Practical Aspects of Krylov-Subspace Iterative Methods In CFD

Thomas H. Pulliam

Advanced Computational Methods Branch

Stuart Rogers

Design Cycle Technologies Branch

Timothy J. Barth

Advanced Computational Methods Branch

NASA Ames Research Center Moffett Field, CA 94035-1000, USA.

AGARD 77th Fluid Dynamics Panel Symposium on Progress and Challenges in CFD Methods and Algorithms

Seville, Spain, 2-5 October 1995

Practical Aspects of Krylov subspace Iterative Methods In CFD

Thomas H. Pulliam,*
Advanced Computational Methods Branch

Stuart Rogers †
Design Cycle Technologies Branch

Timothy Barth ‡
Advanced Computational Methods Branch

NASA Ames Research Center Moffett Field, CA 94035-1000, USA.

September 19, 1995

Abstract

Implementation issues associated with the application of Krylov subspace iterative methods, such as Newton-GMRES, are presented within the framework of practical CFD applications. This paper will categorize, evaluate and contrast the major ingredients (function evaluations, matrix-vector products and preconditioners) of Newton-GMRES Krylov subspace methods in terms of their effect on the local linear and global nonlinear convergence, memory requirements, and accuracy, The discussion will focus on Newton-GMRES in both a structured multi-zone incompressible Navier-Stokes solver and an unstructured mesh finite-volume Navier-Stokes solver. Approximate vs. exact matrix-vector products, effective preconditioners and other pertinent issues will be addressed.

1 Introduction

Interest in iterative methods in CFD has been motivated not only by the requirement for better convergence and speed of numerical codes, but also by the availability of faster, larger memory serial and parallel machines. The coupling of Newton's method with iterative solvers is an effective approach for solving the large systems of nonlinear equations which arise from discretized forms of the Euler and Navier-Stokes equations. One of the main motivations for the use of Newton's method is the possibility of superlinear (and in some cases quadratic) asymptotic convergence. References [7, 21] are examples of successful implementations of exact Newton's method for two-dimensional Navier-Stokes codes. Most of the conventional implicit schemes used today are effectively approximate-Newton methods. The approximations appear in the form of simplifications in the functional Jacobian or some form of under/over relaxation strategy, see e.g. [11] or [15]. In practice these simplifications are employed for reasons such as efficiency, implementation ease, or non-analyticity of operators (e.g., discrete limiters in differencing schemes based on Riemann solvers). Over a wide range of numerical methods developed for the solution of the multidimensional Navier-Stokes equations, the rigorous application of Newton's method would require the inversion of a large block banded matrix, which even by today's standards, poses many obstacles in terms of memory requirements and speed. An alternative to direct matrix inversion is the use of iterative matrix solution methods. In particular, the class of Krylov subspace methods known as GMRES [19] will be considered. Wigton [23] was the first to successfully implement GMRES for a two-dimensional Navier-Stokes code.

The difficulties associated with iterative methods such as GMRES lie in the rapid expansion of memory requirements inherent in the embedded Arnoldi process (storing the Krylov subspace vectors), the need to perform the matrix vector "Ap" products (which sometimes requires the storage of the matrix A), and the preconditioning of the system of equations by some approximate inverse of A to improve the convergence of GMRES.

The purpose of this paper is to focus on the implementation details and specific results from the application of Newton-GMRES to both a structured multi-zone incompressible Navier-Stokes code (INS2D, Rogers [17]) and an unstructured mesh Navier-Stokes code due to Barth [5]. We use these two codes as case studies, but the lessons learned, approximations assumed, results obtained, and general conclusion, are applicable to most implementations of Newton-GMRES to systems of PDE's.

2 General Formulation

In a general form, we cast the discrete approximation to the steady multi-dimensional Navier-Stokes equations as

$$\begin{aligned}\mathcal{R}(\mathbf{Q}) &= \mathcal{R}(\cdots, Q_{j,k-1}, Q_{j-1,k},\\ &\quad Q_{j,k}, Q_{j+1,k}, Q_{j,k-1}, \cdots) = 0, \end{aligned} \quad (1)$$

where we are representing the support of the operator as involving neighboring points in a computational mesh and $\mathbf{Q}$ is the solution variable (typically the conserved variables). Although this representation appears in a structured mesh form and is rather compact (involving only three points in each computational direction), we intend it to also represent an unstructured mesh template and possibly higher order - higher dimensional - more broadband support. We shall refer to this as the **Function Evaluation** step in the overall process. For fixed point solutions (steady-state) we require the solution of $\mathcal{R}(\mathbf{Q}) = 0$.

A time accurate approach to the solution assumes the form of

$$\frac{\partial \mathbf{Q}}{\partial t} + \mathcal{R}(\mathbf{Q}) = 0 \quad (2)$$

which can either represent the artificial-compressibility scheme for the incompressible Navier-Stokes equations [17] or the full Navier-Stokes unstructured mesh scheme [5]. Applying implicit Euler time differencing with the usual Taylor series linearization in time we have

$$\left[\frac{D}{\Delta t} - \frac{\partial \mathcal{R}(\mathbf{Q})^n}{\partial \mathbf{Q}}\right] \Delta \mathbf{Q} = \mathcal{R}(\mathbf{Q})^n \quad (3)$$

with $\Delta \mathbf{Q} = \mathbf{Q}^{n+1} - \mathbf{Q}^n$, $\frac{\partial \mathcal{R}(\mathbf{Q})^n}{\partial \mathbf{Q}}$ the Jacobian ($\mathcal{A}$) of the **Function Evaluation** , $\mathcal{R}(\mathbf{Q})$, and D a positive diagonal matrix. For $\Delta t \to \infty$ this is exactly Newton's method and for finite Δt a relaxed form of Newton's method. In many applications of Newton-like methods to the Euler and Navier-Stokes equations this time-like relaxation is used to start the solution process. A finite time step Δt is used initally to get past the rather violent nonlinear startup and then increased to $\Delta t \to \infty$ leading to rapid linear, super-linear or quadratic convergence depending on the characteristics of the Newton solver, e.g.[21] or [5]. It is convenient to recast Eq. 3 in the general form

$$\mathbf{b} - \mathcal{A}\mathbf{x} = 0, \quad (4)$$

where $\mathbf{b} = \mathcal{R}(\mathbf{Q})$, $\mathbf{x} = \Delta \mathbf{Q}$ and $\mathcal{A}$ is a matrix operator.

The numerical process involved in solving Eq' 3 will be referred to here as the **Inner Iteration** at a particular step n. The overall iteration of the nonlinear system will be referred to as the **Outer Iteration** . There are a number of successful approaches to the **Inner Iteration**. In the case of structured mesh applications, $\mathcal{A}$ represents a sparse block banded matrix which can be solved with various methods such as point or line relaxation [17] or approximate factorization, e.g.[14]. In the unstructured mesh case, $\mathcal{A}$ may not have a simple underlying structure, but the **Inner Iteration** can be successfully solved with a wide variety of relaxation techniques [22], [5]. For the present discussion we shall focus on the GMRES Krylov projection technique for solving the **Inner Iteration**.

The GMRES (Generalized Minimal RESidual) method was introduced by Saad and Schultz [19] for solving large sparse systems of linear equations. The GMRES algorithm is a Krylov subspace method where given a matrix $\mathcal{A} \in \Re^{N \times N}$, a vector $v \in \Re^N$ and an integer $m \geq 1$, the Krylov subspace associated with $\mathcal{A}$, v and m is defined as

$$K_m(\mathcal{A}, v) = span\{v, \mathcal{A}v, \mathcal{A}^2 v, \cdots, \mathcal{A}^{m-1} v\}. \quad (5)$$

In the GMRES algorithm an initial guess x_0 to the solution of the linear system is given from which the initial residual is defined

$$r_0 = \mathbf{b} - \mathcal{A}\mathbf{x}_0. \quad (6)$$

The GMRES method then attempts to find $z_m \in K_m(\mathcal{A}, r_0)$ such that the residual vector $\mathbf{b} - \mathcal{A}(\mathbf{x}_0 + z_m)$ is small . This is done so that at each iteration the residual norm is minimized. One important parameter for the GMRES method is the size of the subspace m. As m increases, the memory increases linearly and the computation quadratically. The parameter m is usually chosen based on storage requirements and effectiveness of the **Inner Iteration**. In the discussion below, we will have more specific things to say about this requirement and it's effect on the overall process. To avoid the increasing memory and computation requirements with increasing m, a common modification of GMRES is to apply restarts. An upper bound m_r on m is chosen and if convergence is not reached, the Krylov subspace process is restarted with the current residual r_{m_r} replacing r_0. In this case, the memory requirements are traded off against the convergence of the **Inner Iteration** process, and this will definitely affect the **Outer Iteration** .

Numerical experience has shown that success or failure of GMRES hinges critically on adequate preconditioning of the linear systems to be solved. A preconditioning matrix M is usually applied in either a left-preconditioning $M(\mathbf{b} - \mathcal{A}\mathbf{x}) = 0$ or right-preconditioning $\mathbf{b} - \mathcal{A}M\mathbf{y} = 0$ (with $\mathbf{y} = M^{-1}\mathbf{x}$) fashion. Ideally M should be chosen to be an approximation to $\mathcal{A}^{-1}$. Although, the most successful and popular form of M appears to be ILU [13] (Incomplete Lower-Upper Factorization), we will also consider alternate preconditioners in the next section.

The important ingredients of the Newton-GMRES method which we will focus on in this paper are the Ap products required to form the Krylov subspace vectors $K_m(A, v)$, the choice of the preconditioner M, the size of the subspace m and restart size m_r, and the storage requirement influenced by all these factors. We will attempt to put the various trade offs in terms of memory requirements, convergence and efficiency in perspective, (in particular for the two approaches discussed here, but also in general).

3 Structured Mesh Incompressible Navier-Stokes

Rogers [17] has implemented the Newton-GMRES algorithm into a two-dimensional incompressible Navier-Stokes code (INS2D) and has made some significant comparisons with the conventional techniques of implicit point and implicit Gauss-Seidel line relaxation. The INS2D flow code [18] solves the Reynolds-averaged incompressible Navier-Stokes equations using the method of artificial compressibility, [9]. It is capable of handling multiple-zone structured grids using either a patched multi-block (pointwise continuous) interface, or an overlaid (chimera) interface between zones. The boundary conditions at the physical boundaries and at zonal boundaries are applied in an implicit fashion during the solution process. A third-order, upwind-differencing scheme based on the method of Roe [16] is used to descritized the convective terms, and the viscous terms are differenced using second-order central differences. The system of equations is integrated in pseudo-time using an implicit Euler time discretization. Typically, the time step is set to infinity (10^8) which results in a Newton's method approach where the implicit point or line relaxation schemes are used for the **Inner Iteration** or more specific to this paper, GMRES is used for the **Inner Iteration**.

In the current implementation, the Jacobian $\mathcal{A}$ is formed based on a first-order differencing of the convective terms, whereas third-order differencing is used for $\mathcal{R}(\mathbf{Q})$. In addition, approximate Jacobians of the Roe flux differences from the upwind-differencing scheme are used in the definition of $\mathcal{A}$, see [1] for more details. The first-order difference operator is used to reduce the bandwidth of the resulting $\mathcal{A}$ matrix, which has lower memory and computational requirements for the solution of Eq. 3. However, this use of approximate Jacobians can also slow the convergence to a steady state, that is, the **Outer Iteration** nonlinear Newton process is affected.

The GMRES implementation is preconditioned using block ILU(0) [13] and the matrix $\mathcal{A}$ is stored so that Ap products can be efficiently formed and the ILU process streamlined. For comparison, block point relaxation and block line relaxation are used as both the **Inner Iteration** solver and as preconditioners for the GMRES **Inner Iteration** process. Including a subspace size typically on the order of $m = 10$ leads to additional storage requirements as discussed below which are somewhat of a burden in two dimensions and would be a significant hindrance in three dimensions. The use of the approximate Jacobian (due to the first order form and the linearization errors associated with the Roe solver) produces an approximate Newton's method and therefore linear convergence is realized as opposed to the potential for quadratic convergence.

Rogers [17] examines a wide range of cases and options in his paper on the Newton-GMRES implementation. Table 1 shows the characteristics of the cases presented and itemizes the costs of the various schemes for each case broken down by the fundamental steps in the algorithm. Base memory (B MW) includes all overhead storage for the algorithm including memory for either L (line relaxation), P (point relaxation) or the Ap product in G (GMRES) ($\approx$ 76 words/point). The additional memory (A MW) is composed of subspace size ($\approx 3 \times (m + 4)$ words/point) and preconditioner ($\approx$ 9 words/point) contributions for GMRES. The timings are in ms/pt : milliseconds/point to convergence, (maximum divergence $\approx 10^{-8}$). The standard approaches of point relaxation and line relaxation are compared directly with the Newton-GMRES scheme and are also assessed as preconditioners for Newton-GMRES. The first few cases are for a NACA 4412 airfoil an angle of attack $\alpha = 13.87°$ and a Reynolds number, $Re = 1.5 \times 10^6$ and are computed on a set of refined grids. The multi-element case is a three element airfoil at $\alpha = 8°$ and $Re = 9 \times 10^6$, a schematic of the grid system is shown in Figure 1.

Figures 2, 3, 4, 5 show comparisons for the above cases, where the symbols represent every 50 **Outer Iteration** . It seems obvious from these results that the GMRES-ILU combination is the more efficient in terms of computation time. On the other hand, the negative aspect of the GMRES-ILU combination is the memory requirements. By examining the trade offs between CPU time to convergence and memory requirements optimal choices can be made.

For example, Figure 2 shows the effect on CPU time to convergence for various choices of m. A subspace size $m = 10$ seems to be optimal in terms of computational costs, including reasonable added memory requirements. Also, note that for the converging cases of $m = 10, 20, 40$, it required 50 **Outer Iteration** to reach the same level of convergence (CPU times are larger reflecting the added computational costs of a larger subspace size). This is not surprising since the inexact Jacobian used in this scheme limits the **Inner Iteration** process to linear convergence. Therefore, after some point, it does not pay to converge the **Inner Iteration** past some tolerance level without incurring additional cost in terms of CPU time and memory.

Case	Method	B MW	A MW	ms/pt
Airfoil Grid 1 119x31	L(5)	0.28	0.006	1.98
	P(20)	0.28	0.001	2.31
	G(10)+LR	0.28	0.161	3.17
	G(10)+PR	0.28	0.156	2.12
	G(10)+ILU	0.28	0.188	1.14
Airfoil Grid 2 237x61	L(10)	1.10	0.011	3.68
	P(20)	1.10	0.002	5.13
	G(5)+L	1.10	0.618	3.77
	G(10)+P	1.10	0.609	4.48
	G(10)+ILU	1.10	0.737	1.45
Airfoil Grid 3 473x121	L(10)	4.35	0.023	8.79
	P(40)	4.35	0.004	12.56
	G(10)+ILU	4.35	2.920	3.91
Multi-Element 68K pts	L(10)	5.17	0.015	49.7
	P(20)	5.17	0.003	14.7
	G(10)+ILU	5.17	3.468	5.37

Table 1: Cost comparisons of iterative methods for INS2D for various cases and schemes. L(n): Line Relaxation for n iterations, P(n): Point Relaxation for n iterations, G(m)+ X : GMRES with subspace size m using scheme X for preconditioner.

Figure 6 shows the effect of decreasing the **Inner Iteration** tolerance level ϵ thereby solving the GMRES step more accurately. In this case, to reach a certain convergence criteria, e.g.**Outer Iteration** residual to 10^{-8}, in the least number of iteration, requires decreasing ϵ, $\epsilon = 10^{-5}$ gets there in 50 **Outer Iteration** . On the other hand, the CPU time cost (shown in milliseconds/point) and average subspace size m (which leads to addition memory requirements proportional to m) indicate that a loose tolerance, say $\epsilon \approx 10^{-1}$ and small m produce the most efficient combination. This leads to $m = 10$ as the optimal choice both in terms of CPU efficiency and memory requirements.

Memory estimates for the three-dimensional code INS3D include a base memory of 146 words/point, additional GMRES memory: $4 \times (m+4)$ words/point and preconditioner memory of 16 words/point. Thus for GMRES(10)+ILU(0) the total memory is 218 words/point. Examples include a simple wing: 0.2 million points (Mpoints): 43.6 MW, a wing+slat+flap: 1.6 Mpoints: 349 MW, and a C17 Aircraft: 25 Mpoints: 5450 MW = 5.45 GW. These requirements are excessive in three-dimensions and need to be reduced if these codes are to be used in practice.

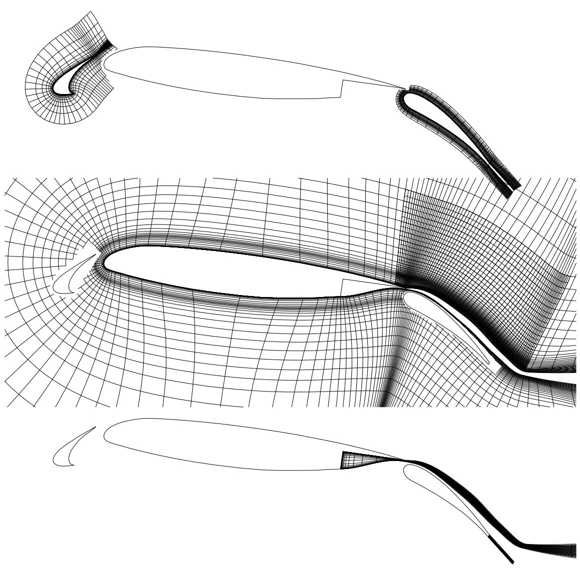

Figure 1: Grid around a three-element airfoil.

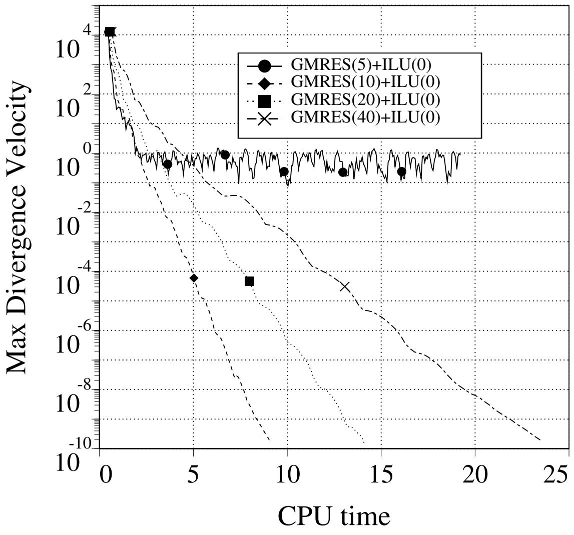

Figure 2: Results for Grid 1 showing effect of subspace size GMRES(m) on Newton-GMRES.

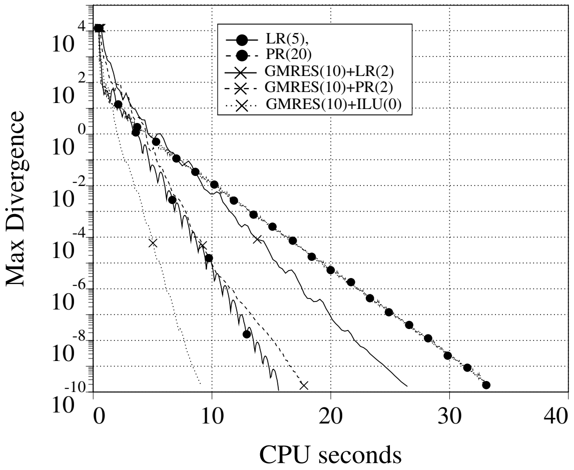

Figure 3: Results for Grid 1 Comparing Various Schemes.

4 Unstructured Mesh Compressible Navier-Stokes

Barth [5] has implemented the Newton-GMRES algorithm into a two and three -dimensional unstructured mesh Navier-Stokes approach. In this case the flow equations are solved using an edge-based unstructured mesh quadrature scheme characterized as an approximate and/or exact Roe Riemann solver based on piecewise polynomial reconstruction, this

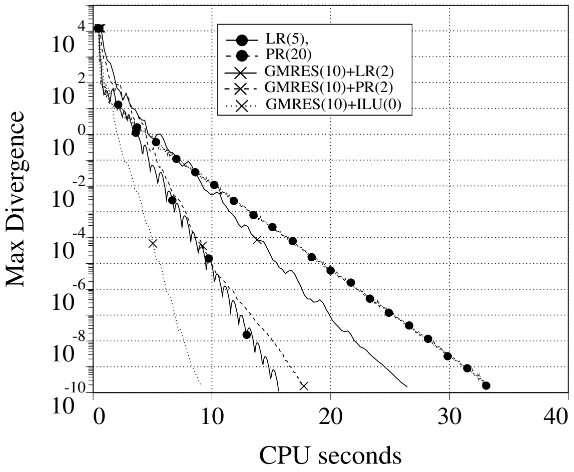

Figure 4: Results for Grid 2 Comparing Various Schemes.

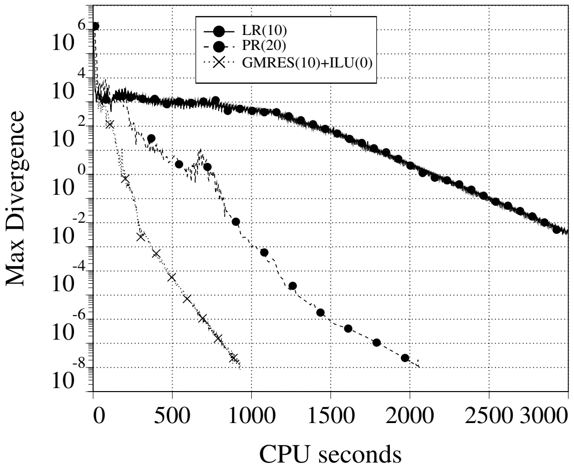

Figure 5: Results for Multi-Element Airfoil Comparing Various Schemes.

defines the **Function Evaluation** . Details of the flow algorithm can be found in Barth [6, 2, 3]. The details of the Newton-GMRES implementation include exact Ap products for the second order discretization with a first or second order approximate $\mathcal{A}$ used only to construct the ILU preconditioner. Barth presents three methods to compute Ap products, one in which the exact Jacobian is stored (requiring a significant increase in memory requirements), a numerical evaluation using Fréchet derivatives [8] which is a matrix-free approach, and another matrix free approach using an exact product form where proper linearization of the Riemann solvers and the recon-

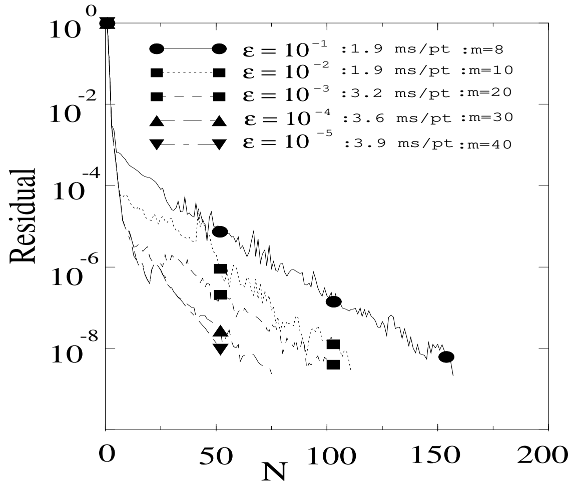

Figure 6: Comparison for various **Inner Iteration** tolerance levels for Grid 2 case. Tolerance levels are given along with milliseconds per point to convergence and average subspace size used m.

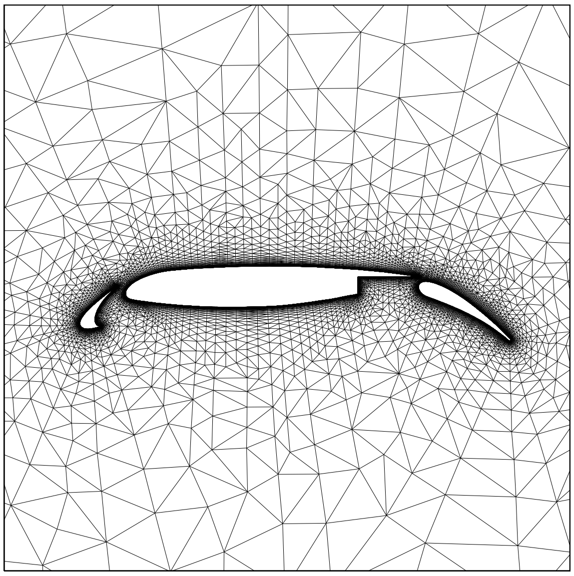

Figure 7: Multi-element airfoil triangulation, 22,000 vertices.

struction/quadrature mechanism of the residual vector assembly are used to produce the Ap product. Since exact Ap products are used, quadratic convergence can be realized. The preconditioned **Inner Iteration** is also fairly efficient, employing a subspace size on the order of $m_r = 12$ and a modest number of restarts. The resulting scheme can be mapped very successfully onto a parallel processor environment.

Figures 7,8,9,10 show an example computation for viscous flow with turbulence about the multiple-element airfoil geometry. This geometry has been triangulated using the Steiner triangulation algorithm described in [4], see Figure 7. The mesh contains approximately 22,000 vertices with cells near the airfoil surface attaining aspect ratios greater than 1000:1. This example provides a demanding test case for CFD algorithms. The experimental flow conditions are $M_\infty = .20$, $\alpha = 16°$, and a Reynolds number of 9^{-6}. Experimental results are given in [20] and computed results are shown in Figure 8. Even though the wake passing over the main element is not well resolved, the surface pressure coefficient shown in Figure 9 agrees quite well with experiment.

The convergence history shown in Figure 10 is typical for aerodynamic high lift computations.

Some of the more practical aspects from Barth's [5] implementation of Newton-GMRES are discussed below.

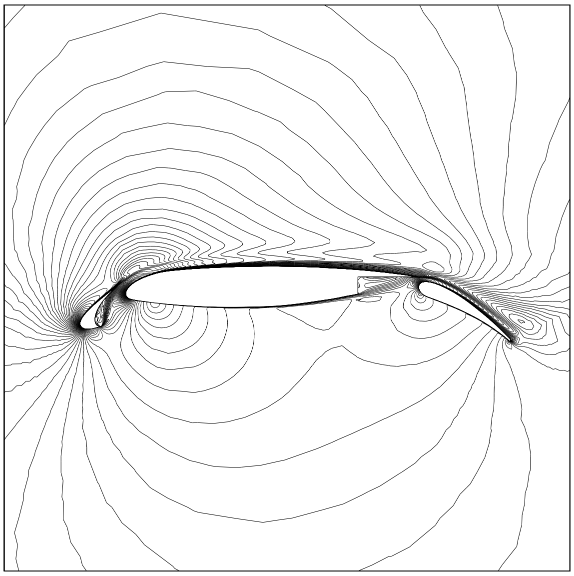

Figure 8: Multi-element airfoil solution isomach contours, $M_\infty = 0.2$, $\alpha = 16.0°$, $Re = 9.0$ million.

4.1 Storage Requirements

In practice we will be solving systems of l coupled equations so that each nonzero entry of the matrix is actually a small $l \times l$ block. The schemes employed require data from distance-one neighbors in

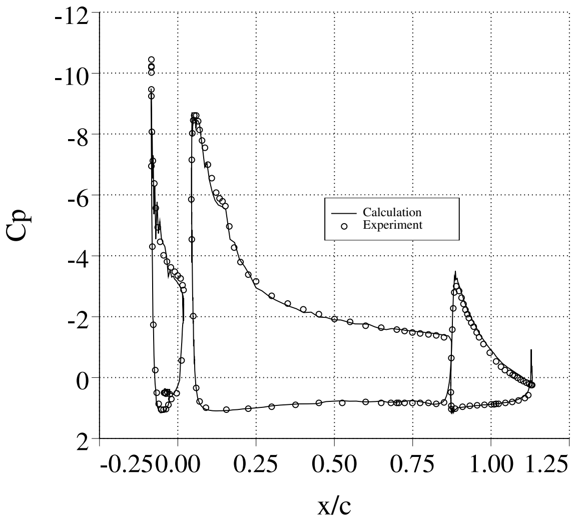

Figure 9: Comparison of computational and experimental surface pressure coefficients.

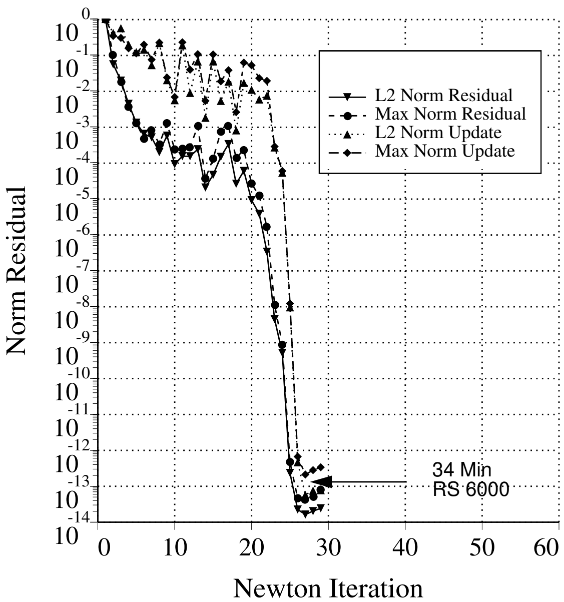

Figure 10: Solution convergence history.

the graph (mesh). In addition, the higher order accurate schemes require distance-two neighbors in building the scheme, see Barth [5, 3, 6]. First consider the situation in which the scheme requires only distance-one neighbors. The number of nonzero entries in each row of the matrix is related to the number of edges incident to the vertex associated with that row. Equivalently, each edge $e(v_i, v_j)$ will guarantee nonzero entries in the i-th column and j-th row and similarly the j-th column and i-th row. In addition, nonzero entries will be placed on the diagonal of the matrix. From this counting argument we see that the number of nonzero block entries, nnz, in the matrix is exactly twice the number of edges plus the number of vertices, $2E+N$ (approximately $7N$ in 2D). Table 2 (based on a similar counting argument) shows approximate requirements for storing distance-one and distance-two neighboring information as a sparse matrix.

Note that the entries of the sparse matrix associated with Newton's method (for solution of the Navier-Stokes equations and an associated 1 equation turbulence model) are actually small 5×5 and 6×6 blocks in two and three dimensions respectively. At first glance, this storage requirement appears prohibitively large. While this may be true to some extent today, the memory capacity of computers is expanding at a rapid rate. It is quite reasonable to expect that in the foreseeable future sufficient memory will be available for solving most problems of engineering interest. Even so, it is possible to reduce, and in some cases eliminate, the explicit storage of the Jacobian matrix without compromising the favorable convergence characteristics of Newton's method.

Dim.	nnz (Distance-1)	nnz (Distance-2)
2	$7N$	$19N$
3	$14N$	$55N$

Table 2: Storage Estimates for Sparse Matrices.

4.2 Calculating Analytic Jacobian Derivatives

In this section we address the task of computing Jacobian derivatives for Newton's method. In the following section we consider the related task of multiplying an arbitrary vector by the Jacobian matrix.

A major task in the overall calculation of the Jacobian derivatives for the finite-volume discretization is the linearization of the numerical flux vector with respect to the two solution states, e.g. given the Roe flux function [16]

$$\mathbf{h}(\mathbf{u}^R, \mathbf{u}^L; \mathbf{n}) = \frac{1}{2}\left(\mathbf{f}(\mathbf{u}^R, \mathbf{n}) + \mathbf{f}(\mathbf{u}^L, \mathbf{n})\right) \quad (7)$$

$$- \frac{1}{2}|A(\mathbf{u}^R, \mathbf{u}^L; \mathbf{n})| \left(\mathbf{u}^R - \mathbf{u}^L\right) \quad (8)$$

we require the Jacobian terms $\frac{d\mathbf{h}}{d\mathbf{u}^R}$ and $\frac{d\mathbf{h}}{d\mathbf{u}^L}$. Here, $\mathbf{f}$ is the flux function, $\mathbf{n}$ a geometric normal, $A = \frac{\partial \mathbf{f}}{\partial \mathbf{u}}$, the physical flux Jacobian, evaluated at some combination of the right and left states of the flow variables, $\mathbf{u}^R, \mathbf{u}^L$. Exact analytical expressions for these terms are available [1]. In constructing the Jacobian matrix for the entire scheme it is useful to conceptualize the finite-volume scheme in *composition form*:

$$\mathbf{R}(\mathbf{Q}) = \mathcal{L}_1(\mathcal{L}_2(\mathbf{Q})), \tag{9}$$

with $\mathcal{L}_1$ representing the flux quadrature and accumulation step and $\mathcal{L}_2$ representing the data reconstruction step. In this form, each operator requires distance-1 information. The Jacobian matrix can then be written as

$$\frac{d\mathbf{R}}{d\mathbf{Q}} = \frac{d\mathcal{L}_1}{d\mathcal{L}_2}\frac{d\mathcal{L}_2}{d\mathbf{Q}} \tag{10}$$

with the critical observation that the Jacobian matrix can be calculated as the sparse product of two matrices. This could potentially be an expensive task, but because of the special form of $\mathcal{L}_1$ and $\mathcal{L}_2$, the resulting sparse product produces at most distance-2 fill and can be computed at reasonable cost.

4.3 Exact and Approximate Jacobian Matrix-Vector Products

Consider the standard matrix equation $\mathbf{b} - \mathcal{A}\mathbf{x} = 0$. Iterative matrix solution algorithms for this problem requires the computation of matrix-vector products of the form $\mathcal{A}\mathbf{p}$ for some arbitrary $\mathbf{p}$ vector. In the approximate Newton algorithm

$$\mathcal{A} = \left[\frac{D}{\Delta t} - \frac{d\mathbf{R}}{d\mathbf{Q}}\right] \tag{11}$$

where D is a positive diagonal matrix. In practice the diagonal entries are locally scaled as a exponential function of the norm of the residual

$$\frac{D_i}{\Delta t} = \frac{\text{cfl}_i}{\text{cfl}_{max}}, \qquad \text{cfl}_{max} = f(||\mathbf{R}(\mathbf{Q})^n||)$$

so that when $||\mathbf{R}(\mathbf{Q})|| \to 0$, $\text{cfl}_{max} \to \infty$ and the scheme approaches Newton's method. It should be emphasized that by using this strategy, the scheme is technically an approximate Newton method which becomes exact only in the final few iterations of the computation.

A major step in the matrix-vector product $\mathcal{A}\mathbf{p}$ is the computation of Jacobian derivatives in the direction of $\mathbf{p}$ (a Fréchet derivative)

$$\mathcal{A}\mathbf{p} = \frac{D}{\Delta t}\mathbf{p} - \frac{d\mathbf{R}}{d\mathbf{Q}}\mathbf{p}. \tag{12}$$

Several possible strategies exist for computing the needed Fréchet derivatives:

4.3.1 Sparse Matrix-Vector Multiply

The most straightforward strategy is to analytically compute and store the Jacobian matrix using a compressed storage scheme designed for sparse matrices. This strategy has the added benefit that a copy of the matrix can also be used as a preconditioner for the iterative solver. In addition, the explicit storage also permits the formation of the transposed matrix problem which is often encountered in optimization procedures coupled with Newton's method. Obviously, a drawback of this approach is the large storage requirement.

4.3.2 Approximate Fréchet Derivatives

An alternative to analytically calculating Fréchet derivatives is to approximate them using finite differences, [12] [8] [10]. The required Fréchet derivative is a limiting form of the difference approximation

$$\frac{d\mathbf{R}}{d\mathbf{Q}}\mathbf{p} = \lim_{\epsilon \to 0} \frac{\mathbf{R}(\mathbf{Q} + \epsilon\mathbf{p}) - \mathbf{R}(\mathbf{Q})}{\epsilon}.$$

The primary concern with this approach is the accuracy of derivatives and the optimal choice for ϵ. If derivatives are not computed accurately then methods such as GMRES iteration may stall or fail. Using a forward difference approximation, ϵ must be carefully chosen. In general it is insufficient to choose ϵ as a constant such as the square root of machine precision. Johan [12] also mentions this fact and gives some analysis for choosing ϵ but this analysis assumes that $\mathbf{R}(\mathbf{Q})$ is well scaled. A common choice for ϵ is given by

$$\epsilon = \delta_0 + \delta_1 \frac{||\mathbf{Q}||}{||\mathbf{p}||} \tag{13}$$

with suitably chosen constants δ_0 and δ_1. An alternative to forward differencing is to use higher order accurate formula such as central differencing at double the computational cost.

The clear attraction of this approach is the low memory requirement. On the other hand, the numerical computation of Fréchet derivatives does not produce a matrix approximation which can be used to precondition the system.

4.3.3 Exact Product Forms

In this section we will present a technique for constructing matrix-vector products which is an exact calculation of the Fréchet derivative. Extension to systems and the inclusion of diffusion terms are also handled using this technique.

Let $G(E, V)$ denote the triangulation in 2D or 3D with n vertices and m edges. Next we define the incidence matrix

$$\mathcal{C}_{il} = \begin{cases} -1 & \text{if } v_i \text{ is the origin of edge } l \\ 1 & \text{if } v_i \text{ is the destination of edge } l \\ 0 & \text{otherwise} \end{cases} . \quad (14)$$

Let $\mathbf{h} = \mathbf{h}(\mathbf{u}^L, \mathbf{u}^R; \mathbf{n})$ denote the numerical flux function as defined by Equation 8. For a system of l coupled differential equations, the Jacobian matrix entries are actually small $l \times l$ blocks. For ease of exposition, we tacitly treat these small blocks as scalar entries. Under these simplifications, the desired matrix-vector product is given by

$$\frac{d\mathbf{R}}{d\mathbf{Q}}\mathbf{p} = \mathcal{C}^T \left[\left[\frac{d\mathbf{h}}{d\mathbf{u}^L}\right] \left[\frac{d\mathbf{u}^L}{d\mathbf{u}}\right] + \left[\frac{d\mathbf{h}}{d\mathbf{u}^R}\right] \left[\frac{d\mathbf{u}^R}{d\mathbf{u}}\right] \right] \mathbf{p} \quad (15)$$

where $\left[\frac{d\mathbf{h}}{d\mathbf{u}}\right] \in \Re^{m\times m}$ with nonzero diagonal elements, and $\left[\frac{d\mathbf{u}^L}{d\mathbf{u}}\right] \in \Re^{m\times n}$. If we do not incorporate monotonicity enforcement into the reconstruction procedure then a considerable simplification occurs in the calculation of matrix-vector products. The main idea is given in the following almost trivial lemma.

Lemma: Let $v = \mathcal{R}(U) = \mathcal{R}(u_1, u_2, ..., u_n)$ denote an arbitrary order reconstruction operator. If $\mathcal{R}$ depends linearly on u_i then

$$\frac{dv}{du}p = \mathcal{R}(p).$$

Proof: Linearity implies that

$$v = \mathcal{R}(u_1, u_2, ..., u_n) = \sum_{i=1}^{n} \alpha_i u_i$$

so that $\frac{dv}{du_i} = \alpha_i$. The desired result follows immediately

$$\frac{dv}{du}p = \sum_{i=1}^{n} \frac{dv}{du_i} p_i = \sum_{i=1}^{n} \alpha_i p_i = \mathcal{R}(p).$$

This lemma suggests the following procedure for calculation of matrix-vector products, from Eq. 15.

$$\frac{d\mathbf{R}}{d\mathbf{Q}}\mathbf{p} = \mathcal{C}^T \left[\left[\frac{d\mathbf{h}}{d\mathbf{u}^L}\right] \mathcal{R}^L(\mathbf{p}) + \left[\frac{d\mathbf{h}}{d\mathbf{u}^R}\right] \mathcal{R}^R(\mathbf{p}) \right] \quad (16)$$

This amounts to a reconstruction of the vectors $\mathbf{p}^L$ and $\mathbf{p}^R$ from $\mathbf{p}$ using the same reconstruction operator used in the residual computation. Next, the linearized form of the flux function is computed:

$$\mathbf{h}_{lin} = \frac{d\mathbf{h}}{d\mathbf{u}^L}\mathbf{p}^L + \frac{d\mathbf{h}}{d\mathbf{u}^R}\mathbf{p}^R.$$

Finally, the linearized fluxes are assembled using the same procedure as the residual vector assembly. In actual calculations, the conservative flow variables are not reconstructed, thereby necessitating that a change of variable transformation be embedded in the formulation. This is not a serious complication.

4.4 Matrix Preconditioning

In the present applications, we consider a preconditioning matrix based on the incomplete lower-upper (ILU) factorization of the matrix $\mathcal{A}$. ILU preconditioning is a popular and robust preconditioning procedure for use in iterative matrix solvers. ILU factorization is a modification to the standard Gaussian elimination for which the nonzero fill pattern is either preimposed or determined dynamically based on the size or location of fill elements. In this way the amount of storage required can be specified and in some instances minimized. Technical aspects of ILU factorization such as existence and spectral properties have been proven for M-matrices, but the general applicability is much broader and well documented in the literature. The triangular solves required in the application of ILU preconditioning generally give the method global support. This is usually considered a favorable characteristic of the method.

The finite-volume scheme with high order data reconstruction suggests two possible matrices suitable for incomplete factorization.

1. Distance-1 matrix preconditioning. Construct the preconditioning matrix from the Jacobian matrix associated with the lower (first) order accurate discretization of the flow equations. This matrix involves distance-1 neighbors in the triangulation. Matrix-vector products are computed "exactly" using the Jacobian matrix associated with the full second order accurate scheme.

2. Distance-2 matrix preconditioning. Use the Jacobian matrix of the entire second order accurate scheme for both matrix-vector products and preconditioning.

4.5 Performance of GMRES

The viscous multi-element test problem given above provides representative matrices for evaluating the GMRES algorithm. We construct approximate Newton matrices corresponding to flow CFL numbers of 10^3 and 10^8. In addition, distance-1 and distance-2 preconditioning matrices are used to accelerate the algorithms.

Figures 11-12 graph the convergence histories for the GMRES algorithm and the two choices of preconditioner. Since the matrix-vector products and preconditioning solves dominate the iterative calculation, convergence histories are plotted against the number of matrix-vector products required. Each GMRES iteration requires one matrix-vector product. The GMRES algorithm is clearly adversely affected by the distance-1 preconditioning. For this case the distance-1 preconditioned system requires roughly twice as many iterations as the distance-2 preconditioned system. In fact for CFL $= 10^8$, the convergence is unacceptably slow. In general we find that when using the distance-1 preconditioning matrix, an optimal CFL number exists for convergence and efficiency, which is large but not infinite.

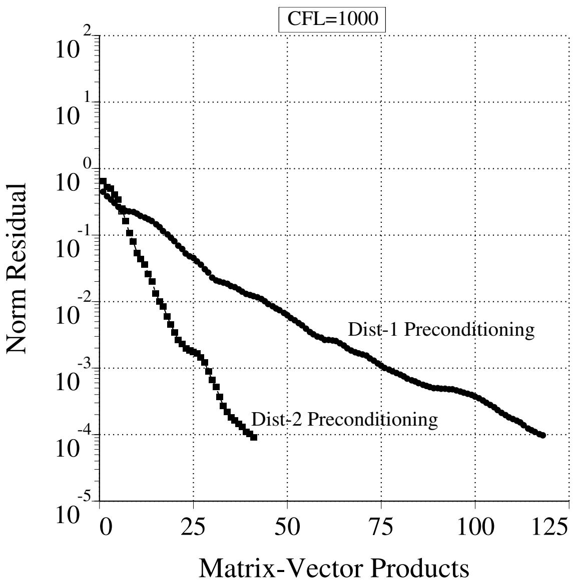

Figure 11: Viscous Flow matrix solution convergence histories for the $GMRES(30)$ algorithm at $CFL = 10^3$ using ILU(0) distance-1 and distance-2 preconditioning matrices.

5 Summary

Practical aspects for Newton-GMRES algorithms from working Navier-Stokes codes have been presented. In particular, implementation issues, such as memory requirements, accuracy requirements for Ap products, tradeoffs between full Newton and relax Newton and other pertinent approximations, have been discussed. Two approaches have been highlighted. In the incompressible Navier-Stokes code,

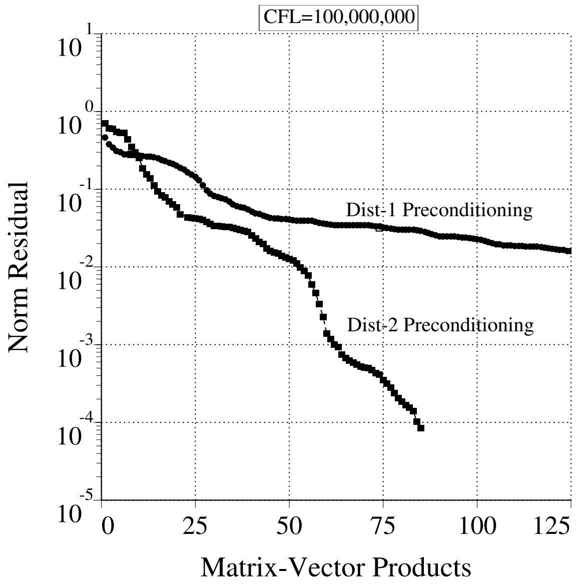

Figure 12: Viscous Flow matrix solution convergence histories for the $GMRES(30)$ algorithm at $CFL = 10^8$ using ILU(0) distance-1 and distance-2 preconditioning matrices.

the best strategy appears to be an inexact Jacobian (a first order accurate approximation to the third order accurate **Function Evaluation**) for the Ap products, a consistent ILU(0) preconditioner, a small subspace size and fairly loose tolerances for **Inner Iteration** convergence. In the unstructured mesh approach, exact Ap products are successfully coupled with a first order approximate ILU(0) preconditioner and tighter tolerances levels for **Inner Iteration** convergence. In both cases, an optimal strategy is found producing enhanced efficiencies. Although these conclusions are not universal, they do provide guidelines and practical suggestions for general implementations.

References

[1] T. J. Barth. Analysis of implicit local linearization techniques for upwind and tvd algorithms. Technical Report AIAA-87-0595, Reno, NV, 1987.

[2] T. J. Barth. *A Three-Dimensional Upwind Euler Solver of Unstructured Meshes.* Technical Report AIAA 91-1548, Honolulu, Hawaii, 1991.

[3] T. J. Barth. *Aspects of Unstructured Grids and Finite-Volume Solvers for the Euler and Navier-*

Stokes Equations, March 1994. von Karman Institute Lecture Series 1994-05.

[4] T. J. Barth. *Steiner Triangulation for Isotropic and Stretched Elements*. Technical Report AIAA 95-0213, Reno, NV, 1995.

[5] T. J. Barth. An unstructured mesh newton solver for compressible fluid flow and its parallel implementation. Technical Report AIAA-95-0221, Reno, NV, 1995.

[6] T. J. Barth and D. C. Jespersen. *The Design and Application of Upwind Schemes on Unstructured Meshes*. Technical Report AIAA 89-0366, Reno, NV, 1989.

[7] R.M. Beam and H.E. Bailey. Newton's method applied to finite-difference approximations for the steady-state compressiblenavier-stokes equations. *Journal of Computational Physics*, 93(1):108–127, 1991.

[8] P. Brown and Y. Saad. Convergence Theory of Nonlinear Newton-Krylov Algorithms. *SIAM J. Optimization.*, 4:297–330, 1994.

[9] A. J. Chorin. A numerical method for solving incompressible viscous flow problems. *Journal of Computational Physics*, 2(1):12–26, Aug. 1967.

[10] S. Eisenstat and H. Walker. Globally Convergent Inexact Newton Methods. *SIAM J. Optimization.*, 4:393–422, 1994.

[11] D. C. Jespersen and T. H. Pulliam. Approximate newton methods and flux vector splitting. Technical Report AIAA Paper 83-1899, AIAA 6th Computational Fluid Dynamics Conference, Danvers, MA, 1983.

[12] Z. Johan. *Data Parallel Finite Element Techniques for Large-Scale Computational Fluid Dynamics*. PhD thesis, Stanford University, Department of Mechanical Engineering, 1992.

[13] J. A. Meijerink and H. A. van der Vorst. Guidelines for the usage of incomplete decompositions in solving sets of linear equations as they occur in practical problems. *Journal of Computational Physics*, 44(1):134–155, 1981.

[14] T. H. Pulliam. Efficient solution methods for the navier-stokes equations. 1985. Lecture Notes for the von Kármán Institute For Fluid Dynamics Lecture Series : Numerical Techniques for Viscous Flow Computation In Turbomachinery Bladings, von Kármán Institute, Rhode-St-Genese, Belgium , 1985.

[15] T. H. Pulliam. *Implicit Methods in CFD*. Clarendon Press, March 1985. The Institute of Mathematics and Its Publications Conference Series,Proceeding of the ICFD 1988 Conference on Numerical Methods for Fluid Dynamics.

[16] P. L. Roe. Approximate Riemann Solvers, Parameter Vectors, and Difference Schemes. *J. Comput. Phys.*, 43, 1981.

[17] S. E. Rogers. A comparison of implicit schemes for the incompressible navier-stokes equations with artificial compressiblity. Technical Report AIAA-95-0567, Reno, NV, 1995.

[18] S. E. Rogers and D. Kwak. An upwind differencing scheme for the time accurate incompressible navier-stokes equations. *AIAA Journal*, 28(2):253–262, Feb 1990.

[19] Y. Saad and M. H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 7(3):856–869, 1986.

[20] W. Valarezo, C. Dominik, R. McGhee, and W. Goodman. *High Reynolds Number Confguration Development of a High-Lift Airfoil*. Technical Report AGARD Meeting In High-Left Aerodynamics 10-01, 1992.

[21] V. Venkatakrishnan. Newton solution of inviscid and viscous problems. Technical Report AIAA Paper 88-0413, AIAA 26th Aerospace Sciences Conference, Reno, NV, 1988.

[22] V. Venkatakrishnan. A perspective on unstructured grid flow solvers. Technical Report AIAA Paper 95-0667, AIAA 33rd Aerospace Sciences Conference, Reno, NV, 1995.

[23] L.B. Wigton, N.J. Yu, and D.P. Young. Gmres acceleration of computational fluid dynamics codes. Technical Report AIAA Paper 85-1494, AIAA 7th Computational Fluid Dynamics Conference, Cincinatti, OH, 1985.