

ON THE COMPUTATION OF HIGHLY VISCOUS FLOWS

C. W. Hirt and C. L. Bronisz
Flow Science, Inc.

June 1991

INTRODUCTION

FLOW-3D solves the Navier-Stokes equations with a time and spatially-dependent viscosity coefficient. The viscous stresses, however, are numerically evaluated in an explicit way. Use of an explicit viscous approximation leads to a simple solution procedure but also requires the time-step size to be kept below a certain limiting value to prevent numerical instability. For example, in a uniform mesh with cell size δx , the time-step size must satisfy the limiting condition,

$$\delta t < \frac{\delta x^2}{2d\nu}, \quad (1)$$

where ν is kinematic viscosity and d is the number of spatial dimensions (i.e., d equals 1, 2 or 3; the factor $2d$ is the number of faces of a computational cell subject to a diffusive flux).

The implications of Eq. (1) can be grasped by a simple example. Suppose we wish to compute the extrusion of a viscous material, with viscosity ν , through a circular orifice of diameter D . Let us compute the minimum number of computational time steps needed to advance the flow through the orifice a distance of one orifice diameter. If the extrusion speed is V , then the time is D/V and the number of time steps M is,

$$M = \frac{D}{V \delta t}.$$

Now suppose N mesh cells are used to resolve the diameter D . Equation (1) for two-dimensional flow then gives for the maximum time-step size,

$$\delta t = D^2 / (4\nu N^2),$$

and when combined with the previous expression we find,

$$M = 4N^2 / Re, \tag{2}$$

where $Re = DV/\nu$ is the flow Reynolds number.

This result shows that even for a very modest resolution of $N = 10$ and Reynolds number of 1.0, at least 400 time steps are required to advance the flow just one orifice diameter. For real materials, for example plastics shaped by extrusion, flow Reynolds numbers are typically orders of magnitude smaller, and Eq. (2) indicates that M will be too large for practical computations. This is the essence of the high viscosity limit of FLOW-3D: when M becomes too large for the user's patience or budget, the FLOW-3D program has not been a viable computational tool for that analysis.

Because of its ease of use and powerful capabilities, however, FLOW-3D is the program of choice for many applications. Recognizing this, more and more persons are looking to FLOW-3D as a means of solving their flow problems. A significant number of these problems involves highly viscous flows where M from Eq. (2) can be quite large. Examples are encountered in applications involving coating processes, lubrication, plastic-injection die casting and small-scale medical devices. Until now these applications could not be effectively treated by FLOW-3D.

In this note we present a modification to FLOW-3D that eliminates the numerical stability condition expressed by Eq. (1). The modification consists of making the viscous stresses implicit functions of the new-time velocity components. Since implicit stresses are not always needed, this addition has been added to the code as an option that can be selected by the user through input data (IMPVIS=1).

The implicit viscous option cannot be used in conjunction with turbulence models because those models involve other kinds of diffusive processes that are not treated implicitly. This limitation is not serious since highly viscous problems are usually laminar anyway.

Addition of implicit viscous stresses in the momentum equations means that those equations, which previously were explicit and only needed to be computed once per time step, must now be made part of an iterative solution strategy. The manner in which this is done is motivated by a simple example in the next section. In the following section the complete solution procedure is outlined, including some limitations with respect to other options in the FLOW-3D code. Finally, the new method is illustrated with some simple flow examples.

A SIMPLE IMPLICIT EQUATION

To motivate the solution procedure for adding an implicit viscous stress into our Navier-Stokes solver, let us first consider a one-dimensional, linear, heat equation for temperature T with a constant coefficient of heat conduction σ ,

$$\frac{\partial T}{\partial t} = \sigma \frac{\partial^2 T}{\partial x^2} \quad (3)$$

Difference Approximation

An unconditionally stable finite-difference approximation for this differential equation is,

$$T_j^{n+1} = T_j^n + \mu(T_{j+1}^{n+1} - 2T_j^{n+1} + T_{j-1}^{n+1}) \quad (4)$$

where

$$\mu = \frac{\sigma \delta t}{\delta x^2} \quad (5)$$

in which δx and δt are space and time increments. A superscript n denotes the time $t = n\delta t$, while a subscript j denotes space location $x = j\delta x$.

Although this difference equation could be solved directly by a tri-diagonal solver, such direct solutions are not always possible in two- and three-dimensional cases. Solutions are usually sought by iterative techniques, and that is what we shall do here.

Iteration Method

Our goal is to use an iterative method that is guaranteed to converge but which is also simple to solve. In particular, we would like a technique that looks much like the explicit solver in FLOW-3D so that we can use the existing solution algorithms in the code with only minor modifications. A scheme that suits this description quite well is the following,

$$T_j^{k+1} = T_j^n + \mu(T_{j+1}^k - 2T_j^{k+1} + T_{j-1}^k). \quad (6)$$

Here we have used a superscript k to denote the iteration level. Level $k=0$ corresponds to time level $n\delta t$. As k increases, one hopes the iteration process will lead to a steady state in which the values of T are no longer changing. When this happens, the resulting values are the desired $(n+1)\delta t$ values that solve Eq. (5). One may observe that Eq. (6) is nearly an explicit approximation (i.e, all terms on right side at level k) except for the T_j term, which has been set at the new time level. We may move this term to the left side of the equation and divide through by the resulting coefficient of T_j^{k+1} . After some rearrangement the resulting equation can be written as,

$$T_j^{k+1} = T_j^k + \left(\frac{\mu}{1+2\mu}\right)(T_{j+1}^k - 2T_j^k + T_{j-1}^k) + \left(\frac{1}{1+2\mu}\right)(T_j^n - T_j^k). \quad (7)$$

Now the equation appears to be completely explicit but differs from the original difference equation in several important ways. For instance, there is one additional term on the right side containing T_j^n .

Another important difference between Eq. (7) and the original equation is that the coefficient of the diffusion term has been divided by the quantity $1+2\mu$. This effective diffusion coefficient is easily shown to satisfy the explicit stability condition Eq. (1) for all values of δt . Therefore, Eq. (7) is unconditionally stable.

It is tempting to use one iteration of Eq. (7) (i.e., $k=1$) to estimate the new value of T . In this case the last term on the right side of the equation would vanish since the initial value for the k iteration is T^n . Unfortunately, this would lead to serious inaccuracies when μ is large because it would be equivalent to using an explicit approximation with an effective diffusion coefficient equal to $\mu/(1+2\mu)$. This reduction in the original diffusion coefficient is precisely why the difference equation is stable; it is a type of flux-limiting scheme in which the diffusive flux is automatically reduced to a value that

insures numerical stability. In fact, the effective coefficient value is the maximum possible value that remains stable for all values of δt .

We note here that the same stabilizing technique can be used in a non-uniform finite-difference grid. The effective μ in that case will depend on the local value of the grid size and will always result in the largest possible effective μ consistent with local stability.

Convergence Rate of Iteration

If the iterative solution of Eq. (4) is to be successful, the values of T_j^k must converge to steady values as k increases. Because the difference equation is linear, we can use a von Neumann Fourier analysis to study the rate of convergence. Subtracting from Eq. (6) the corresponding equation with k replaced by $k-1$ will eliminate the T^n value leaving only terms with superscripts k , $k+1$ and $k-1$. Let us now test the behavior of this equation to a typical Fourier component,

$$T_j^k = r^k e^{i\theta j}, \quad (8)$$

where $\theta = 2\pi\delta x/\lambda$ and where λ is the wave length of the Fourier component. Substituting this term into the difference equation we can solve for the amplitude r ,

$$r = \frac{2\mu}{1+2\mu} \cos\theta \quad (9)$$

Since the behavior of the Fourier mode is proportional to r^k , the magnitude of r must not exceed 1; otherwise the iteration process will diverge. Furthermore, the smaller the magnitude of r the more rapidly the Fourier mode will damp (i.e., converge) with increasing k .

According to Eq. (9) the magnitude of r will be less than or equal to 1 for all values of μ indicating unconditional stability of the difference approximations. It will equal 1 in the limit

of infinite μ when θ is equal to zero or π , implying neutral stability for these special cases (that is, no damping, hence, no steady state convergence for these values of θ). The θ equal to zero case refers to a constant value of T and is usually of no concern. The θ equal to π case, on the other hand, corresponds to a wave length of $2\delta x$, which is the shortest wave length resolvable in the mesh. This case could cause convergence problems, so an additional modification to the iteration equation is recommended.

Improved Iteration Method

A slight modification of Eq. (7) will improve its convergence characteristics and at the same time will allow for a simpler solution algorithm. The modification is to always use the most recent values (highest k level) for the temperatures that are available when evaluating the right side of the equation. For example, if the iteration proceeds from low to high values of j , then the T_{j-1} value would be at the $k+1$ level while the T_{j+1} value will still be at level k .

This simple change helps newly computed changes in T_j to be passed forward in the mesh, which intuitively should increase the rate of convergence. Furthermore, this change means that we don't have to store the level k values in order to compute the $k+1$ level quantities. Only one storage array is needed; whenever a new value of T is computed, it is placed in the T array ready to be used in computing updates to neighboring T values.

A Fourier series analysis of this modified iteration scheme leads to the amplification factor,

$$r^2 = \mu^2 / [(1+2\mu)^2 + \mu^2 - 2\mu(1+2\mu)\cos\theta] \quad (10)$$

In the limit of infinite μ the magnitude of r is given by the equation

$$r^2 = 1 / (5 - 4\cos\theta). \quad (11)$$

We see from this result that the magnitude of r is always less than one, except for the degenerate case θ equal to zero, which means that convergence of the k -iteration process should be better when using the most updated values of T on the right side of Eq. (7).

Summary of Implicit Solution Method

The one-dimensional, linear, diffusion equation, Eq. (3), is a simple partial differential equation. Nevertheless, it is important to recognize that the implicit technique discussed above will also work for more complicated equations. With this in mind it is worthwhile summarizing the most important features of the proposed implicit solution procedure.

First, we replace terms in the implicit difference equation at time level $n+1$ with approximate iteration values denoted by superscript k . The replacement is done in such a way that the k -iteration process looks like an explicit solution method. This is useful because it allows one to use algorithms for time advancement in an explicit code for the k -iteration process. The initial k value ($k=0$) corresponds to the time level n values, and the k iteration must be carried to near steady-state conditions.

Next, we modified the k levels appearing on the right side of the time-advancement equation such that all occurrences of the local variable being updated are at level $k+1$, see Eq. (6). This makes the difference equation "implicit"; however, it is a trivial implicitness since one can collect all the $k+1$ level terms on the left side of the equation and divide through by the coefficient $(1+2\mu)$, Eq. (7). It is this division, in fact, that makes the resulting equation unconditionally stable.

The division by $(1+2\mu)$ may be interpreted in several ways. For example, we may view the right side of the equation as an under-relaxation with the new value of T , multiplied by $1/(1+2\mu)$ while the old value of T , is multiplied by one minus this quantity.

Another interpretation is that the division process reduces the time-step size. That is, the time step δt has been replaced

by $\delta t/(1+2\mu)$. In a non-uniform grid the quantity μ changes with the grid-cell size, and so this can also be viewed as using a variable time step adjusted to match local stability conditions.

Finally, instead of the time-step size being reduced by the division, we could alternatively think of the division as reducing the diffusion coefficient. In this interpretation we have a flux-limiting technique where the diffusive flux has been locally reduced to preserve stability.

All three interpretations are equally valid and are mentioned here to help the reader relate the proposed method to other implicit solution techniques.

For the last step of our implicit solution method we use the most updated k values available for all quantities on the right side of the equation. This was shown to improve the convergence rate of the k -iteration process, Eq. (11).

Since each iteration is stable by construction, we could terminate the process at any k level. For example, after just one iteration we would have a numerically stable approximation to the diffusion equation. The accuracy of that approximation, however, would not be good if the magnitude of μ is one half or larger because the effective diffusion coefficient (or time-step size, etc.) is then reduced by a factor of $(1+2\mu)$, i.e., by more than a factor of two.

On the other hand, by continuing the k iteration we produce a better and better approximation to the full implicit difference equation, which provides a good solution to the original differential equation.

IMPLICIT VERSUS EXPLICIT METHODS

One might reasonably ask what we have gained when we replace an explicit approximation by one that is implicit and then solve the implicit equation by an iteration process that looks like an explicit approximation. The following comments should help clarify this matter.

Time Steps versus Iterations

To begin with, we must clearly distinguish time advancement in steps of δt , indicated by superscript n , from iteration steps used within a given time step and indicated by a superscript k . Time advancement is used to track transient phenomena. In the absence of stability considerations, the time-step size is usually selected to give good resolution of the transients. When only a steady-state result is wanted, the transients are unimportant and one would select a large time-step size in order to reach steady-state conditions more quickly. In the limit of infinite δt , for instance, all time-derivative terms would drop out of the approximation equations.

In contrast, the iteration process (within a time step) is used to obtain a solution of the implicit equations. A finite convergence criteria must be used with this iteration so that it will terminate after changes in the variables on successive iterations are below some limit. Time-step advancement, on the other hand, could continue indefinitely.

When solving for steady-state conditions, the distinction between time steps and iterations becomes somewhat blurred. For instance, when time-step sizes are too large for an accurate representation of the transients, it is probably best to think of both time steps and iterations as intermediate stages in the approach to steady state. In general, a lot of time steps with few iterations in each step is computationally similar to a few time steps with many iterations in each step. A better measure of computational effort in this case would be the total number of iterations summed over all time steps.

Uniform Mesh Comparison

To get a better perspective, let us compare an explicit approximation of Eq. (3) with the explicit approximation to the implicit equation, Eq. (7). Assuming first a uniform mesh for which the parameter μ is constant, let us use its maximum possible stable value, $\mu = 1/2$. In this case explicit and implicit approximations become, respectively,

$$T_j^{n+1} = T_j^n + \left(\frac{1}{2}\right)(T_{j+1}^n - 2T_j^n + T_{j-1}^n), \text{ explicit} \quad (12A)$$

$$T_j^{k+1} = \left(\frac{1}{2}\right)(T_j^k + T_j^n) + \left(\frac{1}{4}\right)(T_{j+1}^k - 2T_j^k + T_{j-1}^k), \text{ implicit} \quad (12B)$$

The two equations are similar except that the coefficient of the diffusion term in the implicit case is only one half that in the explicit equation. This means that the iteration method used in solving the implicit equation is acting with an effective time-step size only one half as big as that in the explicit equation. From this result we conclude that it is better to use a simple explicit method when the time-step size is within the stability range!

Now suppose we seek a steady-state solution in which δt , hence, μ , has been taken to be infinite. The explicit approximation to the implicit equation then becomes,

$$T_j^{k+1} = T_j^k + \left(\frac{1}{2}\right)(T_{j+1}^k - 2T_j^k + T_{j-1}^k) \quad (13)$$

This equation is identical to the explicit equation using the maximum stable time-step size, Eq. (12A). Recognizing this makes us repeat the earlier question: is the proposed implicit technique really any better than an explicit one operating at its stability limit?

The answer is yes for four reasons. First, the k iteration has a convergence level while the explicit equation would con-

tinue to step forward in time. We could, of course, add a convergence test to the explicit solution method for steady-state problems that would make it similar to the implicit technique.

Another reason is associated with situations in which other phenomena are being simultaneously modeled. Since the implicit method only iterates on those processes that require a small explicit time-step size, it is more efficient than an explicit method that updates everything each time step.

A third reason follows from our use of the most updated neighboring values when evaluating the right side of the iteration equation. We have seen that this generally improves the convergence rate. A similar technique should not be used with explicit methods because it leads to asymmetries and a possible loss of conservation properties. For example, conservation will not be maintained when the flux at a common boundary of two control volumes is not the same when evaluated with respect to each volume. In the implicit method, symmetry and conservation are always maintained to the level of accuracy of the specified convergence criteria.

The fourth, and final, reason is that we can easily build boundary conditions into the implicit solution method. For example, if one end of the mesh in our temperature diffusion problem is insulated, the denominator in Eq. (7) for the cell at that boundary would be $(1+\mu)$ instead of $(1+2\mu)$. For large μ values this amounts to nearly a factor of two increase in the diffusion term coefficient. Sensing boundary conditions in this way should improve the convergence rate of the implicit technique.

Non-Uniform Mesh Comparison

The situation is completely altered when the finite-difference grid is non-uniform. Consider the above case in which we use a smaller mesh cell size at one boundary (e.g., to better resolve a thin viscous or thermal boundary layer). If the cell size is reduced by a factor of 5, the time-step size in the purely explicit method must be reduced, according to Eq. (1), by a factor of 25! This clearly makes a big impact on the computational time necessary for an explicit solution.

