \Sigma\, n_{ik}$. Grid ℓ will be the grid from which corrections are interpolated to grid k, and to which residuals from grid k are transferred. We call ℓ "the *predecessor* of k", and k "a *successor* of ℓ". Each subgrid, except for the coarsest, has exactly one subgrid defined as its predecessor, and may have any number of successors. The domain on which each subgrid is defined is always contained in, or coincide with, the domain of its predecessor. Thus, the set of subgrids is arranged logically in a tree, instead of the linear ordering we had before.

All these subgrids are still uniform, and can still easily be handled (created, extended, displayed, etc.) by the system mentioned above. Except that some of the interpolation routines required for this more general situation

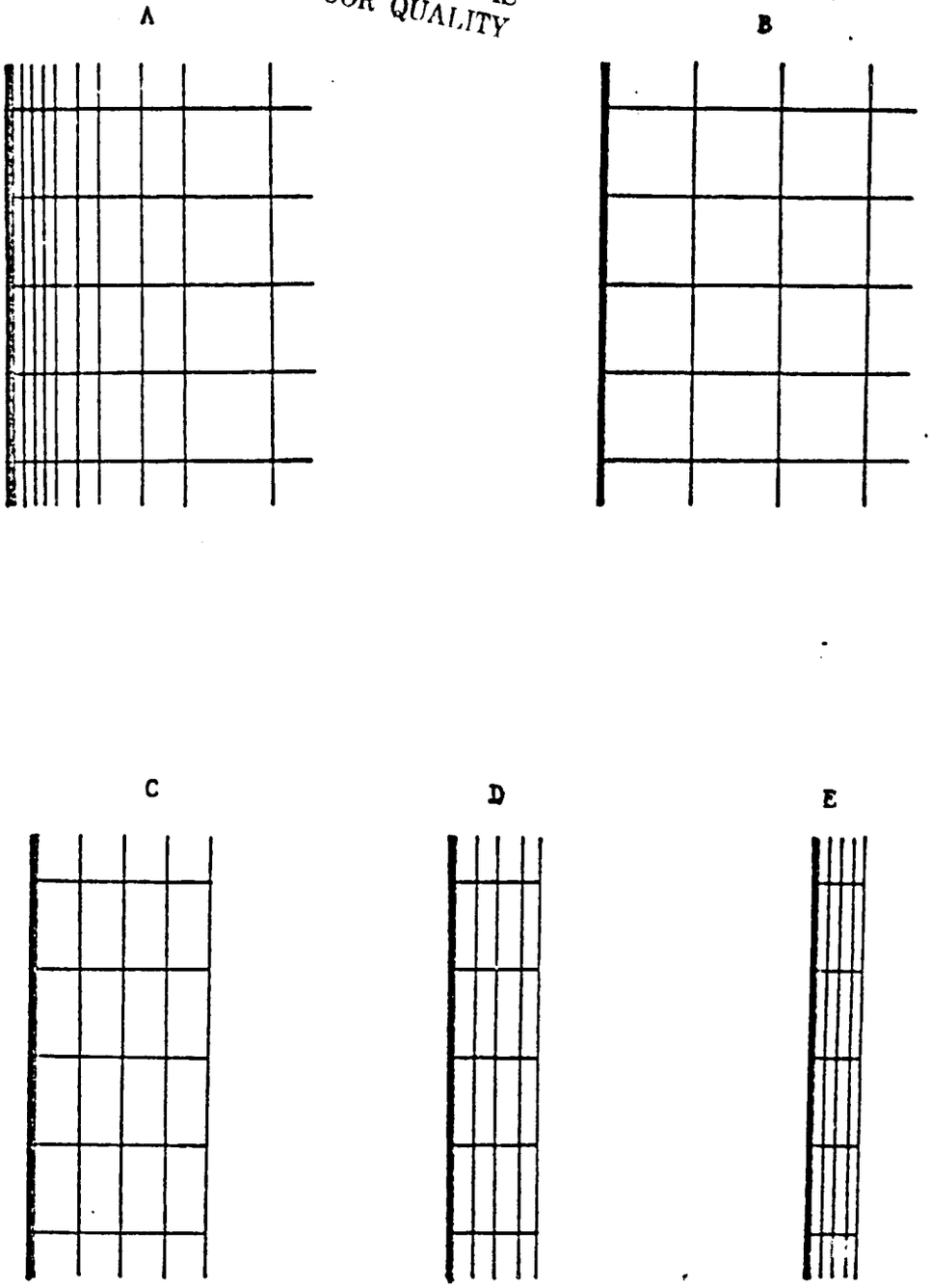


Fig. 3. A piece of non-uniform, boundary-layer type grid (A) and the uniform rectangular subgrids it is made of (B, C, D, E).

are still missing in MUGTAPE (1978a); they have been prepared for MUGTAPE (1978b).

In case the thin layer is not along coordinate lines, some local coordinate transformation is required. This technique is explained in Section 3.5 below.

3.3. Difference Equations and Multi-Grid Processing

The difference equations and their multi-grid solution for the composite structure are explained in Section 2.2 of Brandt (1977b). The main idea is that a fine-to-coarse correction function τ_h^H, correcting the G^H equations, (see (2.16)-(2.17) and the subsequent discussion) can be computed wherever the finer grid G^h exists, or, more precisely, at any interior point of G^H which is also an interior point of its successor G^h. At all other points the original coarse grid equation (2.3) can be used. From this point of view it is clear that we can have various patches of finer subgrids ("successors") thrown over various desired subdomains of G^H; the finer-subgrid accuracy will be established in the equations of such a subdomain via the τ_h^H correction. On any part of such a subdomain a patch of still-finer subgrids can be defined, etc.

The multi-grid process proceeds essentially as before. If, for example, we regard the coarsest subgrid as level 1, and all the successors of level j as level $j+1$, we could use exactly the same algorithm; e.g., the one shown in Table 1 above. Except that now, each operation (relaxation, fine-to-coarse residual transfer, etc.) at each level is performed not on one subgrid, but on the sequence of all subgrids of that level in some preassigned order. Important improvement: Relaxing each cycle progressively smaller <u>parts</u> of the coarser grids.

Note that successors of a given subgrid G^H may geometrically overlap. All is needed in such a case is to set priority relations between the successors, to tell which correction τ_h^H applies at those G^H points which are interior points of more than one successor. Such priority relations are automatically implied by the ordering of subgrids within each level.

An important advantage of this structure is its flexibility. One can add more and more patches of increasingly finer subgrids, where and when they are needed. One can also discard some such patches. Notice, however, that even when a piece of finer grid, G^h say, is discarded one can still retain its correction τ_h^H in the G^H equations. (This leads to multi-grid procedures which require only a small memory. See Section 2.4 in Brandt (1977b).)

Another important advantage of the outlined structure is that our *difference equations are defined on uniform grids only* (patched together by the usual multi-grid interpolations). Such difference equations on equidistant points are simple and can be read from small standard tables, while on general grids their weights would have to be recomputed (or stored) separately for each point, entailing very lengthy calculations especially for high-order approximations. Thus, our system *facilitates the use of high and variable (adaptive) order of approximation.*

Still another advantage is that relaxation sweeps, too, are done on uniform grids only. This simplifies the sweeping and is particularly essential where symmetric and/or alternating-direction line relaxations are required for obtaining high smoothing rates.

3.4. *Remark on High-Order Approximations*

An efficient and convenient way of using high-order difference equations, especially when the order is adaptible (see Section 3.6), is by the well-known technique (suggested by L. Fox) of "deferred correction" (see, e.g., Pereyra (1968), Lentini and Pereyra (1975) and Stetter (1978)). Simply, before starting a multi-grid cycle for improving an approximation u^h, add to the right-hand side of the fine-grid equations (2.2) the correction

$$\sigma_p^h(x^h) = L^h u^h(x^h) - L_p^h u^h(x^h), \qquad (x^h \in G^h), \qquad (3.3)$$

where L_p^h is the higher-order (order p) operator. Then proceed with the multi-grid cycle as usual. The roll of σ_p^h is similar to the roll of the fine-to-coarse correction τ_h^H. We can thus call σ_p^h the *high-order-to-low-order ("deferred", or "defect") correction.*

A certain amount of work is saved if p is advanced in steps; e.g. each multi-grid cycle advance p by 1 or 2. In the adaptive procedures described below, p is always advanced gradually.

Note that, since the multi-grid cycle operates with the original operator L^h, no new routines (such as relaxation routine) should be added, and the same multi-grid efficiency is obtained as in solving low-order equations. Except that the number of multi-grid cycles may increase linearly with p, since $\tau^{2h}/\tau^h \approx 2^p$.

3.5. *Local Coordinate Transformations*

Another dimension of flexibility and versatility can be added to the above system by allowing each subgrid to be defined in terms of its own set of coordinates.

Near a boundary or an interface, for example, the most effective local discretizations are made in terms of local coordinates in which the boundary (or interface) is a coordinate line. In such coordinates it is easy to formulate high-order approximations near the boundary; or to introduce mesh-sizes which are much smaller across than along the boundary layer (see Section 3.2); etc.

Usually it is easy to define suitable *local* coordinates (see below), and uniformly discretize them, but it is more difficult to patch together all these local transformations, especially in an adaptible way. In the above structure, however, this difficulty does not arise, since we can introduce independent and overlapping patches of "successor" grids.

Each set of coordinates will generally have more than one subgrid defined on it, so that (i) local refinement, in the style of Figure 2 and/or Figure 3 above, can be made within each set of coordinates; and (ii) the multi-grid processing retains its full efficiency by keeping the mesh-size ratio between any subgrid and its predecessor properly bounded (e.g., $\geq \frac{1}{2}$).

Since local refinement can be made within each set of coordinates, the *only* purpose of the coordinate transformation is to provide a certain grid direction, i.e., to have a given manifold (e.g., a piece of boundary) coincide with a grid hyperplane. We can therefore limit ourselves to simple forms of transformations. For example, in 2-dimensional problems, let a curve (a boundary, an interface, etc.) be given in the general parametric form

$$x = x_0(s), \quad y = y_0(s), \qquad (0 \leq s \leq s_1) \tag{3.4}$$

where s is the arclength, i.e.,

$$x_0'(s)^2 + y_0'(s)^2 = 1 \ . \qquad (3.5)$$

To get a coordinate system (r,s) in which such a curve will be a grid line, we can always use the transformation

$$x(r,x) = x_0(s) - ry_0'(s), \quad y(r,s) = y_0(s) + rx_0'(s) \ . \qquad (3.6)$$

Near the given curve $(r = 0)$ this transformation (a special case of transformations discussed in Starius (1977a)) is orthogonal, owing to (3.5), and transforms any small $h \times h$ square to another $h \times h$ square.

The main advantage of this transformation is that it is fully characterized by the single-variable functions $x_0(s)$, $y_0(s)$. These functions (together with $x_0'(s)$, $y_0'(s)$ and $q(s) = x_0''/y_0' = -y_0''/x_0'$) can be stored as one-dimensional arrays, in terms of which efficient interpolation routines from (x,y) grids to (r,s) grids, and vice versa, can be programmed once for all. (Such a general routine, however, is not easy to program, and is still missing in MUGTAPE (1978).) The difference equations in (r,s) coordinates are also simple enough in terms of these arrays. For example, by (3.5-6),

$$\frac{\partial}{\partial x} = -y_0' \frac{\partial}{\partial r} + \frac{x_0'}{1+rq} \frac{\partial}{\partial s}, \quad \frac{\partial}{\partial y} = x_0' \frac{\partial}{\partial r} + \frac{y_0'}{1+rq} \frac{\partial}{\partial s} \ . \qquad (3.7)$$

Hence we can easily approximate the (x,y) derivatives by (r,s) finite-differences, with numerical values of $x_0'(s)$, $y_0'(s)$ and $q(s)$ directly read from their stored tables. (No interpolation is needed if the tables contain values for s points which correspond to grid lines and half-way between grid lines.)

Such a system offers much flexibility. Precise treatment of boundaries and interfaces by the global coordinates is not required, since along boundaries the global grids are

only correction grids to the local ones. The local coordinates are easily changeable (changing only the one-dimensional tables of x_0, y_0, x_0', y_0', q) and can therefore be adapted to a moving interface.

The main difference between this structure and the one used by Starius (1977a), (1977b) and (1978) is that the boundary grids are completely embedded in the global grids (their predecessors), allowing a fast multi-grid solution of the equations. Also, since we have the multi-grid method for local refinement, the coordinate transformation is used only for orienting the grid, hence only the simpler transformation (3.6) is needed, allowing simpler differencing and interpolations.

Another variant of this procedure is required in case the location of the curve (interface, shock, etc.) is not fully defined. For example, a solution may include many shocks, some weaker and some stronger, and it is hopeless to try to recognize where a shock occurs, let along determine its exact curve. The usual procedure is to let the shocks develop by themselves, e.g., by adding some artificial viscosity which spread shocks over several mesh-sizes. Sometimes, however, this procedure is unacceptable because too much artificial viscosity is used near strong shock (and because of other reasons). We like to have a procedure which will automatically use smaller mesh-sizes near stronger shocks. This will be done by the general adaptive procedure (Section 3.6 below) if we choose the error functional E so that it contains some measure of the artificial viscosity. In order to obtain full efficiency, however, we like the procedure to be able to produce mesh-sizes which are much smaller in one direction (the direction perpendicular to the shock) than in the other. We therefore need a structure for

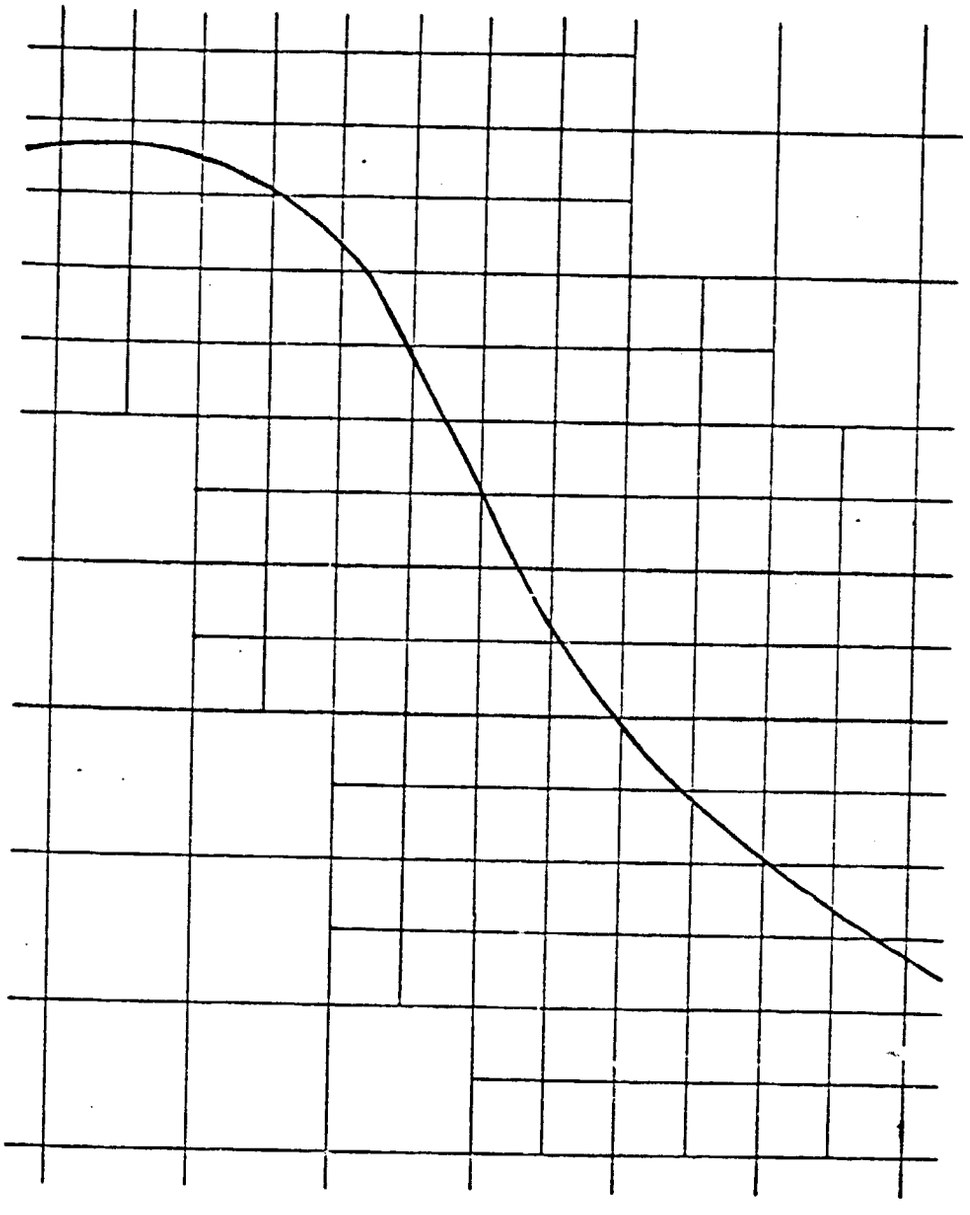
adapting the grid *orientation*, too. Notice that on some coarse grids the orientation is immaterial; the finer the grid the more precisely its orientation should be chosen. Hence, the direction can be refined successively, together with the mesh-sizes. An example is drawn and explained in Figure 4. We see that in this method the more general transformation (3.6) is not needed; only rotations are used. Hence the difference equations may be as simple as in the original (e.g., cartesian) coordinates. This method may therefore be preferable even in cases the curve (e.g., boundary) is known.

3.6. *Adaptation Techniques*

The flexible organization and solution process, described above, facilitate the implementation of variable mesh-size $h(x)$ and the employment of high and variable approximation order $p(x)$. 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 high-order approximation be used at such layers? How does one detect such layers automatically? In this section we survey (for more details, see Brandt (1977a), Chapters 8 and 9) a general framework for automatic selection of $h(x)$, $p(x)$ and other discretization parameters in a (nearly) optimal way. This system automatically resolves or avoids from resolving thin layers, depending on the goal of the computations, which can be stated through a simple function.

As our directive for sensible discretization we consider the problem of minimizing a certain error estimator E subject to a given amount of solution work W (or minimizing W for a given E. Actually, the control quantity will

Figure 4A



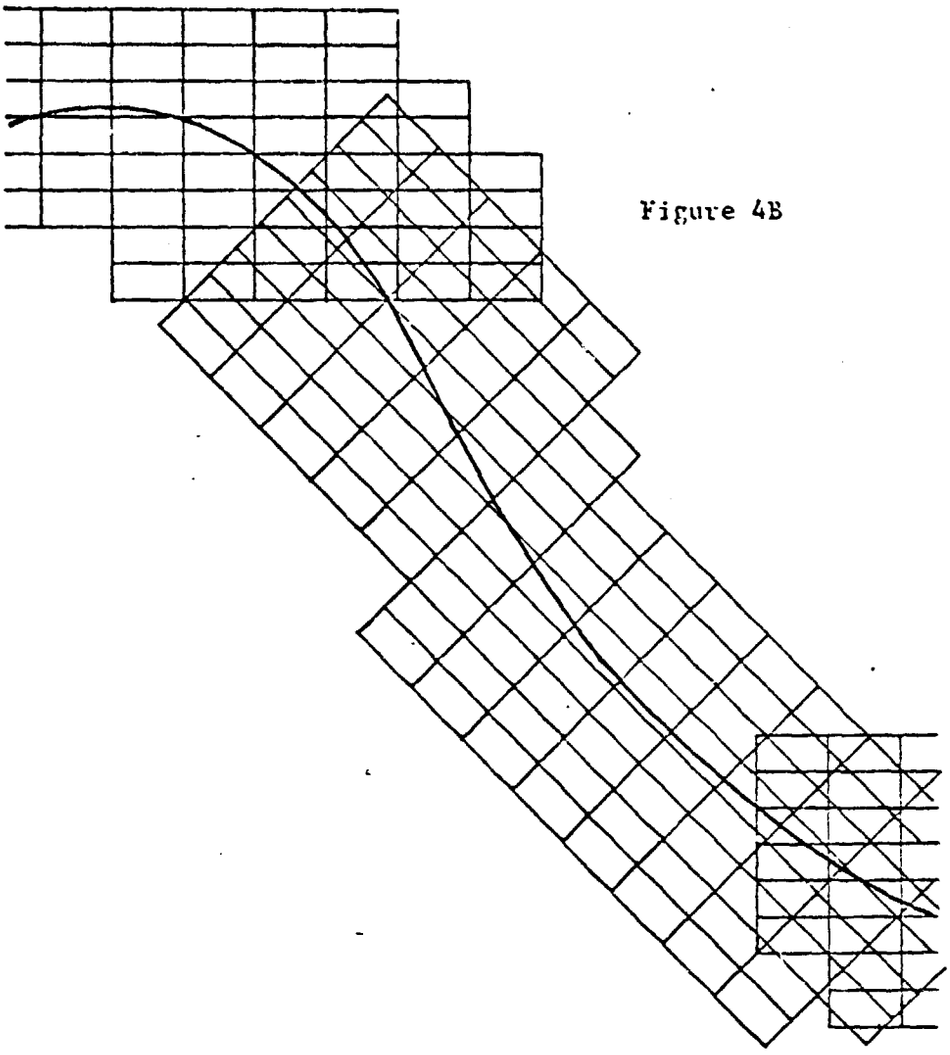


Figure 4B

Figs. 4A and 4B. Grid orientation around an interior thin layer. The two coarsest levels (A) have the usual orientation 0. The next level (B) has 3 orientations: 0, $\frac{\pi}{4}$ and $-\frac{\pi}{4}$ (the later is not applied here). The next level (not shown) would have 7 orientations: 0, $\pm\frac{\pi}{8}$, $\pm\frac{\pi}{4}$, $\pm\frac{3\pi}{4}$, etc. The successors (refinements) of a grid will always have either the same orientation or one of the two closest ones (e.g., each successor of the $\frac{\pi}{4}$-oriented grid in B will have orientation $\frac{\pi}{4}$, $\frac{\pi}{8}$ or $\frac{3\pi}{8}$).

be neither E nor W, but their rate of exchange). This optimization problem should of course be taken quite loosely, since full optimization would require too much control work and would thus defeat its own purpose.

The error estimator E has generally the form

$$E = \int_{\Omega} G(x)\tau(x)dx \ , \tag{3.8}$$

where $\tau(x) = \tau(x,h,p)$ is the magnitude of the local truncation error at x (see (2.18)). $G(x) \geq 0$ is the error-weighting function. It should in principle be imposed by the user, thus defining his goal in solving the problem. In practice G(x) serves as a convenient control. It is only the relative orders of magnitude of G(x) at different points x that really matter, and therefore it can be chosen by some simple rules. For example, if it is desired to compute ℓ-order derivatives of the solution up to the boundary then

$$G(x) \approx d_x^{m-1-\ell} \ , \tag{3.9}$$

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

The work functional W is roughly given by

$$W = \int_{\Omega} \frac{w(p(x))}{h(x)^d} dx \ , \tag{3.10}$$

where d is the dimension and h^{-d} is therefore the number of grid points per unit volume. $w = w(p)$ is the solution work per grid-point. In multi-grid processing, for $p \leq 6$ the work depends mainly on the number of cycles (cf. Section 3.4), hence $w \approx w_0 p$. For (unusually) large p, $w = O(p^3)$ since evaluating L_p^h at each cycle involves O(p) terms and O(p) arithmetic precision.

Treating $h(x)$ and $p(x)$ as continuous variables, the Euler equations of minimizing (3.8) and (3.10) can be written as

$$G\frac{\partial\tau}{\partial h} - \frac{\lambda dw(p)}{h^{d+1}} = 0 \ , \tag{3.11a}$$

$$G\frac{\partial\tau}{\partial p} + \frac{\lambda w'(p)}{h^{d}} = 0 \ , \tag{3.11b}$$

where λ is a constant (the Lagrange multiplier), representing the marginal rate of exchanging optimal accuracy for work: $\lambda = -dE/dW$. The $=$ sign in (3.11b) should be replaced by $\geq$ at points x where p attains its minimal allowable value p_0 (usually 1 or 2). In case we use fixed-order difference equations (adapting $h(x)$ only, p is fixed in advance), equation (3.11b) should be omitted. If constant order is used (p is constant over the domain, but instead of being fixed in advance this constant is to be optimized) equation (3.11b) is replaced by

$$\frac{\partial E}{\partial p} + \lambda\frac{\partial W}{\partial p} = 0 \ . \tag{3.12}$$

In principle, once λ is specified, equations (3.11) determine, for each $x \in \Omega$, the local optimal values of $h(x)$ and $p(x)$, provided the truncation function $\tau(x,h(x),p(x))$ is fully known. In some problems the general behavior of $\tau(x,h,p)$ near singularities or in singular layers is known in advance by some asymptotic analysis, so that approximate formulae for $h(x)$ and $p(x)$ can apriori be derived from (3.11). More generally, however, equations (3.11) are coupled with, and should therefore be solved together with, the given differential equations. Except that (3.11) is solved to a cruder approximation. This is done in the following way:

In the multi-grid solution process (possibly incorporating a continuation process), incidentially to the stage of computing f^{2h} from u^{2h}, u^h and f^h (see Figure 1 above, 2h corresponding to level k and h to k + 1) we can get an estimate of the decrease in the error estimator E introduced by the present discretization parameters. For example, the quantity

$$\begin{aligned} -\Delta E(x) &= G(x)\left|\tau(x,2h,p) - \tau(x,h,p)\right| \\ &\approx G(x)\left|\tau_h^{2h}\right| \\ &= G(x)\left|f^{2h}(x) - F^{2h}(x)\right| \end{aligned} \tag{3.13}$$

(cf. (2.17)-(2.19) above) may serve as a local estimate for the decrease in E per unit volume owing to the refinement from 2h to h (cf. (3.8)). Each such decrease in E is related to some additional work W (per unit volume). For example, that refinement from 2h to h required the additional work (per unit volume; cf. (3.10))

$$\Delta W = \frac{w(p)}{h^d} - \frac{w(p)}{(2h)^d} . \tag{3.14}$$

We say that the present parameter (h in the example) is highly profitable if the local rate of exchanging accuracy for work $Q = -\Delta E/\Delta W$ is much larger than the control parameter λ.

More sophisticated tests may be based on assuming τ to have some form of dependence on h and p. Instead of calculating Q for the previous change (from 2h to h in the example) we can then extrapolate and estimate the rate $\bar{Q}$ for the next change (from h to h/2), which is the more appropriate rate in testing whether to make that next change. The "extrapolated test" is not, however, much different in practice, and may actually be equivalent to testing the former Q against another constant λ.

In deciding whether and where to change the discretization, we adopt rules which stablize the adaptive process. For example, a change (e.g., refinement from h to $h/2$, which in practice (see Sec. 2.2) means an extension of the uniform grid with mesh-size $h/2$) is introduced only if there is a point where the change is "overdue" (e.g., a point where $\bar{Q} > 15\lambda$). But, together with each point where the change is introduced, it is also introduced at all neighboring points where the change is just "due" (e.g., where $\bar{Q} > 3\lambda$).

We can use the Q vs λ test to decide on all kinds of other possible changes, such as: Changing the order p to $p + 1$ (or $p - 1$); or changing the computational boundaries (when the physical domain is unbounded); or we can use such a test to decide whether to discard some terms from the difference operator (such as the highest order terms in some regions of a singular-perturbation problem); or to decide on unisotropic changes in h and p (e.g., changing Δx to $\Delta x/2$ without changing Δy); etc.

In case of optimizing a global quantity (such as constant approximation order; cf. (3.12)) similar tests can still be applied, except that ΔE and ΔW should of course be measured globally (summing over the entire region) instead of locally.

The computer work invested in the tests is negligible compared with the solution work itself. The measure (3.13) is taken only on $G^{2h} \cap G^{h}$, and the stage of computing it occur only once per several relaxation sweeps on G^{h}.

4. MULTI-LEVEL ADAPTIVE SOLUTIONS TO SINGULAR-PERTURBATION PROBLEMS: CASE STUDIES

To get a transparent view of the discretization patterns and the accuracy-work relations typical to the adaptive procedures proposed above when applied to singular perturbation problems, we consider now several cases which are simple enough to be fully analyzed. The simplicity of the solution, it should be emphasized, is not used in the solution process itself.

4.1. Optimal Discretization of One-Dimensional Case

Consider the 2-point boundary-value problem

$$\varepsilon \frac{d^2U}{dx^2} + \frac{dU}{dx} = 0 \quad \text{in} \quad 0 < x < 1 \ , \tag{4.1}$$

with constant $\varepsilon > 0$ and with boundary conditions $U(0)$ and $U(1)$ such that the solution is $U = e^{-x/\varepsilon}$. An elliptic (stable) difference approximation to such an equation can be central for $\varepsilon \geq 2h$ but should be properly directed for small ε. (See Section 5 below.) In either case, the truncation error behaves like

$$\tau(x,h,p) = \left(\frac{h}{\gamma\varepsilon}\right)^p \frac{e^{-x/\varepsilon}}{\varepsilon} \ , \tag{4.2}$$

where γ is a constant close to 2. (Actually, γ slightly depends on p; see (5.14). For simplicity, we neglect this dependence.) For the error weighting function we choose

$$G(x) \equiv 1 \ , \tag{4.3}$$

which would be the choice (see (3.9)) when one is interested in computing boundary first-order derivatives (corresponding, e.g., to boundary pressure or drag, in some physical models). Then, assuming $w(p) = w_0 p$, the optimization equation (3.11a) yields

$$\tau = \frac{\lambda w_0}{h} \quad , \tag{4.4}$$

and (3.11b) therefore becomes

$$h = \frac{\gamma}{e}\varepsilon \quad \text{if} \quad p > p_0; \quad h \geq \frac{\gamma}{e}\varepsilon \quad \text{if} \quad p = p_0 \ . \tag{4.5}$$

From (4.2), (4.4) and (4.5) it follows that

$$h = \frac{\gamma}{e}\varepsilon, \quad p = \log\frac{\gamma}{\lambda w_0} - 1 - \frac{x}{\varepsilon} \quad \text{for} \quad 0 < x \leq x_0 \ , \tag{4.6}$$

$$h = \frac{\gamma}{e}\varepsilon\, e^{(x-x_0)/(\varepsilon p_0+\varepsilon)} \ , \ p = p_0 \quad \text{for} \quad x_0 \leq x < 1 \ , \tag{4.7}$$

where

$$x_0 = \varepsilon(\log\frac{\gamma}{\lambda w_0} - p_0 - 1) \ . \tag{4.8}$$

If $x_0 \geq 1$ then (4.6) applies throughout, hence

$$W = \int_0^1 \frac{w_0 p}{h}\,dx = \frac{w_0 e}{\gamma\varepsilon}(\log\frac{\gamma}{\lambda w_0} - 1 - \frac{1}{2\varepsilon}) \ , \tag{4.9}$$

$$E = \int_0^1 \tau dx = \frac{\lambda w_0 e}{\gamma\varepsilon} = \frac{1}{\varepsilon}\exp(-\frac{\gamma\varepsilon}{w_0 e}W - \frac{1}{2\varepsilon}) \ , \tag{4.10}$$

and the condition $x_0 \geq 1$ itself becomes, by (4.8)-(4.9),

$$W \geq (p_0 + \frac{1}{2\varepsilon})\frac{w_0 e}{\gamma\varepsilon} \ . \tag{4.11}$$

Thus, if W satisfies (4.11), E converges like (4.10). That is, *for large values of* ε, *the total error* E *decreases exponentially as a function of the overall work* W.

Notice that when ε is large no boundary-layer is formed, and the mesh is uniform. Note also that the optimal mesh-size $h = \gamma\varepsilon/e$ is independent of the work W (or the exchange rate λ). That is, when more work can be afforded, it should *not* be invested in refining the grid, but in increasing the approximation-order p. In the MLAT processes described above p will automatically increase by 2 (or by 1, if non-central approximations are used)

every multi-grid cycle, until the desired accuracy (or the work limit, or the prescribed exchange rate λ) is reached.

4.2. Boundary-Layer Resolution

For small ε, however, the mesh-size $h \approx \varepsilon$ is impractical. Indeed, in the optimal discretization (4.6)-(4.7), for small ε we get small x_0, and an "external region" $x_0 \le x < 1$ is formed where the mesh-size grows exponentially. *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 $x \gg \varepsilon$) is practically constant, so that indefinitely large h is suitable. Usually h will grow exponentially, as in (4.7), from $h = \gamma\varepsilon/e$ to some definite value suitable for the external region (the optimal mesh-size of the reduced problem). In the transition region we have $p = p_0$, i.e., the minimal order of differencing is used in the region where h changes. This may be useful in practical implementations.

From (4.6)-(4.8) and (4.2)-(4.4) we get, for small ε,

$$W = \int_0^1 \frac{w_0 p}{h} = \frac{w_0 e}{2\gamma}\left[\left(\log\frac{\gamma}{\lambda w_0} - 1\right)^2 + 2p_0 + p_0^2\right] , \qquad (4.12)$$

$$E = \int_0^1 \tau dx = \frac{\lambda w_0 e}{\gamma} \log\frac{\gamma}{\lambda w_0} , \qquad (4.13)$$

where $\exp(-1/\varepsilon)$ and similar terms are neglected. Using (4.12) we can express λ in terms of W and substitute that expression is (4.13). In reasonable calculations $W \gg w_0$, and then the relation simplifies to

$$E \approx \left(\frac{2\gamma e}{w_0} W\right)^{1/2} \exp\left[-\left(\frac{2\gamma}{w_0 e} W\right)^{1/2}\right] . \qquad (4.14)$$

Thus, essentially -

For small values of ε, *the total error* E *decreases exponentially as a function of* $W^{1/2}$, *where this rate is independent of* ε *and does not deteriorate as* $\varepsilon \to 0$. In principle, this rate is better than $O(h^p)$, for any fixed p.

Notice that here h does depend on W (or λ), but only in some transitional layer. In the inner part of the boundary layer $h = \gamma\varepsilon/e$ still holds, while away from that layer h tends to the optimal mesh-size of the reduced problem. (If the reduced problem is itself regular, its optimal mesh-size will be determined by the "local scale" of that problem. This scale is independent of W, as it is for example in Section 4.1 above for the case of moderate ε. That scale is too small to resolve only when the reduced problem is singular). *What depends on the total computational work is the distance* x_0 *from the wall at which the meshsize starts to grow exponentially.* In fact, from (4.8) and (4.12) we see that

$$x_0 \approx \varepsilon\left(\frac{2\gamma}{w_0 e} W\right)^{1/2} . \tag{4.15}$$

Defining *the computational boundary layer* as the region where $h < h_0$ for some h_0 independent of ε, the width w_{CBL} of the layer is, by (4.7),

$$w_{CBL} \approx x_0 + (p_0 + 1)\varepsilon \log \frac{1}{\varepsilon} . \tag{4.16}$$

Another, quite obvious but interesting observation can be made at this junction, based on (4.11) above. Even for small ε, if W is sufficiently large then the exponential relation (4.10) holds. Hence, in an asymptotic theory for $W \to \infty$ (corresponding to asymptotic theory for $h \to 0$, which is so common in numerical analysis) the relations of this section, which undoubtedly dominate the numerical

process at reasonable values of W, would not be seen. What we should be interested in are *values of* W *which are large but independent of* ε, and therefore not large compared with negative powers of ε.

4.3. Fixed-Order and Constant-Order Discretizations

The optimal approximation-order p calculated above varies with the location x. This is not essential. Indeed, if p is fixed then (3.11b) is omitted, but (3.11a) and (4.2)-(4.3) still imply $\tau = \lambda w(p)/(hp)$, and hence

$$h = \gamma\left(\frac{\lambda w}{\gamma p}\right)^{\frac{1}{p+1}} \varepsilon e^{x/(\varepsilon p+\varepsilon)} \quad . \tag{4.17}$$

Hence, for small ε,

$$W = w\int_0^1 \frac{dx}{h} = \frac{w(p+1)}{\gamma}\left(\frac{\gamma p}{\lambda w}\right)^{\frac{1}{p+1}} , \tag{4.18}$$

$$E = \frac{\gamma w}{p}\int_0^1 \frac{dx}{h} = \frac{\lambda}{p} W = (p+1)\left(\frac{\gamma W}{(p+1)w}\right)^{-p} . \tag{4.19}$$

Thus, $E = CW^{-p}$ with C independent of ε, so that the convergence order is p (analogous to error $E = O(h^p)$ when h is constant). *The variable mesh-size* (4.17) *keeps the convergence rate essentially unimpaired by the singular perturbation, even though convergence is considered of the first derivative up to the boundary.*

The constant approximation-order p need not of course be fixed in advance. It may be optimized just as well, using global tests as mentioned above. From (4.19) it follows that the minimal E for a given W is obtained when p satisfies

$$1 + \frac{pw'(p)}{w(p)} = \log \frac{\gamma W}{(p+1)w(p)} \quad , \tag{4.20}$$

and for $w = w_0 p^{\ell}$ the total error will be

$$E \approx (\gamma W)^{1/(\ell+1)} \exp\left[- \frac{1+\ell}{e} \left(\frac{\gamma W}{w_0}\right)^{1/(\ell+1)} \right] . \tag{4.21}$$

For $\ell = 1$ this rate is almost the same as (4.14). Thus, *we do not lose much by using constant, optimized* p, which, on the other hand, may be considerably simpler to program.

From (4.17)-(4.18) and (4.20) we see that the width of the computational boundary layer is now

$$w_{CBL} \approx \varepsilon(p+1) \log \frac{1}{\varepsilon} \quad . \tag{4.22}$$

For small ε this is $(p+1)/(p_0+1)$ times wider than the variable-order case (4.16).

4.4. A Case of Skipping the Boundary-Layer

To see the effect of choosing different error-weighting functions, consider again problem (4.1), but with the choice

$$G(x) \equiv x \ . \tag{4.23}$$

This will be the choice *in case one is interested in approximating* U *only, not its derivative, and to approximate it in the* L_1 *sense*. By substituting (4.2) and (4.23) into (3.11), and solving for p, we would obtain

$$p = \log \frac{x\gamma}{\lambda w_0} - 1 - \frac{x}{\varepsilon} \tag{4.24}$$

$$\leq \log \frac{\varepsilon\gamma}{\lambda w_0} - 2 \ .$$

For bounded (independent of ε) λ and sufficiently small ε, this p is smaller than p_0. Hence $p = p_0$ should replace (3.11b) and substituting (4.2) into (3.11a) we actually get

$$\left(\frac{h}{\gamma\varepsilon}\right)^{p_0+1} = \frac{\lambda w_0 e^{x/\varepsilon}}{x\gamma} \geq \frac{\lambda w_0 e}{\varepsilon\gamma} . \qquad (4.25)$$

Thus, *for bounded* λ *and sufficiently small* ε, $h \gg \varepsilon$ *everywhere, so that the boundary-layer is not resolved by the grid.*

In fact, since $h \gg \varepsilon$, all interior grid points lie in a region where the rate of convergence would normally be determined by the reduced equation (see Section 4.5). In our simple example (4.1), the reduced equation has the trivial solution $U \equiv 0$, and accordingly $E \to 0$ as $\varepsilon \to 0$, for any fixed W (or λ). This can be verified from (3.8), (4.2) and (4.25).

In case one is interested in *approximating* U *in the* L_∞ *sense*, a precise choice of the error-weighting function is

$$G(x) = 1 - e^{-x/\varepsilon} . \qquad (4.26)$$

With this function, solving (3.11) for p we would get

$$p(x) = \log \frac{\gamma(1 - e^{-x/\varepsilon})}{\lambda w_0} - 1 - \frac{x}{\varepsilon} , \qquad (4.27)$$

$$\max p(x) = p(\varepsilon \log 2) = \log \frac{\gamma}{4\lambda w_0} - 1 ,$$

and hence, for λ reasonably small, $p(\varepsilon \log 2) > p_0$. Therefore, around $x = \varepsilon \log 2$, we again have $h = \frac{\gamma}{e}\varepsilon$. Beyond this point (for $x > \varepsilon \log 2$) the discretization pattern is essentially the same as in Section 4.2 above (since $G(x)$ is essentially the same). Before this point ($x \leq \varepsilon \log 2$) we have $h(x) > x$, so that in practice we do not have there more grid points. Thus, the grid throughout is essentially as in Section 4.2. Similarly, for

constant p the mesh-size distribution will be as in Section 4.3. The accuracy-work relation, too, is essentially as before.

4.5. *Remarks on More General Problems*

For general problems it is of course impossible to find *apriori* the relation between work and accuracy that would result from multi-level adaptive solution processes. In fact, in most non-trivial cases, an optimal (or nearly optimal) choice of discretization parameters (h(x), p, etc.) is not known in advance, since it depends on the particular solution. This is exactly why adaptive techniques are needed. Nevertheless, in this section we will try to indicate that the simple relations described in Section 4.2-4.4 are typical to many, perhaps most, singular-perturbation problems, even in complicated, high-dimensional problems.

Consider first a more general one-dimensional, constant-coefficient equation of the form

$$\varepsilon^{m-n}U^{(m)} + a_{m-1}\varepsilon^{m-n-1}U^{(m-1)} + \cdots + a_n U^{(n)} + a_{n-1}U^{(n-1)} + \cdots + U^{(0)} = 0 , \tag{4.28}$$

normalized so that

$$U(x) = e^{-x/\varepsilon} \tag{4.29}$$

is a solution. And assume the boundary conditions are such that (4.29) is actually *the* solution. The truncation error is then approximately (see (5.14))

$$\tau(x,h,p) = \left(\frac{h}{\gamma\varepsilon}\right)^p \frac{e^{-x/\varepsilon}}{\varepsilon^n} , \tag{4.30}$$

where γ is again a constant (slightly different than in (4.2), but still $\gamma \to 2$ for larger p. We again neglect the changes in γ).

If we are interested in computing $U^{(j)}(x)$ near $x = 0$, the error-weighting function for small x should behave like

$$G(x) \approx \varepsilon^{j-m+n} x^{m-1-j} . \tag{4.31}$$

For the adaptive process, the multiplicative constant in G is immaterial. For our convergence estimates, however, the correct order of ε should be used. The behavior (4.31) results from the observation that if $\varepsilon^{m-n} V^{(m)}(x) = \delta_\xi(x)$ in $(0,1)$ and $V(0) = V(1) = 0$, then, for $0 < x < \xi << 1$, $u^{(j)}(x) = O(\varepsilon^{-m+n} \xi^{m-1-j})$. An additional ε^j factor appears in (4.31) since we are interested in measuring *relative* errors in $u^{(j)}(x)$, and by (4.29), $u^{(j)}(x) = O(\varepsilon^{-j})$ for $x \leq O(\varepsilon)$.

For $j = m - 1$, G_τ is the same as in the special case discussed in Sections 4.2 and 4.3 above. Therefore exactly the same discretization parameters and the same accuracy-work relations will follow. For smaller j, the accuracy-work relation cannot get worse; it may even improve, depending on the norm used (cf. Section 4.4).

In more general singular-perturbation problems, the solution $U(x)$ can be written as a sum of a function $\tilde{U}(x)$ which tends uniformly to the solution U_0 of the reduced problem, and boundary-layer terms, each of which behaves like (4.29) above for some suitable ε. (See O'Malley (1974).) Consider the case of a fixed order p, as in Section 4.3 above. Let $h_0(x)$ be the mesh-size distribution optimal (at some given λ) for the reduced problem, and $h_i(x)$ the optimal distribution in case the solution

contain only the i-th boundary term $(i = 1,2,\ldots,\sigma)$. Let E_i and W_i denote the corresponding total error and overall work $(i = 0,1,2,\ldots,\sigma)$. For any given solution (containing all terms) choose

$$h(x) = \min_{0\le i\le\sigma} h_i(x) \ . \tag{4.32}$$

Then clearly $E \le \Sigma E_i$ and $W \le \Sigma W_i$. In an optimal choice of h, E will be even smaller (for the same value of W; or vice versa). Hence, essentially,

The convergence rate behaves either like one of those described above for the boundary terms (Section 4.3), *or like the convergence rate of the reduced problem, whichever is slower.*

The situation is a little more complicated for variable p, but we saw before that we don't loose much by using a constant p. Moreover, the optimal p (4.20) does not depend on ε and can therefore serve uniformly for all the boundary-layer terms.

Similar convergence rates should be expected in *higher-dimensional problems*, too. To see this, examine the behavior near some portion of the boundary. Assuming our computations use boundary coordinates (see Remark below), we can regard the boundary as $x_1 = 0$. Using Fourier transformation in all but the x_1 coordinate, the solution u can again be written, in many cases, as a sum of $\tilde{u}$ (tending asymptotically to the reduced solution U_0) and boundary-layer terms behaving like (4.29). Assuming also that the only significant adaptation of mesh-size is needed in the x direction (i.e., perpendicular to the boundary), we may repeat the above argument, using (4.32), and arrive at the same conclusion.

Remark about boundary coordinates: "Boundary Coordinates" is a coordinate system in which the boundary is contained, at least locally, in a coordinate hyperplane (e.g., $\{x_1 = 0\}$). In Section 3.5 above it is explained how to construct and use such a coordinate system in multi-grid processes. For the finite-difference equations it is important to use a grid along such boundary coordinates. Otherwise it is impossible to simultaneously use small mesh-sizes in the direction perpendicular to the boundary and large ones in the other direction(s), as required for obtaining efficiencies similar to the one-dimensional ones.

Thin transition layers not on the boundary, such as turning points in ordinary differential equations or contact-discontinuities and shocks in higher dimensions, are likely to be treated by multi-level adaptive techniques as efficiently as the boundary-layer cases analyzed above, since the procedures did not assume any apriori knowledge concerning the location of the layer. The layer is discovered, and if necessary resolved, by the numerical process, using general and automatic criteria. The only difficulty is, in higher dimensional problems, to get a coordinate system in which the internal layer is a coordinate hyperplane. To a suitable approximation, however, this can be done, using the procedure described in Figure 4 above.

Not all singular-perturbation problems can efficiently be solved by the above techniques, of course. For example: problems with highly oscillatory solutions, such as the Helmholtz equation

$$\varepsilon^2 \Delta u + u = 0 \ . \qquad (4.33)$$

In usual norms, this problem is *not uniformly well-posed*.

That is, the change in the solution caused by a certain change in the data is not uniformly bounded: it may increase indefinitely as $\varepsilon \to 0$. Such problems should be reformulated, using other variables and norms, so as to make them uniformly well-posed. (See Section 5 in Brandt (1978b).)

5. UNIFORMLY WELL-POSED HIGH-ORDER DIFFERENCE EQUATIONS

An extended version of this section appears as Section 5 in Brandt (1978b). It discusses the concepts of well-posedness and uniform well-posedness, ellipticity and uniform ellipticity, and their significance for singular-perturbation problems in general, and for their multi-level solutions in particular. Closely related are the extensive theoretical investigations of Frank (1978 and references therein). Related preliminary observations were made in Brandt (1976).

Here we summarize some of the more practical aspects.

5.1. General Remarks

In approximating potentially singular-perturbation equations it is essential to ensure that the discrete problem is uniformly well-posed (uniformly stable) not only with respect to the mesh-size (h), but also with respect to the singular-perturbation size (ε). That is, in suitable norms, a small change in data should cause a small change in the solution, *uniformly in both* h *and* ε. For this to be possible, the original differential problem should be uniformly stable (in ε). This, however, is not sufficient. Innocent-looking difference approximations L^h may easily be uniformly stable in h (i.e., for any fixed ε), but not jointly in h and ε. In such cases satisfactory approximations will still be obtained by sufficiently small

h, but that h will have to be small *compared with* ε (or some power of ε), and hence too small to be practical. In particular such mesh-sizes are unacceptable (even for moderate ε) for the coarse grids of a multi-level sturcture.

There are no general procedures to construct uniformly stable difference approximation; nor even general procedures to check uniform stability of given difference schemes. This is in fact already true for the differential equations. But there are some important classes of uniformly stable operators and some practical ways of constructions.

For various boundary-value problems to be well posed it is required that the partial-differential operator (2.1a) is *elliptic*, i.e., that the homogeneous system of equations has no non-constant periodic solution. This, together with appropriate boundary conditions, ensures well-posedness. Similarly, the difference operator (2.2) can be defined as elliptic if there is no periodic U^h such that $L^h U^h = 0$. For scalar operators ($q = 1$) such a definition was introduced by Thomée (1964), and various results related to the stability of such operators were proved by him and by Thomée and Westergren (1968). Many more results were published in conjunction with finite-element formulations, which yield scalar or vectorial elliptic difference operators. (See Ciartet (1978).). Most of these results, however, hold only for sufficiently small mesh-sizes, and are therefore not directly applicable in the present context (where "sufficient-small" means smaller than ε). Slightly different notions of ellipticity are needed. The most useful for applications is, perhaps, the following.

R-Ellipticity. Assume the $q \times q$ difference operator L^h in (2.2) has constant coefficients, and let

$h = (h_1,\ldots,d_d)$, where h_j is the mesh-size in the x_j coordinate. Denote

$$\theta = (\theta_1,\ldots,\theta_d), \quad \theta\cdot x/h = \theta_1 \frac{x_1}{h_1} + \cdots + \theta_d \frac{x_d}{h_d}, \tag{5.1}$$

$$|\theta| = \max(|\theta_1|,\ldots,|\theta_d|) \ . \tag{5.2}$$

Then, for any constant q-vector V

$$L^h e^{i\theta\cdot x/h} V = B(\theta,h) e^{i\theta\cdot x/h} V \ , \tag{5.3}$$

where A is a $q \times q$ matrix, easily obtained by replacing in the matrix L^h each h_j-translation with the complex function $\exp(i\theta_j)$. B is called the *matrix-symbol* of L^h. The difference operator L^h is called R-elliptic if

$$\mathrm{Re}\ V^T B(\theta,h) V > 0 \quad \text{for all} \quad 0 < |\theta| \leq \pi \quad \text{and all} \tag{5.4}$$

$$\text{real q-vectors} \quad V \neq 0$$

An operator L^h with variable coefficients is called R-elliptic if the frozen-coefficients operator at every point is R-elliptic. A nonlinear difference operator is called R-elliptic if the corresponding linearized operator is R-elliptic (which may depend on the solution around which the linearization is taken).

This is not a complete definition of ellipticity. For example, if L_1^h and L_2^h are R-elliptic, then $L_1^h L_2^h$ is not necessarily also R-elliptic. But the definition gives, on one hand, a concept much more general than the special case of positive-type operators (which trivially satisfy (5.4)); in fact, a definition general enough for almost all scalar equations. On the other hand, the definition has some nice properties. One property is that it restricts the location of the operator, while Thomée's definition allows any translation to be added to the operator (which of course

cannot be permitted in discussing finite mesh-sizes, because for example, it allows the two difference equations at two neighboring points to coincide, i.e., to be just one equation). This restriction is essential in discussing relaxation schemes, where a relation is required between each difference equation and the point at which it is relaxed. Another nice property is that the sum of R-elliptic operators is clearly also R-elliptic. One can therefore construct R-elliptic operators one term at a time.

We can use this property for singular-perturbation operators. If both the perturbed and the reduced equation are elliptic, the required uniform stability is obtained by constructing a difference approximation which is *uniformly* elliptic. A simple way to achieve this is to construct R-elliptic approximations to the various terms in the equation, so that R-elliptic approximation is obtained, in particular, for the reduced equation.

5.2. Examples

R-elliptic approximations, of arbitrary order, will be constructed in this section for the basic one-dimensional operators. Since this construction do not use any relation between terms, these approximations can be used as building blocks for approximating many ordinary and partial differential operators. The approximations are constructed on uniform grids only. As shown in Section 3, this is all we need in a multi-grid environment.

Using the operator notation

$$\delta u(x) = u(x+\tfrac{h}{2}) - u(x-\tfrac{h}{2}), \quad \nabla u(x) = u(x) - u(x-h),$$

$$\Delta u(x) = u(x+h) - u(x),$$

$$\mu u(x) = \tfrac{1}{2}\,[u(x+\tfrac{h}{2}) + u(x-\tfrac{h}{2})], \tag{5.5}$$

$$Du = \frac{du}{dx},$$

and the calculus of such operators (see, e.g., Dahlquist and Björk (1974) p. 311) one can derive the expansions

$$hD = \mu\delta \sum_{q=0}^{\infty} a_q(-\delta^2)^q \tag{5.6}$$

$$(hD)^2 = \delta^2 \sum_{q=0}^{\infty} \frac{a_q}{q+1}(-\delta^2)^q \tag{5.7}$$

where

$$a_0 = 1, \quad a_q = \frac{a_{q-1}}{4 + \frac{2}{q}} \quad \text{and hence} \quad (a_q)^{1/q} \to \tfrac{1}{4}. \tag{5.8}$$

From this we find the following expressions for the 2s-order central approximations to the first and the second derivatives, and for the corresponding local truncation errors:

$$u'(x) = \frac{1}{h}\mu\delta \sum_{q=0}^{s-1} a_q(-\delta^2)^q\, u(x) \tag{5.9}$$

$$+ (-1)^s a_s\, h^{2s}\, u^{(2s+1)}(\xi_1),$$

$$-u''(x) = \frac{1}{h^2}(-\delta^2) \sum_{q=0}^{s-1} \frac{a_q}{q+1}(-\delta^2)^q\, u(x) \tag{5.10}$$

$$-(-1)^s \frac{a_s}{s+1}\, h^{2s}\, u^{(2s+2)}(\xi_2),$$

where ξ_1 and ξ_2 are some intermediate points.

Let us now check these difference approximations for R-ellipticity. It is easy to see that the symbols corresponding to $\mu\delta$ and to $-\delta^2$ are, respectively, $i\sin\theta$ and $2(1 - \cos\theta)$. The latter is positive for all $0 < |\theta| \leq \pi$. Hence all the above approximations to $-u''$ (note the sign!) are R-elliptic. On the other hand any central approximation to either u' or $-u'$ has purely imaginary symbol, and is therefore never R-elliptic.

In various elliptic equations, $-u''$ is added to au', where a may have any sign. We therefore need to construct R-elliptic approximations to both u' and $-u'$. This is done by adding to (5.9) an R-elliptic term of order high enough: To obtain an approximation $O(h^{2s-1})$, add any positive multiple of the term $\frac{1}{h}(-\delta^2)^s$; to retain the $O(h^{2s})$ order, add any positive multiple of $\frac{1}{h}\nabla(-\delta^2)^s$ or $-\frac{1}{h}\Delta(-\delta^2)^s$. These terms are R-elliptic since the symbol of ∇ and $-\Delta$ are $1 - e^{-i\theta}$ and $1 + e^{i\theta}$, respectively, so that their real part is positive for $0 < |\theta| \leq \pi$. The values of the positive multiples can be chosen so that the $O(h^{\ell})$ approximation uses exactly $\ell + 1$ points ($\ell = 2s - 1,s$). This gives the following R-elliptic approximations and truncation errors:

$$\pm u'(x) = \left\{\pm \frac{1}{h}\mu\delta \sum_{q=0}^{s-1} a_q(-\delta^2)^q + \frac{a_{s-1}}{2h}(-\delta^2)^s\right\}u(x) \qquad (5.11)$$

$$+ (-1)^s \frac{a_{s-1}}{2} h^{2s-1} u^{(2s)}(\xi_3)$$

$$u'(x) = \left\{\frac{1}{h}\mu\delta \sum_{q=0}^{s-1} a_q(-\delta^2)^q + \frac{a_{s-1}}{2h}\nabla(-\delta^2)^s\right\}u(x) \qquad (5.12)$$

$$+ (-1)^s \left\{\frac{a_{s-1}}{2} + a_s\right\} h^{2s} u^{(2s+1)}(\xi_4) ,$$

$$-u'(x) = \left\{ -\frac{1}{h}\mu\delta \sum_{q=0}^{s-1} a_q(-\delta^2)^q - \frac{a_{s-1}}{2h}\Delta(-\delta^2)^s \right\} u(x) \quad (5.13)$$

$$- (-1)^s \left\{ \frac{a_{s-1}}{2} + a_s \right\} h^{2s}\, u^{(2s+1)}(\xi_5) \ .$$

Observe that these operators are completely one-sided (so called "upwind") only for the $O(h)$ and $O(h^2)$ approximations. One can describe the above formulae as the non-central operators closest to the central among all operators which use the minimal number of points. The one-sided operators using the same number of points are not R-elliptic (for orders higher than 2).

The error term. For theoretical purposes (as in Section 4.1 above) it is more convenient to express the magnitude of the error terms in (5.10) and (5.13) in the forms

$$\left(\frac{h}{\gamma_s^2}\right)^{2s} u^{(2s+2)}(\xi_2) \quad \text{and} \quad \left(\frac{h}{\gamma_s^1}\right)^{2s} u^{(2s+1)}(\xi_5) \ , \qquad (5.14)$$

respectively, and similarly for (5.11) and (5.12). It is clear from (5.8) that both $\gamma_s^2 \to 2$ and $\gamma_{s_i}^1 \to 2$ as s grows. In fact, as shown in Table 2, each γ_s^i does not change much with s, and for theoretical convenience we treat them as constants.

TABLE 2

s	1	2	3	4	5
a_s^{-1}	6	30	140	630	2772
γ_s^1	1.22	1.71	1.86	1.93	1.97
γ_s^2	3.46	3.08	2.87	2.73	2.64

High-order approximations near boundaries may pose a problem, since the above difference operators may need function values at points which are not on the grid. One way out is to impose this technical restriction on the adaptive process (Section 3.6) which will, as a result, choose to further refine toward the boundary. The refinement will be geometric, so that without using too many points (their number is proportional to the high approximation order desired in the interior), the mesh-size near the boundary will be small enough to allow low-order approximation. In this respect the boundary behaves like a singular curve. Incidentally, for certain error norms (correspondingly: for certain functions G), lower order can be used (correspondingly: will be affected by the adaptive process) near the boundary without spoiling the global order of approximation. (Cf. Bramble and Hubbard (1962).)

6. RELAXATIONS WITH UNIFORM SMOOTHING RATES

A full version of this section appears as Section 6 in Brandt (1978b). Here we summarize the main points through simple examples.

The role of relaxation sweeps in multi-grid algorithms is to smooth the error (Section 2.1). The efficiency of relaxation is therefore measured by its "smoothing factor" $\bar{\mu}$ and the corresponding "smoothing rate" $\bar{\nu} = |\log \bar{\mu}|^{-1}$. The smoothing factor is defined in terms of the local mode analysis. Namely, if $\mu(\theta)$ is the convergence factor per relaxation sweep of the θ Fourier-component (see for example (2.7)) then,

$$\bar{\mu} = \max_{\frac{\pi}{2} \le |\theta| \le \pi} \mu(\theta), \quad \text{where} \quad |\theta| = \max |\theta_i| \ . \tag{6.1}$$

The range $\frac{\pi}{2} \leq |\theta| \leq \pi$, which is somewhat arbitrary (see Section A.3 in Brandt (1977a)), is chosen because these are the components which are too high to be seen on the coarser (2h) grid, so they cannot normally be reduced by the coarse-grid corrections. This definition seems to assume an infinite domain (where the Fourier expansion is made), but the behavior of such high-frequencies is practically independent of the domain. Thus, the smoothing rate $\bar{\nu}$ is, roughly, the number of relaxation sweeps required to reduce all high-frequency components by the factor 1/e. $\bar{\mu}$ and $\bar{\nu}$ can be calculated for any relaxation scheme by the MUGTAPE (1978a) routine SMORATE. A table of representative values is given in Brandt (1977a), pp. 351-352.

For uniformly elliptic operators all (reasonable) relaxation schemes have bounded smoothing rates. (See the general theorems in Section 3.1 of Brandt (1976).) For singular-perturbation problems, however, many relaxation schemes will have $\bar{\nu}$ which grows indefinitely as the size of the perturbation decreases ($\varepsilon \to 0$). That is, the convergence rates of some components θ will not be bounded uniformly in ε. One may sometimes still get a nice multi-grid process if those bad components have only a small contribution to the error norms (see Poling (1978)), but it is better and safer to use other relaxation schemes, with smoothing rates which are bounded *uniformly in* ε.

Two kinds of degeneracies will usually occur in relaxing singular-perturbation problems. One kind occurs in the boundary layer, in dimension $d \geq 2$, when a highly stretched grid (as in Figure 3E) is used. To see the problem, consider the usual, pointwise Gauss-Seidel relaxation for the 5-point Laplace operator on a grid with aspect-ratio

$\alpha = h_1/h_2 << 1$. Examining the component $\theta = (0,\frac{\pi}{2})$ for example, it is easy to see that $\bar{\nu} > .5\alpha^{-2}$. Since α may be comparable to ε, this smoothing rate may be extremely slow. The same slow rate will occur for *any* point-wise relaxation. If, in addition to the Laplace operator, the differential equation has also lower-order terms (as it does, of course, in singular-perturbation problems), the trouble still occurs, since on the finest grid the higher-order term (the perturbation) is dominant.

This kind of trouble can always be avoided by using Gauss-Seidel *line* relaxation. This means a relaxation in which we scan G^h (cf. Section 2.1) not point by point, but line by line. At each line, all the values $u^h(x^h)$ associated with that line are simultaneously replaced by new values which are computed so that they simultaneously satisfy all the difference equations associated with that line. In the above example the lines should be horizontal lines ($x_2 = \text{const.}$), and the resulting smoothing rate will uniformly be $\bar{\nu} = 2/\log 5$, no matter how small α is. The same smoothing rate will be obtained generally in the boundary layer, provided line relaxation is employed with *lines perpendicular to the boundary*.

Another type of degeneracy occurs on the coarser grids, where the reduced part of the equation dominates the smoothing process. The relaxation there should be one which is suitable for the reduced equation. Still more difficult may be the case of intermediate grids, where both the reduced and the perturbation parts interact with the smoothing process. Consider for example the ordinary differential operator

$$Lu \equiv \varepsilon \frac{d^2u}{dx^2} + a \frac{du}{dx} , \tag{6.2}$$

and its lowest-order[8] stable approximations (see Section 5.2)

$$L^h u^h \equiv \frac{\varepsilon}{h^2} \{u^h(x-h) - (2+\eta)u^h(x) + (1+\eta)u^h(x+h)\} \tag{6.3a}$$

for $a \geq 0$,

$$L^h u^h \equiv \frac{\varepsilon}{h^2} \{(1-\eta)u^h(x-h) - (2-\eta)u^h(x) + u^h(x+h)\} \tag{6.3b}$$

for $a \leq 0$,

where $\eta = ah/\varepsilon$ may be moderate or large. Denote by $\bar{\mu}_+(\eta)$ and $\bar{\mu}_-(\eta)$, respectively, the smoothing factors for the forward and backward Gauss-Seidel relaxation of (6.3). Forward and backward refer to the marching direction, i.e., to the order in which the points x are relaxed. A straightforward calculation gives, for $\eta \geq 0$,

$$\bar{\mu}_+(\eta) = \bar{\mu}_-(-\eta) = \left| \frac{1+\eta}{2+\eta+i} \right| \tag{6.4}$$

$$\bar{\mu}_-(\eta) = \bar{\mu}_+(-\eta) = \left| \frac{1}{2+\eta+i(1+\eta)} \right| . \tag{6.5}$$

Observe that $\bar{\mu}_+(\eta) \to 1$ as $\eta \to \infty$, which means that the forward relaxation is not uniformly smoothing for $a > 0$, and should not be used. No "relaxation parameter" will help here. On the other hand in this case $(a > 0)$ we have $\bar{\mu}_-(\eta) < 5^{-1/2}$, so the backward relaxation has a very good uniform smoothing rate. The backward direction corresponds to the *direction of convection, or the down stream direction*, in physical problems modelled by (6.2). Generally, physical insight is an invaluable source for devising successful relaxation schemes.

For $a < 0$ the situation is reversed: Backward relaxation is useless at small ε, but the forward relaxation has excellent (very small) smoothing rates. Slightly more difficult is the case where $a = a(x)$ changes sign in the domain. In that case each relaxation direction will have slow smoothing at some part of the domain. The good scheme then is *symmetric relaxation*, i.e., sweeping forward and then backward. The smoothing factor (per single sweep) of this is

$$\bar{\mu}_s(\eta) = [\bar{\mu}_+(\eta)\bar{\mu}_-(\eta)]^{1/2} = \left|\frac{1+\eta}{3+3\eta+\eta^2+i(2+\eta)^2}\right|^{1/2} \qquad (6.6)$$

$$\leq 3^{-1/2} .$$

Hence $\bar{\nu}_s \leq 2/\log 3$, uniformly bounded for all values of η, positive or negative. Observe that, in fact, the larger is $|\eta|$ the better is the smoothing rate.

The same holds for singular-perturbation equations in higher dimensions: Very good smoothing rates are obtained by a proper choice of the relaxation marching direction. In some situations *all* marching directions should be employed successively if a uniform smoothing is to be achieved. This may require more sweeps per multi-grid cycle, which one would like to avoid. We can, in fact, construct relaxation schemes which have *bounded smoothing rates even when marching against the local convection direction*. These schemes necessarily belong to the following class.

Distributed Relaxation. In classical relaxation we relate the unknown at a grid point to the difference equation at that same point. That is to say, we change that unknown to satisfy the corresponding equation; or, as in line relaxation, we change simultaneously a set of unknowns to satisfy the corresponding set of equations. This "marriage"

between the unknown and the equation at the same point is not always natural. In many cases, especially in solving a *system* of differential equations ($q > 1$), the natural thing is to change *several* unknowns in order to satisfy just *one* difference equation. Such a scheme is called distributed relaxation. (See Lecture 7 in Brandt (1978a).) A special case of such a relaxation was suggested by Kaczmarz (1937) and analyzed by Tanabe (1971)[9]. Various cases of distributed relaxation for singular-perturbation problems are analyzed by Dinar (1978).

Let us show how distributed-relaxation yields uniformly bounded smoothing rates even when the marching direction is upstream, i.e., against the direction of convection. Take again the operator (6.3a) and assume forward relaxation. Instead of changing only the approximation $u^h(x)$ to satisfy $L^h u^h(x) = F(x)$, change now both $u^h(x)$ and $u^h(x + h)$: Change $u^h(x)$ by adding to it δ, and $u^h(x + h)$ by adding to it $-w\delta$, where w is a fixed coefficient (see below) and δ is calculated so that the equation $L^h u^h(x) = F(x)$ is satisfied after these changes. This marching process is stable for $w < (1 + \eta)/2$. The larger w the better is the smoothing rate. By taking w not far from the critical value $(1 + \eta)/2$, we can get smoothing rates $\bar{\nu}$ which are less than 1 for all η, and $\bar{\nu} = O(\eta^{-1})$ for large η.

w is called the distribution coefficient, and should not be confused with the familiar "relaxation parameter". The above scheme is called Distributed Gauss-Seidel (DGS) relaxation, because, as in the Gauss-Seidel scheme, each difference equation in its turn is fully satisfied by the

changes. For all problems examined, including incompressible Navier-Stokes equations, DGS was found to be the best smoother.

One final remark: The difference operator must be uniformly stable (see Section 5), otherwise no relaxation scheme can have uniformly bounded smoothing rates. For example, if the central difference approximation is used instead of (6.3a), even backward relaxation would have

$$\tilde{\mu}_{-}(\eta) = \left| \frac{1 - \eta/2}{2 + i(1 + \eta/2)} \right| \to 1 \quad \text{as} \quad \eta \to \infty .$$

FOOTNOTES

[1] An exception is the case when the coarse-grid difference operator L^H does not fully use the smoothness of the solution. In that case, if I_h^H in (2.4) is of sufficiently high order, then V^h will be smooth enough to be approximated by some V^H. This situation is related, however, to the use of an inappropriate approximation order, and will therefore not arise in a fully adaptive procedure.

[2] Provided the two $I_h^H u^h$ appearing in (2.9) are identically the same. A common programming error is that they differ at some special points.

[3] It is not necessary to compute the residual norm, since this particular algorithm is "fixed", its flow does not depend on the internal measures, and the number of sweeps made at each stage is prescribed in advance. For more complicated equations an "accommodative" algorithm, with internal switching criteria (e.g., the algorithm in Figure 1 above), may be desired. But, for more complicated equations, relaxation is more expensive, so that the extra work in computing $\|r^H\|$ is relatively small.

4) τ-extrapolation is produced by the same FASFMG program of MUGTAPE (1978) through simple changes shown there by Comment cards.

5) Alternatively, extra smoothness on the scale of the finest grid G^h can be used to produce a solution with errors smaller than the truncation errors *in very little work*. Indeed, if the difference equations do not exploit all the smoothness in the solution, an approximation to the level of the G^h truncation errors is obtained (with τ-extrapolation) already on one of the coarser grids. All is needed then is to interpolate from that grid to G^h, with high enough order of interpolation.

6) For example, in the approach taken by Hackbusch (1978), the solution of a coupled pair of elliptic equations requires work equivalent to too many (at least 28/3, instead of just 2) solutions of a single equation.

7) An abnormal run can usually be detected by examining the condensed output (output similar to the first three columns in Table 1). See Debugging Techniques, Lecture 18 in Brandt (1978a).

8) It is enough to study relaxation schemes for the lowest order operator, because (i) one can compute higher-order approximations via the lower-order ones (see Section 3.4). (ii) For any relaxation scheme, the smoothing-rate dependence on the approximation-order is not very significant.

9) References due to Blair Swartz and Gene Golub.

10) F^h in the definition of r^h should later be understood as the *current* right-hand side $f^h = F^h_{h/2}$ (see Fig. 1).

REFERENCES

Babuska, I. (1975). Homogenization and its Application. Mathematical and Computational Problems. In: *Numerical Solution of Partial Differential Equations III* (SYNSPADE 1975), (B. Hubbard, ed.). Academic Press, New York..

Bakhvalov, N.S. (1969). The Optimization of Methods of Solving Boundary Value Problems with a Boundary Layer. *Zurnal Vycislitel'noi Matematiki i Matematiceskoi Fiziki* 9, 841-859.

Bramble, J.H. and Hubbard, B.E. (1962). On the Formulation of Finite Difference Analogues of the Dirichlet Problem for Poisson's Equation. *Numerische Mathematik* 4, 313-327.

Brandt, A. (1973). Multi-Level Adaptive Technique (MLAT) for Fast Numerical Solution to Boundary Value Problems. *Proceedings of the 3rd International Conference on Numerical Methods in Fluid Mechanics* (Paris, 1972), *Lecture Notes in Physics* 18, pp. 82-89, Springer-Verlag, Berlin and New York.

Brandt, A. (1976). Multi-Level Adaptive Techniques, IBM Research Report RC6026.

Brandt, A. (1977a). Multi-Level Adaptive Solutions to Boundary-Value Problems. *Mathematics of Computation* 31, 333-390.

Brandt, A. (1977b). Multi-Level Adaptive Solutions to Partial Differential Equations - Ideas and Software. *Proceedings of Symposium on Mathematical Software* (Mathematics Research Center, University of Wisconsin, March 1977), (John Rice, ed.), pp. 277-318. Academic Press, New York.

Brandt, A. (1978a). Lecture Notes of the ICASE Workshop on Multi-Grid Methods. With contributions also by J.C. South (Lecture 8), J. Oliger (10), F. Gustavson (13), C.E. Grosch (14), D.J. Jones (15) and T.C. Poling (16). ICASE, NASA Langley Research Center, Hampton, Virginia.

Brandt, A. (1978b). Multi-Level Adaptive Techniques (MLAT) and Singular-Perturbation Problems, Mathematics Research Center Report, University of Wisconsin, Madison. To appear.

Brandt, A. (1978c). Multi-Grid Solutions to Flow Problems, Numerical Methods for Partial Differential Equations. Proceedings of Advanced Seminar, Mathematics Research Center, University of Wisconsin, Madison, October 1978. (S. Parter, ed.). To appear.

Brandt, A., Dendi, J.E., Jr. and Ruppel, H. (1978). The Multi-Grid for the Pressure Iteration in Eulerian and Lagrangian Hydrodynamics, LA-UR 77-2995 report of Los Alamos Scientific Laboratory, Los Alamos, New Mexico.

Ciarlet, P.G. (1978). *The Finite Element Method for Elliptic Problems*. Studies in Mathematics and its Applications. (J.L. Lions, G. Papanicolaou and R.T. Rockafellar, eds.). North-Holland Publishing Co., New York.

Dahlquist, G. and Björck, A. (1974). *Numerical Methods*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

Dinar, N. (1978). Fast Methods for the Numerical Solution of Boundary-Value Problems. Ph.D. Thesis, Weizmann Institute of Science, Rehovot, Israel.

Frank, L.S. (1978). Coercive Singular Perturbations: Stability and Convergence. Proceedings of Conference on the Numerical Analysis of Singular Perturbation Problems, (University of Nijmegen, Nijmeden, The Netherlands, May 30 - June 2, 1978).

Hackbusch, W. (1978a). On the Fast Solution of Parabolic Boundary Control Problems, to appear in *SIAM Journal on Control and Optimization*.

Hackbusch, W. (1978b). On the Fast Solving of Elliptic Control Problems. Report 78-14, Angewandte Mathematik, Universität zu Köln.

Hyman, J.M. (1977). Mesh Refinement and Local Inversion of Elliptic Partial Differential Equations. *Journal of Computational Physics* 23, 124-134.

Kaczmarz, S. (1937). Angenaherte Auflosung von Systemen Linearer Gleichungen. *Bulletin de l'Académie Polonaise des Sciences et Lettres* A, 355-357.

Keller, H.B. (1977). Numerical Solution of Bifurcation and Nonlinear Eigenvalue Problems. In: *Applications of Bifurcation Theory*. (P. Rabinowitz, ed.), pp359-384. Academic Press, New York.

Lentini, M. and Pereyra, V.L. (1975). Boundary Problem Solvers for First Order Systems Based on Deferred Corrections. In: *Numerical Solutions of Boundary Value Problems for Ordinary Differential Equations*. (A.K. Aziz, ed.). Academic Press, New York.

MUGTAPE (1978a). A Tape of Multi-Grid Software and Programs. Distributed at the ICASE Workshop on Multi-Grid Methods. Contributions by A. Brandt, N. Dinar, F. Gustavson and D. Ophir.

MUGTAPE (1978b). Updated Version of the Tape, prepared by D. Ophir, Mathematics Department, Weizmann Institute of Science, Rehovot, Israel.

Nicolaides, R.A. (1978). On Finite-Element Multi-Grid Algorithms and Their Use. ICASE report 78-8, ICASE, NASA Langley Research Center, Hampton, Virginia.

O'Malley, R.E., Jr. (1974). Boundary Layer Methods for Ordinary Differential Equations with Small Coefficients Multiplying the Highest Derivatives. *Proceedings of Symposium on Constructive and Computational Methods for Differential and Integral Equations. Lecture Notes in Mathematics* 430, 363-389. Springer-Verlag, New York.

Ophir, D. (1978). Language for Processes of Numerical Solutions to Differential Equations. Ph.D. Thesis, Mathematics Department, Weizmann Institute of Science, Rehovot, Israel.

Pereyra, V.L. (1968). Iterated Deferred Corrections for Nonlinear Boundary Value Problems. *Numerische Mathematik* 11, 111-125.

Poling, T.C. (1978). Numerical Experiments with Multi-Grid Methods. M.A. Thesis, Department of Mathematics, The College of William and Mary, Williamsburg, Virginia.

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

South, J.C., Jr. and Brandt, A. (1976). Application of a Multi-Level Grid Method to Transonic Flow Calculations, ICASE Report 76-8, NASA Langley Research Center, Hampton, Virginia.

Southwell, R.V. (1935). Stress Calculation in Frameworks by the Method of Systematic Relaxation of Constraints. I, II. *Proceedings of the Royal Society London Series A* 151, 56-95.

Spagnolo, S. (1975). *Convergence in Energy for Elliptic Operators, Numerical Solution of Partial Differential Equations III* (SYNSPADE 1975). (B. Hubbard, ed.). Academic Press, New York.

Starius, G.C. (1977a). Constructing Orthogonal Curvilinear Meshes by Solving Initial Value Problems. *Numerische Mathematik* 28, 25-48.

Starius, G.C. (1977b). Composite Mesh Difference Methods for Elliptic Boundary Value Problems. *Numerische Mathematik* 28, 242-258.

Starius, G. (1978). Numerical Treatment of Boundary Layers for Perturbed Hyperbolic Equations. Report No. 69, Department of Computer Sciences, Uppsala University, Uppsala, Sweden.

Stetter, H.J. (1978). The Defect Correction Principle and Discretization Methods. *Numerische Mathematik* 29, 425-443.

Tanabe, K. (1971). Projection Method for Solving a Singular System of Linear Equations and its Applications. *Numerische Mathematik* 17, 203-214.

Thomée, V. (1964). Elliptic Difference Operators and Dirichlet Problem. *Contributions to Differential Equations* 3, 301-324.

Thomée, V. and Westergren, B. (1968). Elliptic Difference Equations and Interior Regularity. *Numerische Mathematik* 11, 196-210.