JOURNAL OF COMPUTATIONAL PHYSICS 92, 349–368 (1991)

A Deferred-Correction Multigrid Algorithm Based on a New Smoother for the Navier–Stokes Equations

PAOLO LUCHINI

Istituto di Gasdinamica, Facoltà di Ingegneria, University of Naples, Italy

Received February 6, 1989; revised December 5, 1989

An algorithm which brings together the techniques of multigrid and deferred correction through their common relationship with imperfect Newton iteration manages to combine the ease of calculation of a low-order with the accuracy of a high-order difference approximation of any given differential-equation problem. A stable explicit Gauss–Seidel relaxation algorithm for the ψ-ζ Navier–Stokes equations based on an appropriate kind of "upwinding" of ψ- as well as ζ-derivatives, especially developed for use as a multigrid smoother in this context, is presented and the complete algorithm is tested on the standard conservative second-order discretization of the driven-cavity problem. © 1991 Academic Press, Inc.

1. INTRODUCTION

The numerical solution of a set of differential equations is composed of two phases: the discretization of the original problem, that is, the identification of a new problem in a finite number of unknowns that is, in a suitable sense, an approximation of the problem originally posed in a continuous variable space, and the solution, which is in turn generally approximate, of the discretized problem. Several techniques are in use both for the discretization of the problem (e.g., difference approximations based on Taylor expansions or splines, finite elements, boundary-integral methods, spectral methods) and for the solution, necessarily iterative when the equations are nonlinear, of the discretized problem (e.g., variational methods, Newton iteration with direct inversion of the derivative matrix, and Gauss–Seidel, ADI, and multigrid relaxation methods).

The method of solution is hardly ever studied separately from the technique of discretization but they are, indeed, distinct aspects of the problem, since the latter involves finding as accurate as possible a discrete approximation of the given differential equations whereas the former involves finding a rough approximation of their solution that may be fast to compute and just good enough to yield a stable feedback loop.

This observation is at the basis of deferred-correction techniques, which consist of combining an accurate discretization of the given problem with an iterative solution method devised for a lower-order approximation in such a way as to solve the former through the latter. In particular, a perfect candidate for the lower-order solution algorithm is one based on the multigrid approach, as will be illustrated in the next section. A computer code designed on these principles keeps the two aspects of accuracy and speed of execution separate down to program structure, and allows an efficient solution method to be applied to a wide range of finite-difference and finite-element discretizations by replacing a single subroutine which basically contains the discretized equations only. As a side-effect of this separation of functions, the speed of execution becomes more or less independent of the form of the discretized equations, thus making complicated and previously unaffordable high-order formulations very appealing.

A multigrid algorithm needs an underlying relaxation procedure to be used as a "smoother." The degrees of freedom offered by the separation between the form of the discretized equations which is finally solved and the one which is used inside the relaxation algorithm has allowed us to taylor this form to the needs of the smoother. Rather than adopting a pre-existing solution procedure of the Navier–Stokes equations as a smoother, as done in most previous multigrid codes, we have developed an ad hoc first-order discretization, based on a suitable "upwinding" of ψ- as well as ζ-derivatives in the vorticity-transport equation, which would be of little use as a stand-alone algorithm but is very well suited as a multigrid smoother because it makes a Gauss–Seidel relaxation loop stable at all Reynolds numbers.

The plan of the paper is as follows. In Section 2 we shall illustrate a similarity between the deferred-correction and multigrid approaches which gives a hint that they should work well together in a combined method. In Section 3 we shall describe the resulting general method, which turns out to be similar to those developed in [1–4]. In Section 4 we shall present the new discretization of the ψ-ζ Navier–Stokes equations which lends itself to Gauss–Seidel relaxation and was expressly devised for the multigrid environment; and in Section 5 we shall give some results of applying the proposed algorithm to the well-known driven-cavity test case.

2. The Connection between Deferred Correction and Multigrid: Imperfect Newton Iteration

Before describing the more practical advantages of bringing together the techniques of deferred correction (also called "defect correction") and multigrid into a combined algorithm, let us point out a philosophical reason to do so: they can both be seen as instances of a single technique which may be called imperfect Newton iteration.

Basically all the existing methods for the numerical solution of differential equations were initially developed for linear boundary-value problems and consist of algorithms for the inversion of matrices of a more or less specialized form. The extension of these techniques to nonlinear boundary-value problems is always explicitly or implicitly based on Newton iteration. (With one important exception:

variational procedures based on the minimization of a convex functional. A convex functional which is minimized by the solution, however, is not available for all problems and in particular not for the Navier–Stokes equations.)

Newton iteration is the well-known technique according to which the solution of a nonlinear equation system $NL(\mathbf{x})=0$ in the vector of unknowns $\mathbf{x}$ is pursued by iteratively linearizing the original problem in a neighbourhood of a tentative approximate solution and solving the linearized problem for a new approximate solution until a satisfactory approximation is attained. The process may be represented schematically as follows:

ALGORITHM A (Newton iteration).

given equation $NL(\mathbf{x})=0$ and L linearization (differential) of NL,

solve $NL(\mathbf{x}_n)+L(\boldsymbol{\delta}\mathbf{x})=0$ for $\boldsymbol{\delta}\mathbf{x}$,

iterate $\mathbf{x}_{n+1}=\mathbf{x}_n-L^{-1}[NL(\mathbf{x}_n)]$.

The difficult part in Newton iteration is the inversion of the linear operator L (i.e., a matrix). If direct (Gauss) inversion is to be used for this purpose the very satisfactory convergence rate of Newton iteration, which is quadratic, i.e., characterized by an error asymptotically going down with iteration number n as $\exp(-2^n)$, is more than offset by the computation time of matrix inversion, which increases with the number of unknowns N as N^3, and by the storage, proportional to N^2, necessary for the matrix elements. When the form of the matrix L does not lend itself to a method less time- and storage-consuming than Gauss inversion, Newton iteration is not to be preferred to other iterative techniques.

However, there is a way to recover some of the nice convergence properties of Newton iteration without losing the iteration speed of other methods: to replace L by an approximation L' which is amenable to a faster inversion. The resulting algorithm may be called imperfect Newton iteration:

ALGORITHM B (Imperfect Newton iteration).

given equation $NL(\mathbf{x})=0$, L linearization (differential) of NL and L' easy to invert approximation of L,

solve $NL(\mathbf{x}_n)+L'(\boldsymbol{\delta}\mathbf{x})=0$ for $\boldsymbol{\delta}\mathbf{x}$,

iterate $\mathbf{x}_{n+1}=\mathbf{x}_n-L'^{-1}[NL(\mathbf{x}_n)]$.

It is clear that $NL(\mathbf{x}_n)=0$ when $\mathbf{x}_{n+1}=\mathbf{x}_n$, so that if the iteration converges the problem will have been solved; on the other hand, it cannot be guaranteed, as it can under some restrictions for true Newton iteration, that the process *shall* converge provided the initial guess is close enough to the solution, nor is convergence quadratic. Nevertheless it is reasonable to expect if L' is in some sense close to L that the process will converge quickly, and practice confirms this expectation. (More precisely, the theoretical condition under which the iteration can be shown to converge is that the matrix $1-L'^{-1}L$ should have all eigenvalues less than unity in modulus; however, for a practical problem such as the Navier–Stokes equations it is difficult to verify this condition other than by observing the convergence of the algorithm itself.)

Although quicker than Algorithm A, Algorithm B is still formulated in terms of matrices. One can dispense with matrices completely by replacing L' by a finite difference of an operator NL', of which L' is the differential. This further modification gives

ALGORITHM C (Modified imperfect Newton iteration).

given equation $NL(\mathbf{x})=0$ and NL', easy to invert nonlinear operator such that its differential is an approximation of the differential of NL,

solve $NL(\mathbf{x}_n)+NL'(\mathbf{x}_n+\boldsymbol{\delta}\mathbf{x})-NL'(\mathbf{x}_n)=0$ for $\boldsymbol{\delta}\mathbf{x}$,

iterate $\mathbf{x}_{n+1}=NL'^{-1}[NL'(\mathbf{x}_n)-NL(\mathbf{x}_n)]$.

Again, it is evident from Algorithm C that $NL(\mathbf{x}_n)=0$ if $\mathbf{x}_{n+1}=\mathbf{x}_n$. Moreover, Algorithm C can be expected to have similar convergence properties to Algorithm B, but no matrix appears any longer.

Those familiar with either of the techniques of deferred correction and multigrid will have recognized the above reasoning, as it is used in both.

To obtain deferred correction, just identify NL with a high-order discretization of a given differential problem and NL' with a low-order discretization of the same problem. Provided a fast solution procedure exists for NL', Algorithm C allows that same procedure to be used for the solution of NL, by putting on the r.h.s. of NL', rather than zero, the "deferred correction" $NL'(\mathbf{x}_n)-NL(\mathbf{x}_n)$. Notice that, on looking at Algorithm C alone, it is not at all apparent that such a process should converge. It is only through its relation to Algorithm B and the fact that NL and NL', being different discretizations of the same problem, are expected to have mutually close differentials, that Algorithm C may be justified.

To obtain multigrid, choose NL to be a fine-grid discretization of the given differential problem and NL' to be a discretization of the same problem on a coarser grid. The foregoing reasoning may be repeated and Algorithm C allows a fine-grid solution to be obtained using an inversion algorithm that works on the coarser grid. Contrary to the previous case, however, care must be taken of the fact that the coarser grid contains less information than the finer one, namely it is only able to represent a shorter range of wavenumbers, by providing a separate procedure (smoother) to handle the higher-wavenumber portion of the error. Fortunately almost any relaxation algorithm has the property of dealing with high-wavenumber error components much better than with low-wavenumber ones. Suitable restriction and prolongation procedures must also be provided to move the low-wavenumber part only of the solution and error between grids.

3. Deferred-Correction Multigrid

Deferred correction is used often in connection with the Navier–Stokes equations. In fact, whereas several relaxation algorithms are known which work for an upwinded, first-order discretization of these equations, these algorithms generally fail when applied to a second- or higher-order discretization of the same equations, and in particular to the standard conservative central ψ–ζ discretization we shall use as an example later.

However, first-order upwinded convection terms are known to produce poor, and in some cases outright wrong [5], results in the very range of large mesh Reynolds numbers for which they are necessary and therefore are not considered a satisfactory solution. There are two ways out of this difficulty: to use specially designed higher-order upwind discretizations, as for instance in [6], or to make it such that the properties of upwinded differences are exploited during the iterative process but the final converged solution satisfies a higher-order approximation of the differential equations, e.g., by using upstream-weighted difference schemes [7, 8] or deferred correction.

In connection with deferred correction, it should be noticed that Algorithm C as written above involves the exact inversion of the approximate problem NL' at each iteration step. While in an iterative context it is reasonable to replace this exact inversion by an approximate one, such as may be provided by one or more iterations of a relaxation algorithm, it is necessary, for Algorithm C to resemble Newton iteration, that this approximation be sufficiently good. In particular, it is known that deferred correction of the upwind to the central-difference approximation of the ψ-ζ Navier-Stokes equations, or, for that matter, even of the advection–diffusion equation, does not yield a stable iteration loop if simple point-Gauss–Seidel relaxation is used in the place of NL'^{-1}. On the other hand, this technique has been applied with success in conjunction with line relaxation and ADI algorithms [9–11].

Since an accurate and fast approximate inversion of NL' is necessary, and given the conceptual relationship between deferred-correction and multigrid techniques indicated in the previous section, it becomes very natural to roll deferred correction and multigrid into a single combined algorithm, as was proposed in general in [1, 2] and applied to the Euler equations in [3, 4].

A deferred-correction multigrid (DCMG) procedure may be viewed in two complementary ways: as a deferred-correction algorithm which uses a multigrid iteration as the approximate inversion of NL', or as a multigrid algorithm that uses a higher-order discretization rather than an even finer grid to calculate correction terms for the finest-grid equations, similar to those which are added to coarser-grid equations in order to comply with imperfect Newton iteration (Algorithm C). In pseudocode, the outmost loop of a DCMG program is

ALGORITHM DCMG (Deferred-correction multigrid).

```
repeat
    apply low-order smoother on the finest grid
    restrict solution and residuals to the next coarser grid
    calculate correction terms for the coarser grid (difference between coarse-
    grid and fine-grid residuals)
    apply low-order smoother
    restrict solution and residuals to the next coarser grid
    ....
    prolong the solution difference to the next finer grid
    apply low-order smoother
    prolong the solution difference to the finest grid
    apply low-order smoother
    calculate correction terms for the finest grid (difference between low-order
    and high-order residuals)
until  high-order residuals satisfactorily small.
```

As is seen, a DCMG code requires a standard multigrid procedure, composed of a sequencer to handle the cascade of grids, restriction and prolongation routines, and a smoother written for the low-order equations, with the addition of a residual-calculating routine for the high-order equations which is invoked once per multigrid iteration to provide deferred-correction terms.

One unique asset of the DCMG approach is the flexibility ensuing from the almost complete segregation of the functions of providing accuracy and speed of execution into different program modules. In particular, whereas the relaxation procedure for the low-order equations (the smoother) is the most frequently executed piece of code and should be kept as simple as possible, the high-order equations, in spite of being the very ones which are being solved, are used sparingly in the calculation, their residuals needing only to be computed once per multigrid iteration. Of course, this is quite consistent with the Newton iteration procedure, in which residuals are only calculated once per matrix inversion, but it is a big departure from other existing algorithms and opens some new possibilities.

In particular, a DCMG program is a sort of "black box" into which different discretizations of the same problem may be quickly plugged in and tried out. This capability, precious for problems in which the accuracy of one or another discretization method is questionable, is not shared by any other differential-equation solution algorithm.

Moreover, the use of approximations of a high order is particularly convenient, even in cases where it leads to complicated difference equations, because these difference equations are only used moderately often for computing their residuals and have little effect on the overall computation time. In fact, it would be quite conceivable to adopt a DCMG algorithm to solve finite-element equations, thus providing a much faster alternative to the traditional direct-matrix-inversion methods.

Let us finally remark that the deferred-correction and multigrid algorithms have already appeared together in Navier–Stokes solving programs in the past [12–14], but with a very different approach. In those works a complete implicit Navier–Stokes solver, previously developed by the same or other authors as a stand-alone program, which happened to exploit also the deferred-correction technique, was inserted in a multigrid cycle for the purpose of gaining an improved rate of convergence. In DCMG, instead, deferred correction is used outside, and in some sense towards completion, of the multigrid cycle for the purpose of solving a high-order difference formulation of a differential problem by adopting a very simple relaxation algorithm, which would be of no use as a stand-alone unit, as a smoother for the multigrid procedure. A smoother with these properties will be presented in the next section.

A DCMG program for the Navier–Stokes equations with a structure very similar to the present one, but working with primitive-variable equations and using a different smoother, has appeared very recently [15].

4. A New Explicit Smoother for the Navier–Stokes Equations

A relaxation algorithm that is useful as a smoother must have the property of damping large-wavenumber error components (those having a wavelength comparable to mesh size), but it is not very important how it behaves with respect to low-wavenumber components, as those are taken care of by a different part of the multigrid program. In addition, a smoother ought to be simple and fast, because it is the most frequently executed piece of code. For these reasons it is generally found in connection with simple elliptic problems (such as the ones governed by the Laplace, Poisson, and biharmonic equations) that an explicit point-Gauss–Seidel smoother is preferable to more complicated line-relaxation or ADI methods [16, Sect. 3.3], because the large-wavenumber properties of the latter are not enough superior to those of the former to compensate for the increased computation time per iteration.

Relaxation algorithms for the Navier–Stokes equations, however, are not as nice, and in particular, no explicit Gauss–Seidel procedure for the two-dimensional Navier–Stokes equations in ψ-ζ form seems to have been found to converge at relatively large Reynolds numbers (by which we mean large enough that the Reynolds number based on mesh size is larger than unity but not so large that the differential equations themselves are unstable) without the introduction of inpractical underrelaxation factors: in fact, even line-relaxation procedures, which are currently the preferred solution either in their ADI or line-Gauss–Seidel variation (see, e.g., [9–11, 17, 18]), converge only for the difference equations containing the convective terms in upwinded form, which is a way of restoring a sort of diagonal dominance into the derivative matrix, and, even so, often require the introduction of underrelaxation factors which must be given different values for different Reynolds numbers.

In this section we shall see how trying to understand the reason for the necessity of these underrelaxation factors leads to the development of a stable point-Gauss–Seidel smoother.

The two-dimensional, steady, incompressible Navier–Stokes equations in ψ–ζ form are

$$\Delta_2 \psi = \zeta \tag{1a}$$

$$\Delta_2 \zeta = \mathrm{Re}(\psi_y \zeta_x - \psi_x \zeta_y), \tag{1b}$$

where ψ and ζ denote the stream function and vorticity of the flow field and Re is the Reynolds number. Two very common discretizations of these equations on a "cross" formed by five points, which we shall denote by subscripts E, N, W, S, and C for east, north, west, south, and center, respectively, are the second-order form given by

$$\psi_E + \psi_N + \psi_W + \psi_S - 4\psi_C = h^2 \zeta_C \tag{2a}$$

$$\zeta_E + \zeta_N + \zeta_W + \zeta_S - 4\zeta_C = \mathrm{Re}(\psi_{NS}\zeta_{EW} - \psi_{EW}\zeta_{NS})/4, \tag{2b}$$

where all first derivatives are approximated by central differences, and the first-order upwind form given by Eq. (2a) together with

$$\begin{aligned}\zeta_E &+ \zeta_N + \zeta_W + \zeta_S - 4\zeta_C \\ &= \mathrm{Re}[(\psi_{NS} - |\psi_{NS}|)\,\zeta_{EC} + (\psi_{NS} + |\psi_{NS}|)\,\zeta_{CW} \\ &+ (\psi_{EW} + |\psi_{EW}|)\,\zeta_{NC} - (\psi_{EW} - |\psi_{EW}|)\,\zeta_{CS}]/4,\end{aligned} \tag{3}$$

where symbols with two subscripts denote differences of the corresponding values (e.g., $\psi_{NS} = \psi_N - \psi_S$). In the above equations h denotes mesh size, which is assumed constant for the sake of simplicity even though all that we are going to say can easily be extended to variable-spacing meshes. The reason why Eq. (3) is introduced at all is that Eq. (2b), although more precise, is not fit for relaxation algorithms because the value of vorticity at the central point ζ_C does not appear in the convective term (the right-hand side).

The function of the smoother in a multigrid algorithm is mainly to damp short-wavelength components of the error, and its long-wavelength behaviour does not really matter. This makes simple explicit smoothers preferable to implicit ones, because the larger computation time that the latter require is not sufficiently compensated for by the marginal improvement they offer in short-wavelength damping (It is at long wavelengths that implicit algorithms really outperform explicit ones). Another nonnegligible point in favour of choosing an explicit smoother is that the smoother is the most frequently executed piece of code, and if it is a very simple routine a considerable speed advantage can be obtained by optimizing its coding, for instance, by means of an assembler, whereas such a measure is too costly in programming time for a complicated routine. An explicit smoother is also much better fit for parallel processing.

Despite these considerations, the multigrid programs for the Navier–Stokes equations (1) that can be found in the literature (e.g., [12–14, 19]) utilize implicit smoothers. The reason is that explicit Gauss–Seidel relaxation methods for these equations, even in upwinded form, do not converge at relatively high Reynolds numbers. (An explicit smoother for the upwinded Navier–Stokes equations in primitive variables has been introduced in [20] under the name of distributive Gauss–Seidel (DGS) technique for the purpose of being used in a multigrid algorithm and has been later adopted in [21, 22]. It is, however, reported to diverge in some cases.) In fact, even implicit smoothers for the upwind equations are generally reported [6, 12–14, 19] to require a certain amount of overrelaxation of the continuity equation (2a) and underrelaxation of the vorticity equation (3) in order to avoid instability, which gives us a clue to what is wrong with standard upwinding.

The consideration that upwinding restores diagonal dominance into the derivative matrix, as set forth, e.g., in [9] and in Appendix A of [13], is true only if in the vorticity-transport equation vorticity only is treated as an unknown, with a known stream-function field. This is exactly the condition that the combination of overrelaxation and underrelaxation adopted in [6, 12–14, 19] tends to establish, giving a faster smoothing rate to the stream function than to the vorticity, much as though the continuity equation were solved exactly for the stream function for any one iteration of the vorticity distribution. (A more efficient way to achieve the same aim would probably be to perform a larger number of relaxation sweeps on the continuity equation than on the vorticity equation, rather than underrelaxing the latter.)

We are going to pursue a different approach: to seek a formulation of the difference equations that takes explicitly into account that vorticity and stream function are to be relaxed simultaneously (any first-order formulation of the equations may be chosen as the basis for a smoother in the DCMG context, since this choice has no bearing on the accuracy of the solution obtained). In particular, it may be observed that if upwind differences are used for vorticity in the convective terms but central differences are retained for stream function, as is done in the classical formulation of Eq. (3), the value of stream function at the central point ψ_C does not appear in the vorticity equation and is necessarily determined by the continuity equation only; if, on the other hand, lateral differences (not necessarily in the upwind direction) are used for the derivatives of stream function as well, the value of stream function at the central point acquires a considerable influence in the vor- [illegible] algorithm. The safest way to take this influence into account in a Gauss–Seidel-type algorithm is to solve the two equations for the central values of ψ and ζ simultaneously. A close look at what happens in doing so will also reveal the appropriate choice of the sides from which ψ-derivatives should be taken.

Let us first of all consider the linearized form of Eqs. (1), since it is this form which determines the properties of the algorithm even if it does not appear explicitly in the deferred-correction formulation. We may write a five-point first-order difference equation system for this linearized problem as

$$\psi_E^{(1)}+\psi_N^{(1)}+\psi_W^{(1)}+\psi_S^{(1)}-4\psi_C^{(1)}-h^2\zeta_C^{(1)}=-R_{\text{cont}} \tag{4a}$$

$$\begin{aligned}&\zeta_E^{(1)}+\zeta_N^{(1)}+\zeta_W^{(1)}+\zeta_S^{(1)}-4\zeta_C^{(1)}-h\,\text{Re}[s_1\psi_y^{(0)}(\zeta_C^{(1)}-\zeta_1^{(1)})\\&\quad+s_2\zeta_x^{(0)}(\psi_C^{(1)}-\psi_2^{(1)})-s_3\psi_x^{(0)}(\zeta_C^{(1)}-\zeta_3^{(1)})\\&\quad-s_4\zeta_y^{(0)}(\psi_C^{(1)}-\psi_4^{(1)})]=-R_{\text{vort}},\end{aligned} \tag{4b}$$

where superscripts (0) and (1) denote base values and corrections, R_{cont} and R_{vort} are the residuals of the continuity and vorticity-transport equations, which may include the constant correction terms needed for the multigrid and/or deferred-correction processes, and in Eq. (4b) ζ_1 may stand for either ζ_W or ζ_E, ψ_2 for either ψ_S or ψ_N, ζ_3 for either ζ_S or ζ_N, and ψ_4 for either ψ_W or ψ_E, and the multipliers s_1, s_2, s_3, and s_4 equal $+1$ or -1 depending on whether the first or the second alternative is chosen in each case. It is not necessary for the present purpose to specify exactly how derivatives $\psi_y^{(0)}$, $\zeta_x^{(0)}$, $\psi_x^{(0)}$, and $\zeta_y^{(0)}$ are discretized. Inverting Eqs. (4) with respect to $\psi_C^{(1)}$ and $\zeta_C^{(1)}$ gives

$$\psi_C^{(1)}=(a_2b_1-h^2b_2)/D \tag{5a}$$

$$\zeta_C^{(1)}=(4b_2-a_1b_1)/D, \tag{5b}$$

where

$$a_1=h\,\text{Re}(s_2\zeta_x^{(0)}-s_4\zeta_y^{(0)}) \tag{6a}$$

$$a_2=4+h\,\text{Re}(s_1\psi_y^{(0)}-s_3\psi_x^{(0)}) \tag{6b}$$

$$b_1=\psi_E^{(1)}+\psi_N^{(1)}+\psi_W^{(1)}+\psi_S^{(1)}+R_{\text{cont}} \tag{7a}$$

$$\begin{aligned}b_2&=\zeta_E^{(1)}+\zeta_N^{(1)}+\zeta_W^{(1)}+\zeta_S^{(1)}\\&\quad+h\,\text{Re}(s_1\psi_y^{(0)}\zeta_1^{(1)}+s_2\zeta_x^{(0)}\psi_2^{(1)}\\&\quad+s_3\psi_x^{(0)}\zeta_3^{(1)})+s_4\zeta_y^{(0)}\psi_4^{(1)})+R_{\text{vort}}\end{aligned} \tag{7b}$$

$$\begin{aligned}D&=4a_2-h^2a_1\\&=16+4h\,\text{Re}(s_1\psi_y^{(0)}-s_3\psi_x^{(0)})-h^3\,\text{Re}(s_2\zeta_x^{(0)}-s_4\zeta_y^{(0)}).\end{aligned} \tag{8}$$

Equation (8) is enlightening as to the problem of "upwinding" ψ-derivatives. The equivalent of pursuing diagonal dominance, in our approach in which two unknowns are recalculated at once, is to make the determinant D as large as possible, so that recalculating ψ_C and ζ_C involves the least correction. This means choosing multipliers s_1, s_2, s_3, and s_4, and indirectly the directions of ζ_1, ψ_2, ζ_3, and ψ_4 which are tied to those, so that the terms $s_1\psi_y^{(0)}$, $-s_2\zeta_x^{(0)}$, $-s_3\psi_x^{(0)}$, and $s_4\zeta_y^{(0)}$ of Eq. (8) are all positive. For the two ζ-derivatives the result is classical upwinding, i.e., the side from which they should be taken is determined by the sign of the ψ-derivatives; for the two ψ-derivatives we obtain the new result that, symmetrically, the side from which they should be taken is determined by the sign of the ζ-derivatives. Notice that it is not correct to take ψ-derivatives from the same side from which ζ-derivatives are taken; in fact such a solution, which we anyhow tried out of curiosity, does not produce an algorithm any better than upwinding ζ-derivatives only does.

It remains to be seen which full nonlinear difference approximation of Eqs. (1) has Eqs. (4) as its linearized version. The obvious answer is to replace each derivative by a suitable difference expression, choosing for all first derivatives lateral differences taken from a side conforming to the above criteria. There are, however, two drawbacks to doing so: one is that the signs of the ψ and ζ differences determine the choice of the side from which each other should be taken, but the signs themselves may in turn depend on this choice, causing nontrivial programming problems; the other is that the resulting equation system is quadratic with respect to the unknowns ψ_C and ζ_C (although not for all combinations of signs), and its exact resolution involves a square root which is slow in execution. However, there is not only one nonlinear equation corresponding to a given linearized form. A computationally more efficient solution is furnished by the difference equation

$$\begin{aligned}\zeta_E + \zeta_N + \zeta_W + \zeta_S - 4\zeta_C \\ &= \mathrm{Re}[(\psi_{NS} - |\psi_{NS}|)\,\zeta_{EC} + (\psi_{NS} + |\psi_N|)\,\zeta_{CW} \\ &\quad - (\psi_{EW} + |\psi_{EW}|)\,\zeta_{NC} - (\psi_{EW} - |\psi_{EW}|)\,\zeta_{CS} \\ &\quad + (\zeta_{EW} + |\zeta_{EW}|)\,\psi_{NC} + (\zeta_{EW} - |\zeta_{EW}|)\,\psi_{CS} \\ &\quad - (\zeta_{NS} - |\zeta_{NS}|)\,\psi_{EC} - (\zeta_{NS} + |\zeta_{NS}|)\,\psi_{CW} - \psi_{NS}\zeta_{EW} + \psi_{EW}\zeta_{NS}]/4. \qquad (9)\end{aligned}$$

As may be seen, in the r.h.s. of Eq. (9) the convective term is added twice and subtracted once and is written the first time with upwinded ζ-derivatives, the second time with "upwinded" ψ-derivatives (in the sense of the above discussion), and the third time with all central differences. The net result is that Eq. (1b) is approximated to first order, the system formed by Eqs. (2a) and (9) is linear in ψ_C and ζ_C and has the same coefficients as Eqs. (4) and all the calculations are straightforward.

To finish this section, a few words must be spent on the ordering of recalculation of the Gauss–Seidel cycle. Any Gauss–Seidel algorithm, since values are put back as soon as they are recalculated, is sensitive to the order in which data points are traversed. Sometimes a linear order in which rows of points are scanned in strict succession is chosen just because it is the easiest to program. Five-point difference equations like the ones dealt with in the foregoing, however, also allow a different choice which in a multigrid context is more than worth the, slight indeed, extra programming effort: the "hopscotch" ordering of Sheldon [23] and Gourlay [24] (now more often called "red–black" ordering), which is like alternately recalculating points corresponding to the black and red squares of a chequerboard. Since each "red" point is surrounded by four "black" points and vice versa, it is always possible to explicitly recalculate the variables pertaining to any single point of one "colour" from those of the other. Each relaxation sweep is composed of two phases, one in which all the "black" points are calculated from the "red" ones and one in which the "red" points are recalculated from the "black." Notice that knowledge of the solution at only half the points (i.e., either the black or the red) is sufficient to reconstruct the other half exactly.

The hopscotch method has the particular merit, in a multigrid context, of not introducing disuniformities in the spatial distribution of residuals. In fact a Gauss–Seidel algorithm necessarily leaves the points which have been recalculated last in a different condition from those less recently recalculated (thinking of the iteration number as a continuous time variable, one could say that the former points are a fraction of a step ahead of the latter) thus giving rise to perturbations in the distribution of residuals which must be filtered out by the interpolation algorithms used to pass from one grid to the next coarser or finer. Instead, in a hopscotch algorithm there is a definite difference (of half a step in a certain sense) between "black" and "red" points, but each of these two classes is homogeneous, since all of its members are recalculated using only information coming from the other class. Now, in a multigrid scheme in which mesh size is doubled from one level to the next, all the points belonging to the coarser mesh happen to be of the same "colour" on the finer one, and therefore their residuals may be transferred to the coarser mesh by simple injection with no fear of dangerous short-wavelength disuniformities caused by the relaxation sweep. In fact, as also pointed out in [2], injection is particularly well suited to red–black smoothers, provided a factor of $\frac{1}{2}$ is applied to the fine-mesh residuals before subtracting them from the coarse-mesh residuals to account for the fact that two iteration steps are effectively performed in going from the black to the red and back to the black points; even the theoretical disadvantages of injection can be circumvented by considering injection as a five-point restriction with weights of $\frac{1}{2}$ for the center and $\frac{1}{8}$ for the end points, which is possible since when a red-to-black sweep has just been performed (assuming the center is red) black residuals are zero. When going back from the coarser to the finer mesh, corrections for the points falling exactly in between two points of the coarser mesh will be calculated (by linear interpolation) as the average of the corrections obtained for the two enclosing points, which are always one "red" and the other "black"; corrections for the other points need not be calculated at all, since the full values of the variables pertaining to these points will be calculated directly from the previous ones in the following hopscotch sweep. Again a homogeneous situation is obtained, thus avoiding the introduction of spurious short-wavelength error components.

5. Application to Driven-Cavity Flow

As a test for the algorithm we programmed the resolution of the second-order conservative finite-difference form of the Navier–Stokes equations for the driven-cavity problem, on which a vast literature is available (e.g., [12–14, 22]), using both constant- and variable-spacing meshes. The difference equations used are Eq. (2a) and

$$\zeta_E+\zeta_N+\zeta_W+\zeta_S-4\zeta_C=\mathrm{Re}(\psi_{yE}\zeta_E-\psi_{yW}\zeta_W-\psi_{xN}\zeta_N+\psi_{xS}\zeta_S)/4, \qquad (10)$$

where ψ_{yE}, ψ_{yW}, ψ_{xN}, and ψ_{xS} denote central-difference approximations of derivatives about the corresponding points. Notice that the conservative form (10)

C-2] difference formula in the smoother and the Woods [26] third-order accurate formula, which also contains the value of vorticity at the point next to wall, in the deferred correction. In the variable-spacing runs a one-dimensional stretching transformation was applied independently to either coordinate, similar to that in [13]. A fixed number of smoother sweeps were performed at each multigrid level; on trying different numbers of sweeps it was found best to perform six sweeps per level when going upwards from finer to coarser levels and three sweeps per level when going downwards from coarser to finer.

Figures 1–4 report the streamline patterns obtained for $\mathrm{Re}=1$, 100, 1000, and 5000 using a 65×65 point uniform grid in the first three cases and an 89×89 point stretched grid in the last one. These are, of course, not different from those presented by other authors, since the same difference equations have been solved at convergence, and will not be discussed in detail. More interesting are Figs. 5–8, which show the convergence history of the algorithm for the same four cases, giving the maximum absolute values of the residuals of the continuity and vorticity-transport equations, on a logarithmic scale, as functions of run time in work units, nine work units corresponding to a full multigrid cycle (one work unit per hopscotch sweep at the finest level). It will be noticed that, just as happens for all the other Navier–Stokes solving programs, the residual-damping rate deteriorates with increasing Reynolds number. This rate, however, may be seen from Figs. 5 and 6 to be independent of mesh size, as is characteristic of a multigrid algorithm. (The calculation with 33×33 points was not performed in the other two cases because this number of points is insufficient at the higher Reynolds numbers.) Moreover, the damping rate compares favourably with the data reported for $\mathrm{Re}=1000$ in Fig. 1 of [12] and in Fig. 4 of [13], which were obtained from a multigrid program utilizing an implicit smoother, once account is taken of the fact that their work unit corresponds to one line-relaxation sweep and a line-relaxation sweep may take from 2 to 4 times as long to execute as an explicit Gauss–Seidel sweep (we cannot be very precise on this number because the two programs have not been run on the same computer).

It must also be noted that the present DCMG algorithm converges starting from a zero initial guess, although a somewhat better time can be obtained at the higher Reynolds numbers by using a solution for a lower Reynolds number as a starting point. In particular, the convergence histories reported in Figs. 5–7 were obtained

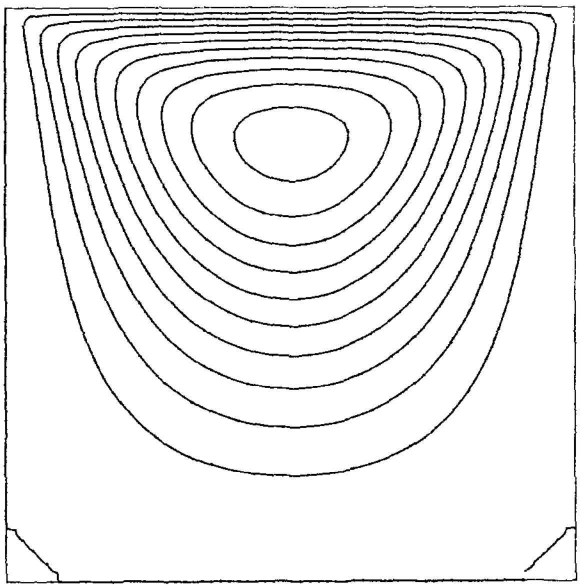

FIG. 1. Streamline pattern of driven-cavity flow at Re = 1 calculated on a 65 × 65-point uniform mesh.

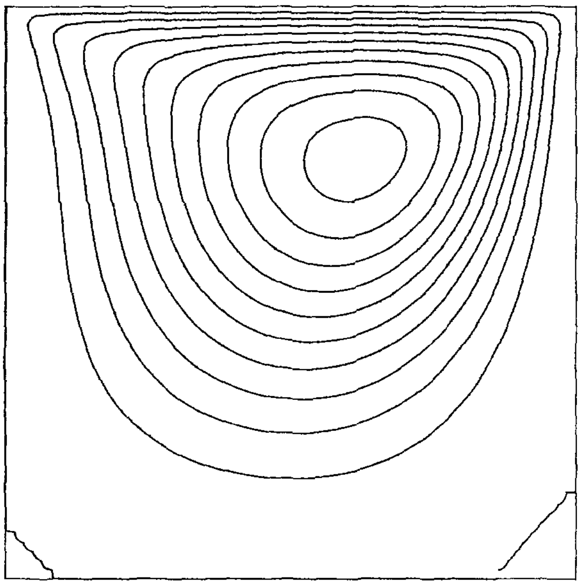

FIG. 2. Streamline pattern of driven-cavity flow at Re = 100 calculated on a 65 × 65-point uniform mesh.

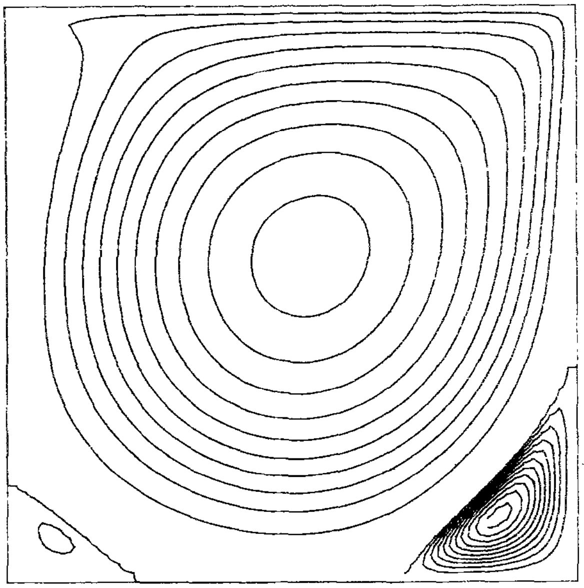

FIG. 3. Streamline pattern of driven-cavity flow at Re = 1000 calculated on a 65 × 65-point uniform mesh. For the sake of clarity, streamline spacing in the corner vortices and in the main flow has been chosen differently.

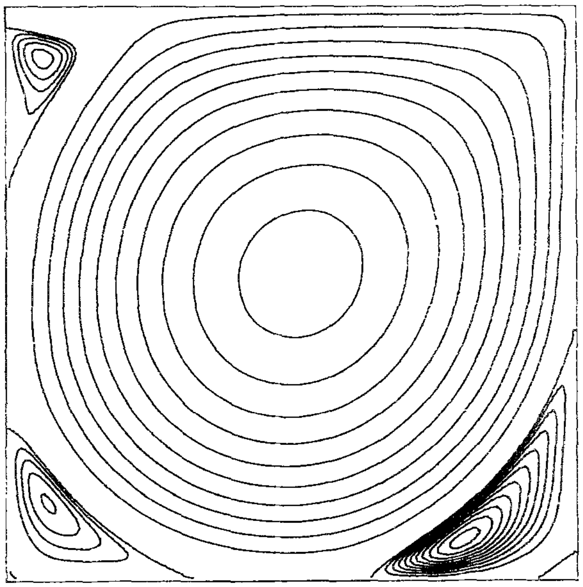

FIG. 4. Streamline pattern of driven-cavity flow at Re = 5000 calculated on an 89 × 89-point stretched mesh. Streamline spacings in the corner vortices and in the main flow are different.

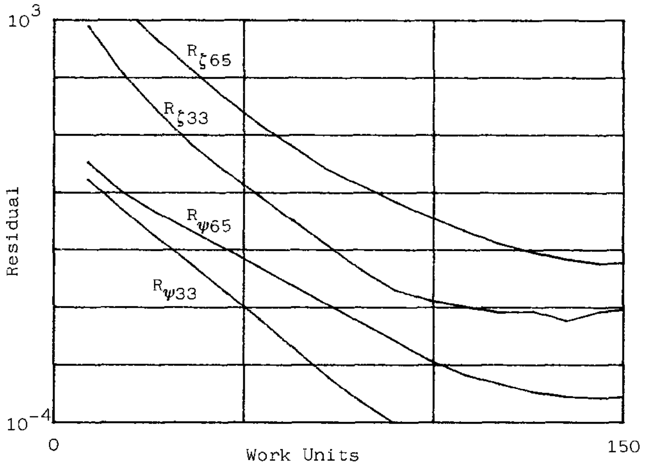

FIG. 5. Convergence history of the calculations performed at Re = 1 on 65 × 65 and 33 × 33-point uniform grids. R_ψ and R_ζ stand for the maximum-absolute-value residuals of the ψ- and ζ-equations, respectively.

starting from zero. It may be noticed that for Re = 1000 convergence is slower in the beginning part and then speeds up once a close enough approximation has been attained, so that indeed a better time would have been achieved if the calculation had been started from a previous solution at a lower Reynolds number. This has been done in the last case, Re = 5000 reported in Fig. 8, where the first rapidly oscillating part refers to the calculation initially performed by periodically increasing the Reynolds number and the regular part refers to the time when the

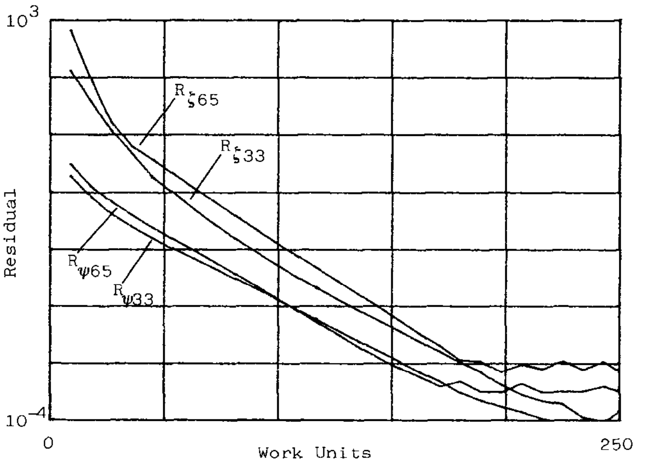

FIG. 6. Convergence history of the calculations performed at Re = 100 on 65 × 65 and 33 × 33-point uniform grids.

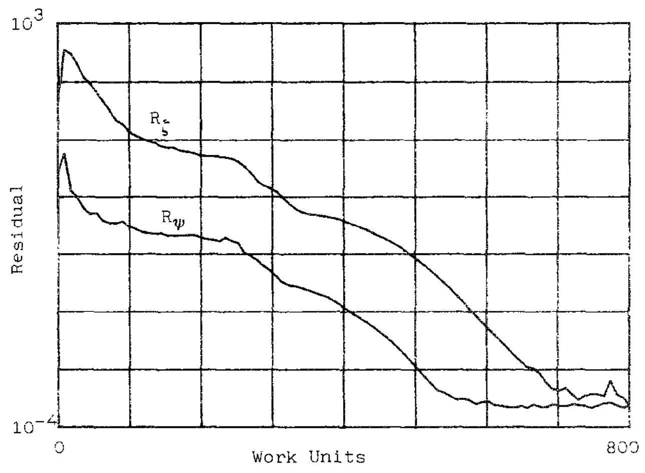

FIG. 7. Convergence history of the calculation performed at Re = 1000 on a 65 × 65-point uniform grid.

Reynolds number was eventually settled at 5000. Even in this case, however, the solution does converge starting from zero, although very slowly.

Finally, Figs. 9 and 10 report the streamline patterns obtained at Re = 1000 for the smoother alone, with no deferred correction, and the standard upwind vorticity equation (3). (The latter was obtained by simply substituting Eq. (3) for Eq. (10) in the deferred-correction procedure.) Observing how different both of these diagrams are from Fig. 3 gives a further confirmation, if one is necessary, of the inadequacy of first-order difference approximations for the resolution of Navier–Stokes problems.

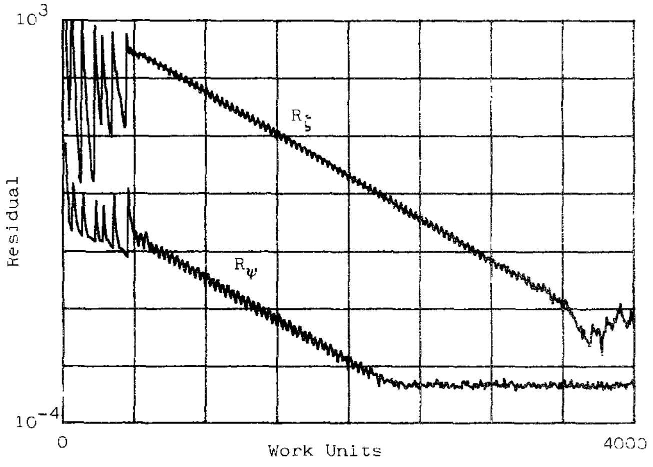

FIG. 8. Convergence history of the calculation performed at Re = 5000 on an 89 × 89 point stretched grid.

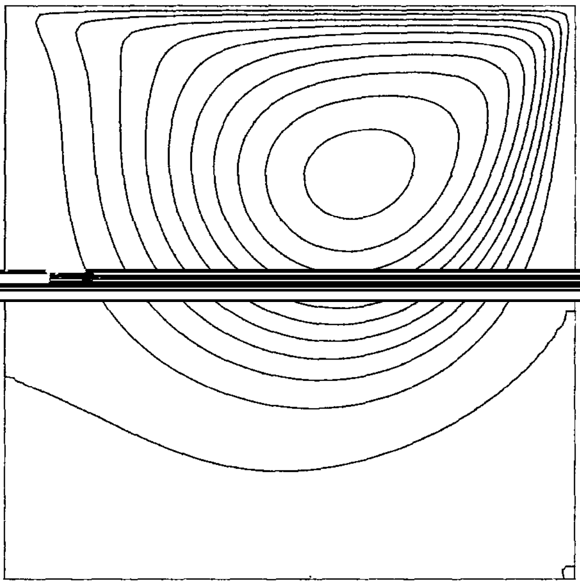

FIG. 9. Streamline pattern obtained at Re = 1000 on a 65 × 65 point mesh from the first-order smoother equations with no deferred correction added.

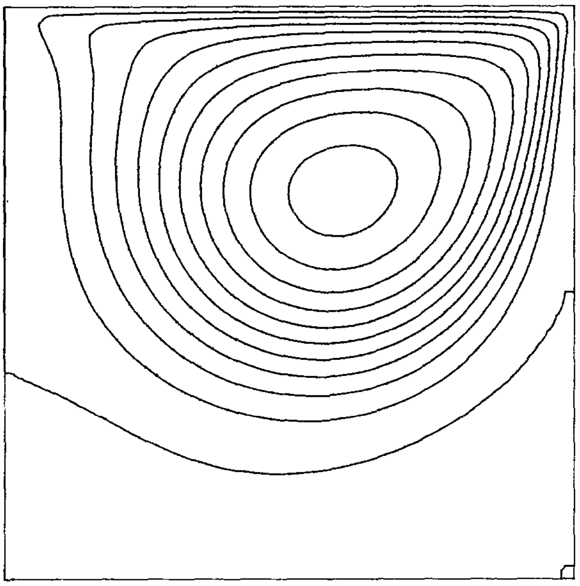

FIG. 10. Streamline pattern obtained at Re = 1000 on a 65 × 65 point grid for the standard upwind first-order discretization.

6. Conclusions

Bringing together the techniques of multigrid and deferred correction, through their common relationship with imperfect Newton iteration, yields an algorithm which combines the ease of calculation of a low-order smoother with the accuracy of a high-order difference approximation. A DCMG algorithm takes two difference formulations of the same problem, one accurate but difficult to cope with and another low-order but such that an easy and fast relaxation method is available, and by an imperfect Newton iteration process manages to solve the high-order equations using a smoother developed for the low-order ones.

The peculiarity of the DCMG algorithm is that, although the high-order equations are eventually solved, they appear only in a single residual-calculating routine which is executed once per multigrid cycle. There are two consequences: the execution speed of the algorithm is very little influenced by the complexity of the difference equations, so that high-order approximations are particularly convenient, and it is quite easy to test different approximations by simply writing the relevant equations in that single procedure.

An explicit smoother has been presented for the ψ-ζ Navier–Stokes equations which uses an appropriate kind of "upwinding" of ψ-derivatives in the vorticity-transport equation to achieve stability at relatively high Reynolds numbers. This smoother is simple and particularly well suited to the multigrid environment, and it contains no parameters, such as overrelaxation or underrelaxation factors, that must be adjusted to different values for different Reynolds numbers. The hopscotch ordering of the Gauss–Seidel relaxation procedure ensures that no spurious disuniformities of the residual distribution are introduced, thus allowing simpler interpolation formulae to be used for passing data from one to the next multigrid level.

In the application to driven-cavity flow the DCMG algorithm has proven to yield consistent results at all Reynolds numbers up to 5000. Since DCMG also has a relatively simple program structure, it may be considered a general-purpose Navier–Stokes solver.

Future applications of this method may include the study of higher than second-order formulations of the Navier–Stokes equations and the fast resolution of finite-element formulations, in addition to the resolution of other differential problems, of a physical nature unrelated to the Navier–Stokes equations, that have the common characteristic that a low-order difference approximation is unreliable but a higher-order one is hard to solve.

Acknowledgment

This research was supported by the Italian Ministry of Public Education.

REFERENCES

1. W. HACKBUSH, *Rev. Roumaine Math. Pures Appl.* **26**, 1319 (1981).
2. W. HACKBUSCH, *Multi-Grid Methods and Applications* (Springer-Verlag, Berlin, 1985), Chap. 14.
3. P. W. HEMKER, in *Multigrid Methods II*, edited by W. Hackbusch, Lecture Notes in Mathematics, Vol. 1228 (Springer-Verlag, Berlin, 1986), p. 149.
4. S. P. SPEKREIJSE, in *Multigrid Methods II*, edited by W. Hackbusch, Lecture Notes in Mathematics, Vol.1228 (Springer-Verlag, Berlin, 1986), p. 286.
5. J. C. STRIKWERDA, *J. Comput. Phys.* **47**, 303 (1982).
6. R. K. AGARWAL, AIAA Paper 81-0112, 1981 (unpublished).
7. G. D. RAITHBY AND K. E. TORRANCE, *Comput. Fluids* **2**, 191 (1974).
8. P. LUCHINI, *J. Comput. Phys.* **68**; 283 (1987).
9. P. K. KHOSLA AND S. G. RUBIN, *Comput. Fluids* **2**, 207 (1974).
10. M. NAPOLITANO, *Int. J. Num. Methods Fluids* **4**, 1101 (1984).
11. M. NAPOLITANO AND R. W. WALTERS, *AIAA J.* **24**, 770 (1986).
12. M. NAPOLITANO, *AIAA J.* **24**, 2040 (1986).
13. J. H. MORRISON AND M. NAPOLITANO, *Comput. Fluids* **16**, 119 (1988).
14. U. GHIA, K. N. GHIA, AND C. T. SHIN, *J. Comput. Phys.* **48**, 387 (1982).
15. M. C. THOMPSON AND J. H. FERZIGER, *J. Comput. Phys.* 82, 94 (1989).
16. A. BRANDT, *Math. Comput.* **31**, 333 (1977).
17. S. G. RUBIN AND P. K. KHOSLA, *Comput. Fluids* **9**, 163 (1981).
18. J. S. BRAMLEY AND D. M. SLOANE, *Comput. Fluids* **15**, 297 (1987).
19. G. LONSDALE, J. S. BRAMLEY, AND D. M. SLOANE, *J. Comput. Phys.* **78**, 1 (1988).
20. A. BRANDT AND N. DINAR, "Multigrid solution to elliptic flow problems," in *Numerical Methods in PDEs*, edited by S. V. Parter (Academic Press, New York, 1977), p. 53.
21. L. FUCHS AND H.-S. ZHAO, *Int. J. Num. Methods Fluids* **4**, 539 (1984).
22. S. SIVALOGANATHAN AND G. J. SHAW, *Int. J. Num. Methods Fluids* **8**, 417 (1988).
23. J. W. SHELDON, "Iterative Methods for the Solution of Elliptic Partial Differential Equations," in *Mathematical Methods for Digital Computers* (Wiley, New York, 1962).
24. A. R. GOURLAY, *J. Inst. Math. Appl.* **6**, 375 (1970).
25. J. P. ROACHE, *Computational Fluid Dynamics* (Hermosa, Albuquerque, NM, 1976).
26. L. C. WOODS, *Aero. Q.* **5**, 176 (1954).