A Newton-GMRES Method for the Parallel Navier-Stokes Equations

J. Häuser[a], R.D. Williams[b], H.-G. Paap[c], M. Spel[d], J. Muylaert[d] and R. Winkelmann[a]

[a]Center of Logistics and Expert Systems, Salzgitter, Germany
[b]California Institute of Technology, Pasadena, California
[c]Genias GmbH., Regensburg, Germany
[d]ESTEC, Noordwijk, The Netherlands

1. INTRODUCTION

CFD is becoming increasingly sophisticated: grids define highly complex geometries, and flows are solved involving very different length and time scales. The solution of the Navier-Stokes equations must be performed on parallel systems, both for reasons of overall computing power and cost effectiveness.

Complex geometries can either be gridded by completely unstructured grids or by structured multiblock grids. In the past, unstructured grid methods almost exclusively used tetrahedral elements. As has been pointed out in [1] and recently in [2] this approach has severe disadvantages with regard to program complexity, computing time, and solution accuracy as compared to hexahedral finite volume grids. Multiblock grids that are unstructured on the block level but structured within a block provide the geometrical flexibility and retain the computational efficiency of finite difference methods.

In order to have the flow solution independent of the block topology, grids are slope continuous, and in the case of the N-S solutions an overlap of two points in each coordinate direction is provided. This causes a memory overhead: if N is the number of internal points in each direction for a given block, this overhead is the factor $(N+4)^3 / N^3$. The overhead is caused by geometrical complexity, i.e., to generate a block topology that aligns the flow with the grid as much as possible [3].

Since grid topology is determined by both the geometry and the flow physics, blocks are disparate in size, and hence, load balancing is achieved by mapping a group of blocks to a single processor. The message-passing algorithm must be able to handle efficiently the communication between blocks that reside on the same processor, meaning that there can only be one copy operation involved. Message passing (PVM or MPI) is restricted to a handful of functions that are encapsulated, and thus full portability is achieved. In addition, the code is written in ANSI-C to guarantee portability and provide significant advantages over Fortran (see for example [4]). A serial machine is treated as a one-processor parallel machine without message passing. Therefore the code, *Parnss* [5], will run on any kind of architecture. Grids generated by *GridPro* [7] or *Grid**[1] (Plot3D format) can be directly used.

Available parallelism (the maximum number of processors that can be used for a given problem) is determined by the number of points in the grid: a tool is available to

split large blocks, if necessary.

Regarding the solution algorithm for the Navier-Stokes equations, an explicit algorithm is easiest, but is not as efficient as relaxation schemes in calculating the steady state. Often relaxation schemes are used, but it should be remarked that even these methods may not converge for highly stretched grids with large aspect ratios (10^6), as is needed in most viscous flows.

Thus, we shall use implicit methods for these grids, with a linear solver chosen from the Krylov family. To make good use of these techniques, we need an effective and efficient preconditioner; this paper is about the convergence properties of the Navier-Stokes code for different preconditioners used on parallel architectures.

2. Solving the N-S Equations

An implicit step of the discretized N-S equations can be cast in the form of a set of nonlinear equations for the flow variables, whose numerical solution is by a Newton scheme. The computational kernel is a linear solve with the matrix being the Jacobian of the equations. There are numerous schemes for such solves, such as Jacobi relaxation or lower-upper triangular split (Gauss-Seidel). As mentioned above, these schemes are slow to converge for a stiff system, caused by widely varying temporal and spatial scales.

The numerical solution proceeds in five major stages: in this work, stage (4) will be described in some detail.

1. *Topology:* Perform domain decomposition of the solution domain.
2. *Grid generation:* Create a high-quality grid within each domain. Spatial discretization reduces the N-S equations to a set of ODE's.
3. *Explicit Solution:* Advance explicitly in time by using a two step Runge-Kutta scheme.
4. *Implicit Solution:* Advance the solution implicitly in time with the backward Euler scheme, thus requiring solution of nonlinear equations, which can be solved by a Newton or quasi-Newton algorithm, which in turn requires solving sets of linear equations: we use preconditioned GMRES for these.
5. *Root Polishing:* For steady state solution, use a Newton iteration to derive the steady state, which is equivalent to an implicit step with infinite time step.

Investigations are underway to determine if it is possible to relax the accuracy with which the nonlinear equations in (4) are solved, yet still obtain robust and accurate convergence through stage (5).

2.1. Krylov Subspace Methods

In what follows, we describe the CG method, because it is the basis for the *Generalized Minimal Residual (GMRES)* technique, as used in this paper, and for example, in [10]. However, the CG method will be presented mainly from a geometrical point of view to provide the motivation and insight in the workings of the method.

2.2. Conjugate Gradient

We have a system of linear equations, $\mathbf{Ax} = \mathbf{b}$, derived from an implicit step of the N-S equations, together with an initial solution vector $\mathbf{x}^0$. This initial vector may be obtained by an explicit step, or simply may be the flow field from the previous step.

We can write this linear system as the result of minimizing the quadratic function

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{x}^T\mathbf{b}$$

where the gradient of f is **Ax** - **b**. In the CG method, a succession of search directions $\mathbf{p}^m$ is employed — how these directions are constructed is of no concern at the moment — and a parameter α_m is computed such that $f(\mathbf{x}^m - \alpha_m\mathbf{p}^m)$ is minimized along the $\mathbf{p}^m$ direction. Upon setting $\mathbf{x}^{m+1}$ equal to $\mathbf{x}^m - \alpha_m\mathbf{p}^m$, the new search direction is then to be found: the construction of the search directions can be directly seen

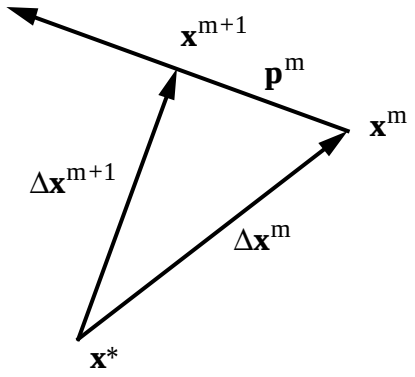

Figure 1. The CG method: one-dimensional minimization. Let $\mathbf{x}^*$ denote the exact (unknown) solution, $\mathbf{x}^m$ an approximate solution, and $\Delta\mathbf{x}^m = \mathbf{x}^m - \mathbf{x}^*$. Given any search direction $\mathbf{p}^m$, the minimal distance from the line to $\mathbf{x}^*$ is found by constructing $\Delta\mathbf{x}^{m+1}$ perpendicular to $\mathbf{p}^m$. Since the exact solution is (of course) unknown, we make the residual perpendicular to $\mathbf{p}^m$. Regardless how the new search direction is chosen, the norm of the residual is not increasing.

from Fig. 2.

In two dimensions, the contours $f(\mathbf{x})$=const form a set of concentric ellipses whose common center is the minimum of $f(\mathbf{x})$. It can be shown that the residual vectors $\mathbf{r}^m$ form an orthogonal system and that the search vectors $\mathbf{p}^m$ are mutually **A**-orthogonal. The CG method has the major advantage that only short recurrences are needed, that is, the new vector $\mathbf{x}^m$ depends only on $\mathbf{x}^{m-1}$ and search direction $\mathbf{p}^m$. In other words, storage requirements are low.

The number of iterations of *CG* needed to achieve a prescribed accuracy is proportional to the square root of the *condition number* κ of the matrix, which is defined as the ratio of the highest to the lowest eigenvalue. Note that for second-order elliptic problems, κ increases by a factor of four when the grid-spacing is halved.

2.3. GMRES

The matrix obtained from the N-S equations is neither symmetric nor positive definite. If we use the CG method, the term $(\mathbf{p}^m, \mathbf{A}\mathbf{p}^m)$ is not guaranteed to be positive, and the search vectors are not mutually orthogonal. It should be remembered that $\mathbf{p}^{m+1} = \mathbf{r}^m + \alpha_m\mathbf{p}^m$ and that the α_m are determined such that the second orthogonality

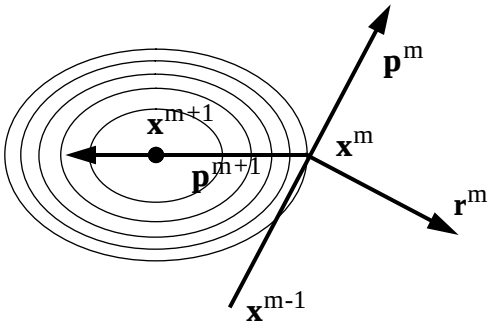

Figure 2. The CG method: Search directions. The next search direction $\mathbf{p}^{m+1}$ is **A**-orthogonal, or conjugate, to $\mathbf{p}^m$, and it is a linear combination of $\mathbf{r}^m$ and $\mathbf{p}^m$. This determines $\mathbf{p}^{m+1}$. In two dimensions, this second search direction goes through the midpoint of the ellipse, giving an exact solve at the second stage. Note that simply setting $\mathbf{p}^{m+1} = -\mathbf{r}^m$, the method of steepest descent, would not result in the most efficient search direction.

condition holds, but this is no longer possible for the unsymmetric case. However, this feature is mandatory to generate a basis of the solution space. Hence, this basis must be explicitly constructed. The extension of the CG method, termed *GMRES* (Generalized Minimized Residual), minimizes the norm of the residual in a subspace spanned by the set of vectors $\mathbf{r}^0$, $\mathbf{A}\mathbf{r}^0$, $\mathbf{A}^2\mathbf{r}^0$, ..., $\mathbf{A}^{m-1}\mathbf{r}^0$, where vector $\mathbf{r}^0$ is the initial residual, and the *m*-th approximation to the solution is chosen from this space. The above mentioned subspace, a *Krylov space*, is made orthogonal by the well-known *Gram-Schmidt* procedure, known as an *Arnoldi* process when applied to a *Krylov* subspace.

When a new vector is added to the space (multiplying by **A**), it is projected onto all other basis vectors and made orthogonal with the others. Normalizing it and storing its norm in entry $h_{m,m-1}$, a matrix $\mathbf{H}_m$ is formed with nonzero entries on and above the main diagonal as well as in the subdiagonal. Inserting the ansatz for $\mathbf{x}^m$ into the residual equation, and after performing some modifications, a linear system of equations for the unknown coefficients γ_l^m involving matrix $\mathbf{H}_m$ is obtained. $\mathbf{H}_m$ is called an upper *Hessenberg* matrix. To annihilate the subdiagonal elements, a 2D rotation (*Givens rotation*) is performed for each column of $\mathbf{H}_m$ until $h_{m,m-1} = 0$. A Givens rotation is a simple 2×2 rotation matrix. An upper triangular matrix $\mathbf{R}_m$ remains, which can be solved by backsubstitution.

3. Preconditioners

In order to reduce the condition number of **A**, the system is premultiplied by a so-called preconditioning matrix **P** that is an approximation to $\mathbf{A}^{-1}$, but is easy to compute. Instead of solving the sparse linear system $\mathbf{Ax}=\mathbf{b}$, the equivalent system $\mathbf{(PA)x}=\mathbf{Pb}$ is solved. The choice of an effective (less iterations) and an efficient (less computer time) preconditioner is crucial to the success of *GMRES*.

For the preconditioning to be *effective*, **P** should be a good approximation of $\mathbf{A}^{-1}$, so that the iterative methods will converge fast. For *efficiency*, the memory overhead and

the additional cost per iteration should be small.

Any kind of iterative scheme can be used as a preconditioner, for instance, Jacobi or Gauss-Seidel relaxation, Successive Overrelaxation, Symmetric Successive Overrelaxation (SSOR), Red-Black or Line Gauss-Seidel Relaxation, Incomplete Lower-Upper Factorization (ILU). Thus, the linear solver is a two-part process, with an inner loop of a simple iterative scheme serving as the preconditioner for an outer loop of *GMRES*.

3.1. Preconditioners in *Parnss*

First, we recall that the condition number, and hence the number of sweeps to convergence, of a grid increases dramatically as the grid is made finer. Therefore, we expect a good strategy is to use multiple grids at different resolutions, the coarser acting as a preconditioner for the finer.

In the following, however, we describe the various preconditioning matrices that have been constructed and implemented for use with the *Parnss* code on a fixed grid. In each case, the preconditioner is made by repeating a simple iteration derived from a splitting of **A**:

$$\mathbf{x}^{k+1} \leftarrow \mathbf{B}^{-1} [(\mathbf{B} - \mathbf{A}) \mathbf{x}^{k} + \mathbf{b}]$$

The various forms of matrix **B** that have been implemented are based on splitting **A** into diagonal, lower-triangular, and upper-triangular parts, **D**, **L**, and **U**, respectively. They may be written:

- Diagonal: $\mathbf{B} = \mathbf{D}$
- Gauss-Seidel: $\mathbf{B} = \mathbf{D} - \mathbf{L}$
- Successive Overrelaxation: $\mathbf{B} = \mathbf{D}/\omega - \mathbf{L}$
- Symmetric Successive Overrelaxation: $\mathbf{B}_1 = \mathbf{D}/\omega - \mathbf{L}$ alternating with $\mathbf{B}_2 = \mathbf{D}/\omega - \mathbf{U}$

In contrast to the CG method, GMRES does not generate short recurrences, but needs the full set of basis vectors, thereby incurring a substantial memory overhead. Second, the computational cost increases linearly with the number of basis vectors. Since the code is written in ANSI-C, memory is allocated and freed dynamically. Moreover, only one block at a time is computed per processor, so that only this block must be stored. In principal, the Krylov basis could be made as large as the available memory allows, dynamically adding new basis vectors. However, because of the computational overhead and the effects of rounding error in the orthogonalization, the algorithm should be restarted: we have chosen to do this after 20 iterations.

3.2. Experimental Results

We conclude this section with some measurements of the effectiveness of the various preconditioners. Figure 3 shows the fall of the norm of the residual for a sample calculation against wall-clock time. The block-diagonal preconditioning is a variant of diagonal, where 5×5 diagonal blocks of the matrix are inverted; red-black preconditioning is a Gauss-Seidel process where the grid points have been ordered in a particular way. It is clear that the SSOR is the most efficient.

4. Results for NASA-ESA Huygens Space Probe

Parnss was used to perform several testcase computations for the Huygens space

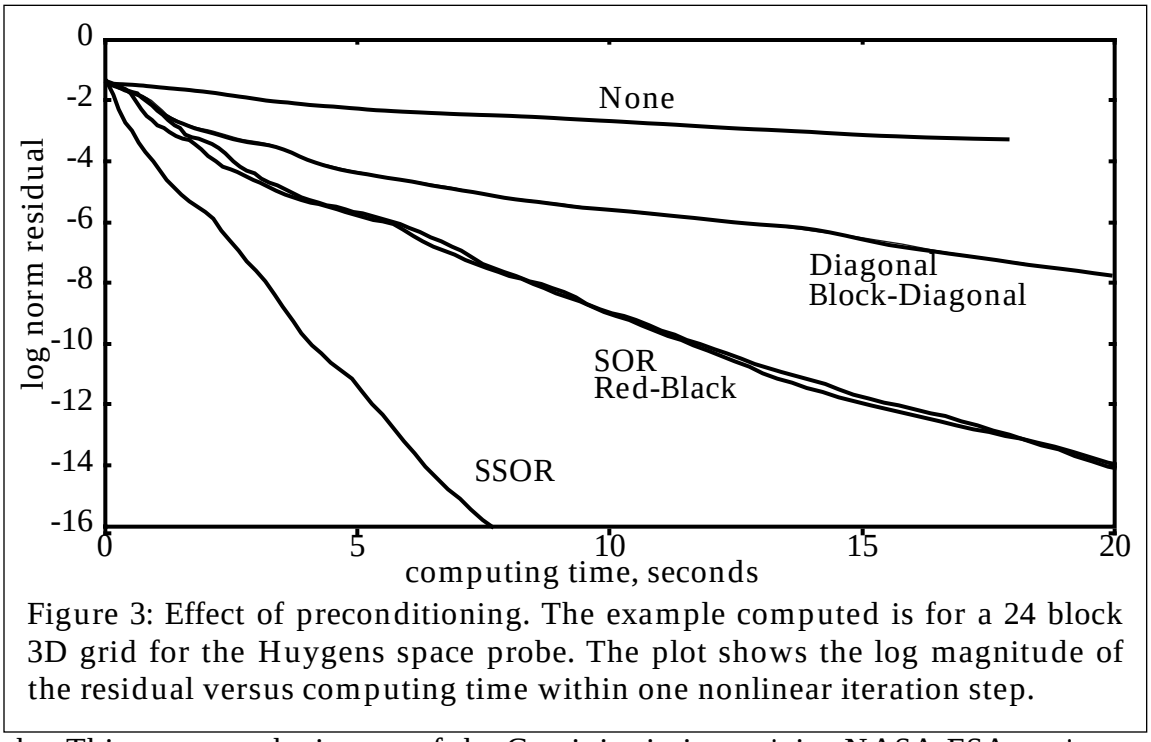

Figure 3: Effect of preconditioning. The example computed is for a 24 block 3D grid for the Huygens space probe. The plot shows the log magnitude of the residual versus computing time within one nonlinear iteration step.

probe. This space probe is part of the Cassini mission, a joint NASA-ESA project and will be launched in 1996. After a six-year flight, Cassini will have reached Saturn, and the Huygens probe will make a landing on Titan, the largest moon. Upon completing the entry phase, it will descend by parachute (Mach 0.1) to measure the composition of Titan's atmosphere (mainly nitrogen). The concern is that sensors (lasers, mass spectrometer) located on the windward side of the probe may become inoperational if the local flow field is such that dust particles may be convected onto the optical surfaces of these sensors. So far, all computations have been for inviscid flow, but high cell aspect ratios were already used. In addition to the incompressible case, a Mach 3.1 computation has been performed. Computations were stopped when the residual had dropped to 5×10^{-6}. The computation for Ma 3.1 is not a realistic case, since the aerobraking shield was not modeled. For both low and high Mach numbers, the SSOR performed best.

5. Towards an Efficient Parallel Solution Strategy

In this paper we have briefly described a combined solution strategy for the N-S equations. Solutions for steady state as well for transient flow can be computed, covering the range from incompressible to hypersonic flows. The numerical solution technique is based on a Krylov subspace method. In particular, for very large and stiff problems the condition number will also be large, and preconditioning is essential. For these problems, conventional relaxation schemes converge very slowly or may not converge at all. A large number of preconditioners have been tested, and symmetric SOR has been found to work best. An excellent discussion of preconditioners for 2D test cases is given in [11].

From theoretical models [12] and computational experience [5] it can be concluded that load balancing for complex geometries and parallel efficiency up to several hun-

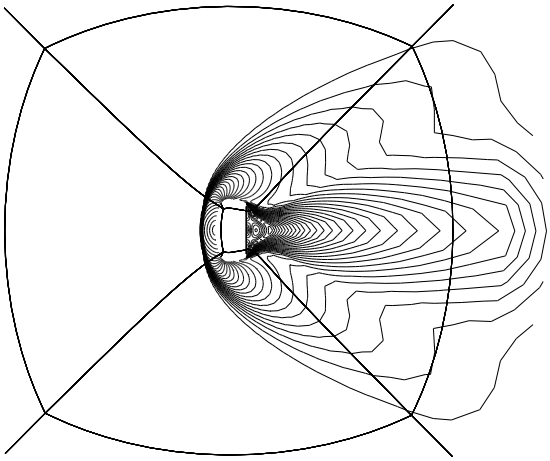

Figure 4: Mach distribution for a 6 block Huygens grid with AoA=0, Ma=3.1, and blocksize 19×61×19.

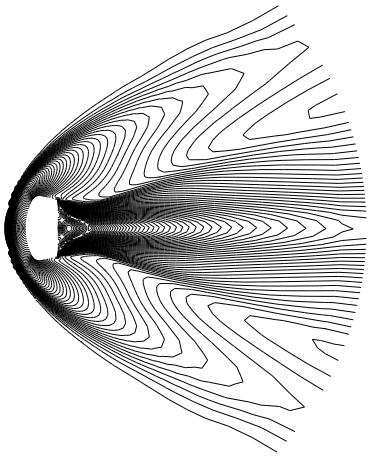

Figure 5: The same problem as in Fig. 4, but now the solution domain is split into 384 blocks of size 12×12×12, resulting in a total of 663,552 grid points. Although block size is small, communication overhead is not important because of the large amount of computation per grid point.

dred processors do not pose a problem for the N-S simulations for the strategy outlined in this paper.

However, there is a need to investigate the convergence rate of these implicit solution schemes on serial and parallel architectures. The time-dependent N-S equations used to obtain the steady state are hyperbolic, that is, there is a finite propagation speed. A full numerical coupling of all linear equations is therefore unphysical, and actually reduces the convergence rate for steady state problems, as large scale computations have shown. The 384-block Huygens space probe that was presented in Section 4 can also be modeled using a 6-block grid. For exactly the same problem, a much

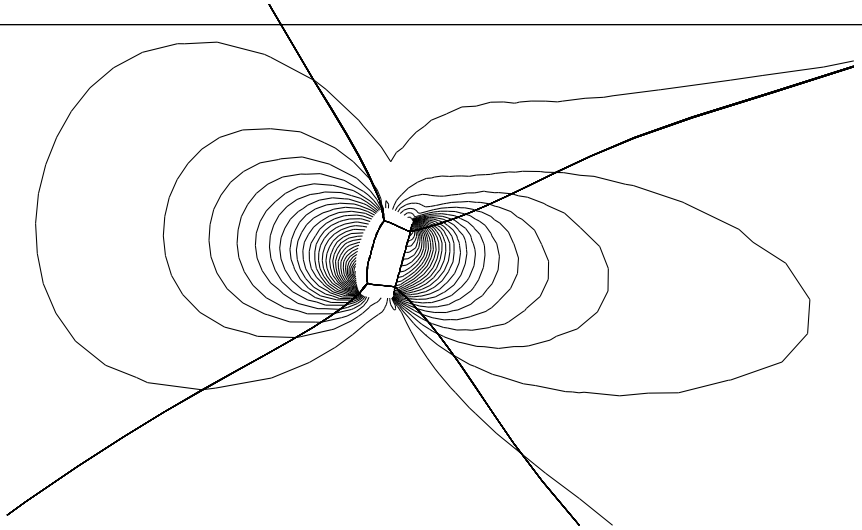

Figure 6: The Mach distribution for the 6-block grid with AoA=15, Ma=0.1 and blocksize 19×19×19.

slower convergence to the steady state has been observed, exhibiting the same trend as in the 2D testcases [5]. Since the exact numbers have not been firmly established at the time of this writing, they will not be presented here.

6. REFERENCES

1. J. Häuser, J. Muylaert, H.-G. Paap, M. Spel, and P.R. Eiseman, Grid Generation for Spaceplanes, 3rd Space Course, University of Stuttgart, Germany, 1995, 66 pp.
2. D. J. Mavriplis and V. Venkatakrishnan, A Unified Multigrid Solver for the Navier-Stokes Equations on Mixed Element Meshes, AIAA-95-1666.
3. D. A. Kontinos, McRae, Rotated Upwind Algorithms for Solution of the Two- and Three-Dimensional Euler and Navier-Stokes Equations, AIAA 94-2291
4. J. Häuser, M. Spel. J. Muylaert and R. D. Williams, *Parnss:* An Efficient Parallel Navier-Stokes Solver for Complex Geometries, AIAA 94-2263.
5. J. Häuser, J. Muylaert, M. Spel, R. D. Williams and H.-G. Paap, Results for the Navier-Stokes Solver *Parnss* on Workstation Clusters and IBM SP1 Using PVM, in Computational Fluid Dynamics, Eds. S. Wagner et al., Wiley, pp. 432-442.
6. M. J. Bockelie and P.R. Eiseman, A Time Accurate Adaptive Grid Method for the Numerical Simulation of a Shock-Vortex Interaction, NASA-2998, 1990.
7. P. R. Eiseman, et al., *GridPro/az 3000*, Users's Guide and Reference Manual, 111 pp., Program Development Corporation of Scarsdale, NY.
8. D. Whitfield, Newton-Relaxation Schemes for Nonlinear Hyperbolic Systems, Mississippi State University Preprint MSSU-EIRS-ASE-90-3, 1990.
9. K. J. Vanden, Direct and Iterative Algorithms for the Three-Dimensional Euler-Equations, Dissertation Thesis, Mississippi State University, December 1992.
10. G. Golub, G., J.M. Ortega, Scientific Computing, Academic Press, 1993.
11. K. Ajmani, W. F. Ng, M. S. Liou, Preconditioned Conjugate-Gradient Methods for the Navier-Stokes Equations, J. of Comp. Phys., **110** (1994) 68-81.
12. J. Häuser, and R. D. Williams, Strategies for Parallelizing a Navier-Stokes Code on the Intel Touchstone Machines, Int. J. Num. Meth. Fluids, **15** (1992) 51-58.