Intelligent Nonlinear Solvers for Computational Fluid Dynamics

Thomas M. Smith, * Russell W. Hooper, † Curtis C. Ober ‡
and
Alfred A. Lorber §
Sandia National Laboratories,¶ Albuquerque, New Mexico, 87185-0316, USA

Implicit nonlinear solvers for solving systems of nonlinear PDEs are very powerful. Many compressible flow codes utilize Newton-Krylov (NK) methods and matrix-free Newton-Krylov (MFNK) methods for a range of flow regimes and different flow models such as inviscid, laminar, turbulent and reacting flows. One drawback is that these solvers are complex requiring the specification of many settings. Expertise is necessary to achieve high performance. There is a need to develop "intelligent nonlinear solvers" that are capable of changing settings dynamically and adapting to evolving solutions and changing solver performance, in order to reduce the burden on the user, and improve overall efficiency and reliability. In this paper we take the first steps in achieving automatic control of nonlinear solvers for compressible flows by combining semi- and fully- implicit solver strategies in ways that utilizes them more efficiently than simply applying one method or another during the entire solution procedure. The understanding gained from this work will lay the groundwork for future development of more autonomous "intelligent solvers".

I. Introduction

Implicit solvers are widely used to in compuational fluid dynamic applications to obtain steady-state solutions to the equations governing fluid flow. Semi-implicit (point-implicit) methods are one of the most common. Semi-implicit methods are relatively easy to implement, have low memory requirements and can march at large time step sizes compared to explicit methods. Semi-implicit iterations are only modestly more expensive than explicit iterations and tend to converge linearly. They are also robust in the sense that they are relatively easy to use. However, convergence "stalling" can be a problem in certain circumstances.

In recent years, Newton-Krylov (NK) methods are becoming more popular. NK methods are less straight forward to use and more expensive per iteration than semi-implicit methods. However, NK methods are very efficient for as the solution is approached in an iterative sense, quadratic convergence rates can be achieved. Very large time steps can be used to advance the solution to steady-state. They are also robust due to the effectiveness of Krylov subspace iterative linear solvers.

In both methods, solutions are achieved iteratively by solving a series of nonlinear problems where the system equations are linearized and then solved with an iterative linear solver. Semi-implicit solvers combine the nonlinear and linear loops together, solving a modified linear system less accurately but more cheaply. Typically, semi-implicit solvers are cheaper than NK methods in the beginning when the CFL is small and the linear systems are dominated by a large diagonal inertia term. Later, as the inertia term becomes smaller, the linear problem becomes more difficult to solve and NK methods become more efficient. Therefore, it would make sense to combine these approaches in a single solver stategy.

*Computational Sciences, Senior Member of the Technical Staff, Exploratory Simulation Technologies, AIAA Member
†Applied Computational Methods
‡Exploratory Simulation Technologies
§Thermal/Fluid Computational Engineering Sciences
¶Sandia is a multiprogram laboratory operated by Sandia Corporation, A Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000. This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

NK methods have many solver settings that must be "tuned" in order for the solver to perform optimally. These settings are problem dependent and sometimes non-intuitive. Prescribing these settings requires expert knowledge and can be intimidating to new application code users. In addition, static settings may not be the most efficient or even adequate to ensure convergence to a solution. One reason is that as the solution evolves iteratively, the nonlinear behavior changes. A good example of this can be found in shock capturing problems. The steady-state solution is obtained by time marching from initial to final steady-state. Typically, simulations are initialized with uniform flow conditions. The shock wave initially forms on the body leading edge and propagates upstream until an equilibrium position is reached. This transient behavior prevents rapid convergence until the final shock location is reached, at which time the solution convergence can progress rapidly. The most effective solver setting values may change during the solution procedure. Therefore, it is not possible to prescribe static settings from an input file and expect to achieve the highest performance from the solver.

Based on this discussion, there is a need to develop "intelligent nonlinear solvers" that can reduce the burden on users of prescribing solver settings, and can automatically modify these settings in order to improve overall robustness and performance. It is envisioned that a controller would monitor solver diagnostics such as nonlinear convergence rate, number of linear iterations per nonlinear iteration step and time step size and based upon expertise encoded in the controller, modify the appropriate solver settings and continue the process. A controller using expertise encoded into the software would rapidly process the diagnostic data to produce an appropriate response.

The approach we are taking does not rely on traditional control theory where a mathematical model is derived that can predict output response from given input. Instead we propose to use control theory which can process vague or imprecise input data and produce an effective output response. Unlike traditional linear control theory, no mathematical model exists for the nonlinear solver which is viewed as a dynamical system. Instead expert knowledge is encoded into the solver controller in the form of rules that take the form of IF-THEN-ELSE constructs.[1] Fuzzy logic is a good example of this type of control theory.[1]

The approach that we will take contains a four step iterative process. First we identify a solver setting or multiple settings that we want to dynamically control and write the software infrastructure that will allow those particular parameters to be dynamically modified. In other words, most solver settings are statically set once in the input file, not expected to change. Several examples are worth noting.

1) Adaptive removal/addition of the mass terms in the Jacobian and preconditioner matrix. When the solution is close to steady-state, removal of the mass term can greatly speed convergence. Removing the mass term too soon can be catastrophic. Manzano *et al.*[2] used this strategy as a start-up procedure.

2) Adaptive solver algorithms. Using explicit, and semi-implicit solver algorithms to get close to the final solution followed by NK in the final stages can be much cheaper than using NK for the entire solution procedure.

3) Adaptive Jacobian recompute frequency. Computing and factorizing the Jacobian matrix each nonlinear iteration is not necessary in the initial stages of the solution procedure when the time step size is small and the linear solves are relatively easy. Recomputing the preconditioner every nonlinear iteration is sometimes necessary in the final stages. A single static frequency used throughout is sub-optimal.

4) Solver startup strategies. In some situations, a lower fidelity discretization can be used to obtain a solution and then be used as the initial conditions for a higher-order solution. This is accomplished by changing the order of spatial accuracy of the discretization.

The second step involves an audit of operator costs. This simply means computing the cost of evaluating operators such as; residual function, preconditioning matrix, factorizing the matrix, a matrix-free Krylov iteration or a matrix-vector product. This data will be used to do a cost analysis and guide the development of rules used by the controller to make decisions on how best to change settings.

The third step is to quantify setting sensitivities to the solution convergence. This is where expert knowledge is introduced into the process. By choosing a representative set of problems and quantifying the change in convergence with change in solver setting, "knowledge" is gained that will be useful in developing rules used by the controller for making decisions. These rules may take the form of IF-THEN-ELSE constructs.

The fourth step is to put it all together. Using the cost analysis data and sensitivity data as input, a single setting controller component will be designed and implemented that outputs a new solver setting. The controller will be based upon the rules generated in step three. Once successful control has been demonstrated, the process can be repeated for the next setting of interest.

The paper is organized as follows; first the governing equations for compressible fluid flow are presented

followed by a brief description of the finite-volume spatial discretization. Next, time marching strategies are presented with an emphasis on NK and semi-implicit methods. Candidate solver algorithms and settings that arise in the discussion will be identified. Next, results that illustrate the benefit from using adaptive solver strategies will be presented. Finally, conclusions will be made.

II. Flow Model

A. Governing Equations

We seek solutions to the Euler and Navier-Stokes equations that govern fluid flow. In integral form the governing equations can be written as

$$\int_{\mathcal{V}} \frac{\partial \mathbf{W}}{\partial t}\, d\mathcal{V} + \oint_{\Gamma} \left(\mathbf{F}^c - \mathbf{F}^v\right) \cdot \mathbf{n}\, d\Gamma = 0 \tag{1}$$

where $\mathbf{W} = \{\rho,\, \rho u,\, \rho v,\, \rho w,\, \rho E\}^T$ is the vector of conserved variables, $\mathcal{V}$ represents the fluid domain and Γ its boundary. The inviscid flux vector is;

$$\mathbf{F}^c \cdot \mathbf{n} = \left\{ \begin{array}{c} (\rho u_i) n_i \\ (\rho u_i u_j + p\delta_{ij}) n_j \\ ((\rho E + p)\, u_i) n_i \end{array} \right\}$$

the viscous flux vector is;

$$\mathbf{F}^v \cdot \mathbf{n} = \left\{ \begin{array}{c} 0 \\ \tau_{ij} n_j \\ (-q_i + u_j \tau_{ij}) n_i \end{array} \right\}.$$

In the above equations, ρ is the density; u_i ($u = u_1$, $v = u_2$, $w = u_3$) which can be written $\mathbf{u}$ are velocity components in each Cartesian direction x_i ($i = 1, 2, 3$ and $j = 1, 2, 3$ is implied); p is the pressure defined by the ideal gas equation of state, $p = \rho R T$; R is the gas constant; E is the total energy per unit mass, defined as $E = e + (u^2 + v^2 + w^2)/2$; e is the specific internal energy, defined as $e = \frac{p}{(\gamma-1)\rho}$; γ is the ratio of specific heats; T is the temperature; $n_i = \mathbf{n}$, is the outward pointing surface normal vector, and δ_{ij} is the Kronecker delta. We also define a the primitive state vector as $\mathbf{U} = \{\rho, u, v, w, p\}^T$.

The viscous stress tensor is given by,

$$\tau_{ij} = \mu \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) - \frac{2}{3} \mu \frac{\partial u_k}{\partial x_k} \delta_{ij} \tag{2}$$

where μ is the dynamic coefficient of viscosity. The heat flux vector is assumed to follow Fourier's Law,

$$q_i = -\kappa \frac{\partial T}{\partial x_i} \tag{3}$$

where κ is the thermal conductivity.

On a solid impermeable surface, in the case of inviscid flow, the velocity vector must be tangent to the surface, $\mathbf{u} \cdot \mathbf{n} = 0$ and there is no mass or heat flux crossing the boundary and so the flux function becomes,

$$\mathbf{F}^b = \mathbf{F} \cdot \mathbf{n}^b = \left\{ \begin{array}{c} 0 \\ p\mathbf{n}^b \\ 0 \end{array} \right\}.$$

On solid no-slip boundaries, $\mathbf{u} = 0$ is enforced strongly. In addition, if the surface temperature is specified, a boundary temperature is enforced strongly. The adiabatic condition is implemented as a zero flux condition. Far-field boundary conditions are defined by specifying an infinity state $\mathbf{U}_\infty$ that is applied as the right state in the Riemann solver resulting in a boundary flux.

B. Edge-Based Finite Volume Spatial Discretization

The compressible flow code, Premo, uses the SIERRA framework,[3] which provides common services to application codes such as user-input parsing, bulk mesh-data input/output, interfaces to linear and nonlinear solvers, inter-mesh transfers, parallel communications, dynamic load balancing, and mesh adaptivity.

Premo is a node-centered, edge-based, finite-volume, algorithm similar to the formulations of Haselbacher and Blazek[4] and Luo *et al.*[5]

Control volumes and surfaces are defined by the median dual of the primitive finite-element mesh. This construction results in non-overlapping, space-filling, control volumes that are associated with nodes. Control-surface area vectors are area-weighted averages and are stored on the edges.

Equation 1 must hold for each control volume. If we define volume-averaged values of the conserved state vector ($\mathbf{W}$), and apply the midpoint rule for integration on surfaces, we can write the semi-discrete form of eq. (1) as,

$$\frac{\partial \mathbf{W}_I}{\partial t} + \frac{1}{\delta \mathcal{V}_I} \sum_{J \in NE(I)} (\mathbf{F}^c_{IJ} - \mathbf{F}^v_{IJ}) |\delta \Gamma_{IJ}| = 0 \tag{4}$$

$$\frac{\partial \mathbf{W}^b_I}{\partial t} + \frac{1}{\delta \mathcal{V}_I} \sum_{J \in NE(I)} (\mathbf{F}^c_{IJ} - \mathbf{F}^v_{IJ}) |\delta \Gamma_{IJ}| + \mathbf{F}^b |\delta \Gamma^b_I| = 0 \tag{5}$$

where I is the current node (*i.e.*, control volume), $\delta \mathcal{V}_I$ is the control volume, $NE(I)$ is the set of nodes connected to node I by edge IJ, J is the second node on the edge, $|\delta \Gamma_{IJ}|$ is the magnitude of the control-surface area, $\mathbf{W^b_I}$ are boundary nodes and $\mathbf{F}^b |\delta \Gamma^b_I|$ represents boundary fluxes. The convective and viscous fluxes are evaluated at the edge midpoint and depend on the reconstructed data, normal vector and nodal gradients; $\mathbf{F}^c_{IJ} = \mathbf{F}(\mathbf{U}^+_I, \mathbf{U}^-_J, \mathbf{n}_{IJ})$, $\mathbf{F}^v_{IJ} = \mathbf{F}(\mathbf{U}_I, \nabla \mathbf{U}_I, \mathbf{U}_J, \nabla \mathbf{U}_J, \mathbf{n}_{IJ})$. Details of the reconstruction of interface data is given the Appendix.

III. Nonlinear Solution Strategies

We seek a steady-state solution for eq. (1) for which the time derivative term is zero. Spatial discretization of eq. (1) leads to a system of nonlinear algebraic equations represented by

$$\mathbf{F}(\mathbf{W}) = \frac{\partial \mathbf{W}}{\partial t} + \mathbf{R}(\mathbf{W}) = \mathbf{0} \tag{6}$$

where $\mathbf{W}$ is the vector of unknowns, and $\mathbf{R}(\mathbf{W})$ is the spatial residual equations which have no explicit time derivative dependencies. At steady-state $\mathbf{R}(\mathbf{W}) = \mathbf{0}$. This is achieved by solving a sequence of nonlinear problems of the form of eq. (6) in which the stabilizing term represented by $\frac{\partial \mathbf{W}}{\partial t}$ is gradually removed.

A. Pseudo-Transient Newton-Krylov Based Solver

This technique is a variation of the standard Newton's method. Consider an incremental update to the state vector,

$$\Delta \mathbf{W}^n = \mathbf{W}^{n+1} - \mathbf{W}^n \quad . \tag{7}$$

where n is the nonlinear iteration/time step. Linearizing $\mathbf{R}$ about the current state and rearranging yields a pseudo-transient variation to Newton's method,

$$\left(\frac{1}{\Delta t} + \frac{\partial \mathbf{R}^n}{\partial \mathbf{W}} \right) \Delta \mathbf{W}^n = -\mathbf{R}^n \tag{8}$$

which can be written as

$$\mathbf{J} \Delta \mathbf{W}^n = -\mathbf{R}^n \ . \tag{9}$$

This form is used extensively in CFD codes where the governing equations are advection dominated and time marching must be used to solve them.

We choose to increase the pseudo-time step at each step as follows:

$$\Delta t^n = f_R \Delta t^{n-1} \quad or \quad \Delta t^n = f_A \frac{||\mathbf{R}||^{n-2}}{||\mathbf{R}||^{n-1}} \Delta t^{n-1} \tag{10}$$

where f_R are f_A are ramp and adaptive CFL factors. For some problems with complex physics, *e.g.*, shocks, turbulence, etc., use of a monotonically increasing Δt can lead to catastrophic failures such as NaN, resulting from the nonlinear solver attempting to sample inadmissible solution states (*i.e.* negative density). The nonlinear solver library NOX provides checks for such failures and allows the application code, Premo, opportunity to recover from these without terminating the simulation. In this work, we choose to simply reduce the pseudo-time step by half and resume the solution process with the last "good" solution vector. Hence, for catastrophic failures,

$$\Delta t^n = 0.5\Delta t^{n-1} \tag{11}$$

$$\mathbf{W}^n = \mathbf{W}^{n-1} \quad . \tag{12}$$

Both of these techniques, acceleration of the time step size and fault tolerance are widely used in the CFD community. These are examples of adaptive methods and have a great impact on solver performance, robustness and usability.

Iterations are continued until a user-specified convergence is achieved,

$$||\mathbf{R}^n|| \leq \epsilon_a, \quad ||\mathbf{R}^n||/||\mathbf{R}^0|| \leq \epsilon_b, \quad ||\Delta\mathbf{W}^n|| \leq \epsilon_c \ . \tag{13}$$

Each nonlinear iteration of eq. (8) requires a linear system of equations to be solved. Inexact Newton's methods are commonly used with iterative solvers. In this case the each linear sub-problem is solved such that the inequality

$$||\mathbf{J}\Delta\mathbf{W} + \mathbf{R}(\mathbf{W})||_2 \leq \eta||\mathbf{R}(\mathbf{W})||_2 \tag{14}$$

is satisfied. We note that research[6] has been conducted in the area of adaptively changing η with successive nonlinear iterations. These techniques are available in NOX, however, limited experimentation with these techniques for the class of problems of interest has not shown any advantage in using them.

The work presented here is based on solving eq. (8) using the iterative method GMRES[7] and various means of preconditioning. We select GMRES in order to accommodate non-symmetric Jacobian operators arising from up-winding methods.

The matrix-free operator is used exclusively as a Jacobian operator and is based on the matrix-free Newton-Krylov (MFNK) ideas described by Brown and Saad[8] and Knoll and Keyes.[9] Linear solves involve repeated action of the Jacobian on a vector, *e.g.*, $\mathbf{p}$, (used to generate each Krylov vector), which can be approximated as follows:

$$\mathbf{J}\mathbf{p} \approx \frac{\mathbf{F}(\mathbf{W} + \epsilon\mathbf{p}) - \mathbf{F}(\mathbf{W})}{\epsilon} \quad . \tag{15}$$

Right-preconditioning[7] is defined as follows,

$$\mathbf{J}\mathbf{M}^{-1}\mathbf{y} = -\mathbf{R} \tag{16}$$

$$\mathbf{M}\Delta\mathbf{W} = \mathbf{y} \tag{17}$$

where $\mathbf{M}$ is the preconditioning operator. When preconditioning is applied to solving eq. (8) using the matrix-free Jacobian operator, eq. (15) takes the modified form

$$\mathbf{J}\mathbf{M}^{-1}\mathbf{p} \approx \frac{\mathbf{F}(\mathbf{W} + \epsilon\mathbf{M}^{-1}\mathbf{p}) - \mathbf{F}(\mathbf{W})}{\epsilon} \tag{18}$$

and each finite-difference based matrix-vector operation now requires application of the preconditioning operator $\mathbf{M}$. For this work, we use incomplete LU factorization (ILU(#)) of the matrix $\mathbf{M}$ as the preconditioner for the iterative linear solver. The preconditioner matrix is assembled based on the edge-graph taking the partial derivatives of the first-order advection operator, (eq. 35) and viscous fluxes based on edge differences (eq. 40). A summary of how the Jacobian is formed can be found in the Appendix and in Smith *et al.*[10] where performance and robustness comparisons of several different approximations to the Jacobian are presented. Matrix assembly and LU factorization account for a significant fraction of the cpu time in each nonlinear iteration. However early in most simulations, the matrix is dominated by a large mass term which is changing relatively slowly - late in the simulation, the solution is changing gradually and again the matrix may be changing slowly. Therefore, it is reasonable to expect that the preconditioner could be reused in successive nonlinear iterations to reduce cpu time. This is indeed the case. The goal in adaptive

solver development would be to try and predict the "best" recompute frequency based upon available solver performance data.

Both GMRES and application of the preconditioner are performed using the AztecOO linear solver library[11] via an interface provided by NOX. Efficient testing of the many Jacobian/preconditioner operator choices (along with associated options) is enabled by interfacing the compressible flow code, Premo, to a library of nonlinear solver algorithms, NOX,[12] currently being developed at Sandia National Laboratories.

B. Semi-Implicit Solvers

Semi-implicit nonlinear solvers designed to achieve steady-state solutions are widely used by the CFD community. Memory requirements are less than NK methods because only the diagonal block matrix is stored and there are no Krylov sub-space vectors to store. Large CFL numbers can be used. The linear system is solved by successive "sweeps" over the control volumes. The cost of each sweep is comparable to an explicit time step using a Runge-Kutta integration method.

Chen and Wang[13] and Sharov *et al.*[14] present a symmetric Gauss-Seidel solver for steady-state solutions for unstructured-mesh solvers. With reference to eqs. (8-9), the linearized system is factored into the strictly upper, strictly lower and block diagonal matrices

$$\mathbf{J} = \mathbf{L} + \mathbf{U} + \mathbf{D} \tag{19}$$

where $\mathbf{J}$ is the Jacobian. Factorization of $\mathbf{J}$ and rearrangement of eq. 9 results in,

$$(\mathbf{D} + \mathbf{L})\mathbf{D}^{-1}(\mathbf{D} + \mathbf{U})\Delta\mathbf{W} = -\mathbf{R} + (\mathbf{L}\mathbf{D}^{-1}\mathbf{U})\Delta\mathbf{W} \ . \tag{20}$$

Neglecting the second term on the right-hand-side, this modified system is solved by symmetric forward and backward sweeps.[15] Starting with the guess $\Delta\mathbf{W} = \mathbf{0}$ for the $k = 0$ iteration, $\Delta\mathbf{W}$ is updated by at each node by a (forward sweep):

$$\Delta\mathbf{W}^{k+1/2} = \mathbf{D}^{-1}[-\mathbf{R} - \mathbf{L}\Delta\mathbf{W}^{k+1/2} - \mathbf{U}\Delta\mathbf{W}^{k}] \tag{21}$$

followed by a (backward sweep):

$$\Delta\mathbf{W}^{k+1} = \mathbf{D}^{-1}[-\mathbf{R} - \mathbf{U}\Delta\mathbf{W}^{k+1} - \mathbf{L}\Delta\mathbf{W}^{k+1/2}] \ . \tag{22}$$

Forward sweeps loop over all nodes starting from node 1 to node N - backward sweeps loop node N to 1, where N is the number of nodes. Iterations are terminated when a specified number of iterations (k_{max}) have been completed or when the norm tolerance

$$\frac{||\Delta\mathbf{W}^{k} - \Delta\mathbf{W}^{k-1}||}{||\Delta\mathbf{W}^{0}||} \leq \epsilon \tag{23}$$

is satisfied. If a single iteration is requested, the resulting scheme is referred to as LU-SGS by Sharov *et al.*[14] and in this case, forward and backward sweeps are simplified.

The block diagonal matrix operator is formed by assembly of the edge-based and node-based Jacobian of the advection scheme or the combination of the flux Jacobian and the spectral radius of the the flux Jacobian as advocated by Sharov *et al.*[14] In the case of the latter, the edge contributions result in a pure diagonal, however, the inclusion of boundary conditions results in a block diagonal. A block diagonal is therefore always assumed in the current implementation. The off-diagonal matrix-vector products are formed by either using the Jacobian of the advection scheme or the combination of flux Jacobian and spectral radius and is implemented either analytically or in a matrix-free form[13,14] . The LU-SGS solver is also useful, applied as a preconditioner in GMRES in the MFNK method[15] replacing ILU.

IV. Results

In this section we present results of several solver strategies for solving inviscid and viscous flows. These results can be considered as steps 1-3 discussed earlier. Eight different nonlinear solver strategies have been tested and will be illustrated by solving three supersonic flow problem; Mach 2 flow over a NACA0012 airfoil, Mach 3 inviscid flow over a blunt-wedge configuration, and finally Mach 3 laminar flow over a blunt-wedge. The solver strategies can be summarized as follows;

- MFNK/ILU(0) - This is the baseline solver. It uses the matrix-free Jacobian operator, and ILU(0) incomplete factorization of the edge-based global matrix.
- SI (10/.01) - This is the baseline semi-implicit solver, where ten inner iterations or a inner norm reduction of 0.01 is specified.
- S1 - Initially, the SI(10/.01) solver is used until a specified norm reduction $||R||/||R||_0$ is reached, then the solver switches to MFNK/ILU(0) with $\eta = 0.01$ and runs to completion.
- S2 - Initially, the MFNK/LUSGS(2/.01), $\eta = 0.1$ (where 2 sweeps are requested) is run, the at a specified norm reduction, the solver switches to MFNK/ILU(0), $\eta = 0.01$ for the completion of the run.
- S3 - The same as S2 except that $\eta = 0.1$ is used throughout, and switching is based on the number of linear iterations. If more than the specified number of iterations is taken, the solver switches to ILU(0) and if fewer than the specified number are taken, the solver switches back to LUSGS(2/.01). A buffer value defines a small overlap so the solver does not switch too often. The number of iterations is based on the cost of the LUSGS operator vs. the cost of assembling the matrix and factorizing it. At present number is determined prior to the simulation by running both preconditioners separately.
- S4 - The solver uses MFNK/ILU(0) , $\eta = 0.1$ but recomputes the preconditioner based on a specified iteration frequency (MPF), typically 5-10 iterations.
- S5 - The solver begins by using the edge matrix as the Jacobian operator with ILU(0) (referred to as deferred correction, NK/ILU(0)) and then switches to MFNK/ILU(0)$\eta = 0.01$ when a specified norm reduction is achieved.
- S6 - The solver begins with first-order spatial discretization, using the edge matrix which is consistent with first-order discretization, for the Jacobian operator and ILU(0) preconditioning, and then switches to higher-order discretization and MFNK/ILU(0) $\eta = 0.01$ preconditioning.

Strategies S1, S2, S5 and S6 are triggered with a specified binary switch. S3 is triggered by the number of linear iterations required for the particular nonlinear step, S4 is a static setting. There are many more strategies that could be realized. These were chosen based on experience running the MFNK solver on a variety of different problems.

A. Supersonic Inviscid Flow

The first problem deals with two-dimensional Mach 2 inviscid flow over a NACA0012 airfoil at zero degrees angle of attack. An O-grid ($129 \times 33 \times 2$) is used for these simulations. In each simulation a steady-state solution is sought and the solution is assumed to be converged when the L2 norm of the entire residual vector is reduced by 10^{-8} relative to the initial norm. Local time stepping is used and the CFL varies from 1 to 5000 using the adaptive formula eq. 10, with the parameter given in tables 1 and 2. Mach number contours for a typical steady-state solution are shown in figure 1. Nonlinear convergence behavior for the nonlinear solvers including the two baseline solvers and the six other strategies are shown in figure 2. Convergence times show a slight sensitivity to the adaptive CFL parameter. Convergence times are also slightly sensitive to η which is not presented. It is not the intention to find an optimal set of parameters for each solver strategy, instead efforts were made to keep the solver parameters as consistent as possible so that differences in run times would reflect on the behavior of the strategies themselves. The S4 summaries are presented along with the MFNK baseline summaries in table 1 because the S4 solver is very similar to the baseline solver. Though run times vary with MPF and f_A, clearly this strategy represents a boost in performance.

The most dramatic boost in performance is realized in strategy S1. This strategy was used by Nielsen *et al.*[16] in a similar context, however, their formulatoin of the semi-implicit solver was different. In this case, running the semi-implicit solver for just 10^{-2} reduction in the residual norm and then switching to the baseline solver, produces nearly a factor of 3 speedup! It would be reasonable to expect the run times for strategy S1 to lie somewhere in between the two baseline solver run times since the initial solver in S1 is exactly the same as SI_base1 and the final solver in S1 is exactly the same as MFNK/ILU(0)_base1, but instead, the run time for S1 is 36% of MFNK/ILU_base1 and 29% of SI_base1.

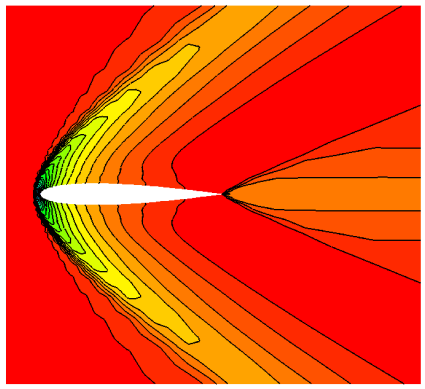


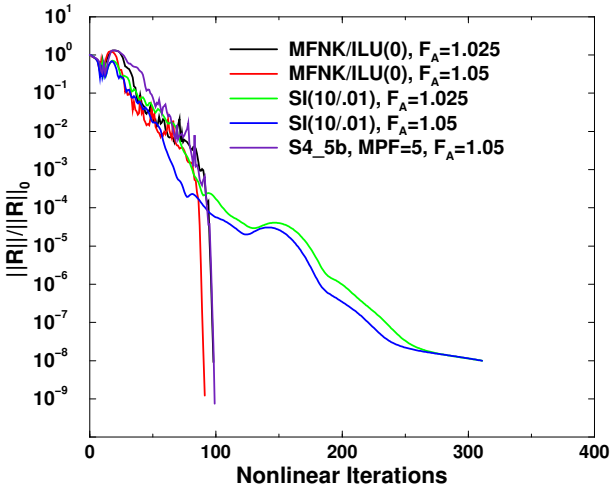
Figure 1. Mach contours for inviscid flow around a NACA0012 airfoil.

Table 1. Inviscid flow over NACA0012 airfoil at Ma=2.0.

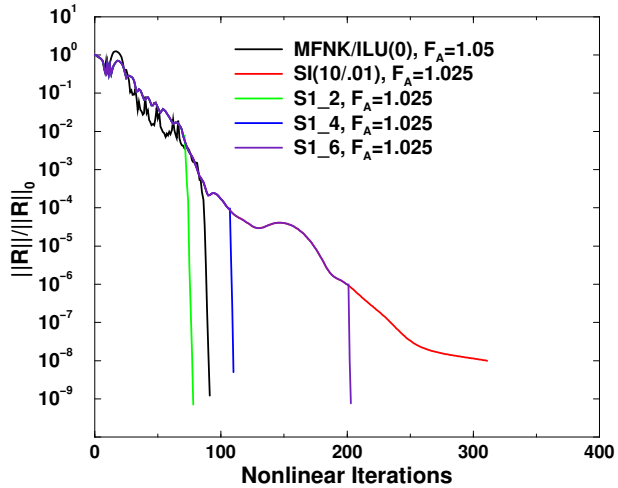
Strategy	f_A	MPF	CPU sec.
MFNK/ILU(0)_base1	1.025	1	305
MFNK/ILU(0)_base2	1.05	1	292
SI_base1	1.025		378
SI_base2	1.05		389
S4_5a	1.025	5	259
S4_5b	1.05	5	218
S4_5c	1.1	5	253
S4_10	1.2	10	249

Table 2. Inviscid flow over a NACA0012 airfoil, Ma=2.0.

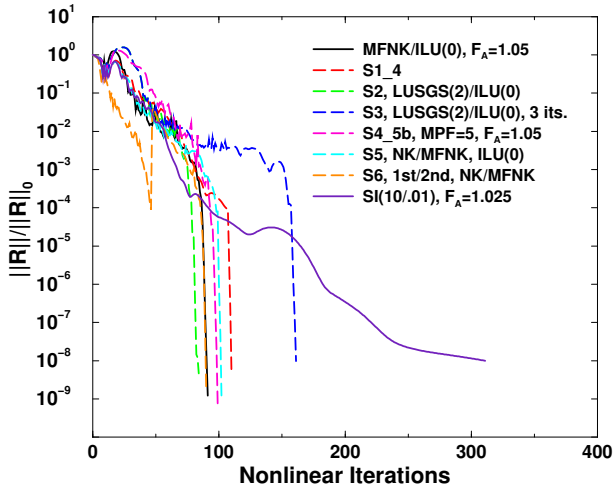
Strategy	comment	switch	**CPU** sec.
S1_2	SI(10/.01)-MFNK/ILU(0)	10^{-2}	111
S1_4	SI(10/.01)-MFNK/ILU(0)	10^{-4}	137
S1_6	SI(10/.01)-MFNK/ILU(0)	10^{-6}	248
S2	MFNK/LUSGS(2)-MFNK/ILU(0)	10^{-4}	308
S3	MFNK/LUSGS(2)-MFNK/ILU(0)	3 its.	403
S5	NK/ILU(0)-MFNK/ILU(0)	10^{-4}	211
S6	1st-2nd, NK/ILU(0)-MFNK/ILU(0) NK/ILU(0)-MFNK/ILU(0)	10^{-4}	279



(a) Nonlinear convergence behavior for; MFNK base, SI base, and S4 strategies.



(b) Nonlinear convergence behavior for S1 strategies.



(c) Nonlinear convergence behavior for S1-S6 strategies.

Figure 2. Inviscid flow over a NACA0012 airfoil at Ma=2.0.

B. Supersonic Blunt Wedge

The second problem deals with supersonic flow over a blunt wedge shape with nose radius 0.0088 m and wedge angle 5.25 degrees. The inviscid case corresponds to Mach 3 conditions at an altitude of 8 km. The temperature was 236.215 K and the pressure was 3.56516×10^4 Pa. The laminar flow problem corresponds to Mach 3 conditions at an altitude of 40 km and a Reynolds number, $Rex = 2.4 \times 10^5$. The free stream temperature and pressure for the laminar case was 250.35 K, and 2.87144×10^2 Pa respectively, and an isothermal boundary condition was applied to the blunt-wedge and the surface temperature was 300 K. Sutherland's law was assumed for viscosity and a constant Prandtl number Pr=.72 was used. The mesh for both the inviscid and laminar simulations was the same, $(81 \times 75 \times 3)$. Local time stepping was used and the CFL varies from 1 to 10^5 using the adaptive formula eq. 10, with $f_A = 1.01$.

For these simulations eigenvalue smoothing was added to the Roe flux function to reduce pressure stair stepping in the vicinity of the shock. The formulation follows Peery and one to 10^5. Imlay.[17]

Pressure contours from the solution of the inviscid and laminar cases appear in figure 3. The pressure contours show very similar patterns dominated by the presence of the shock wave. The flow is initialized with uniform flow. A detached bow shock forms at the leading edge of the blunt wedge and propagates up stream until the equilibrium location is reached. This highly transient nonlinear behavior prevents rapid convergence of the solution until the shock settles down to its final location. This behavior explains why the MFNK method performs poorly by itself. Only small incremental steps may be taken while the shock is in transit. However, each of these small steps are quite expensive compared to the relatively cheap semi-implicit iterations. Once the shock wave settles down though, MFNK with its superior convergence behavior quickly reaches a solution.

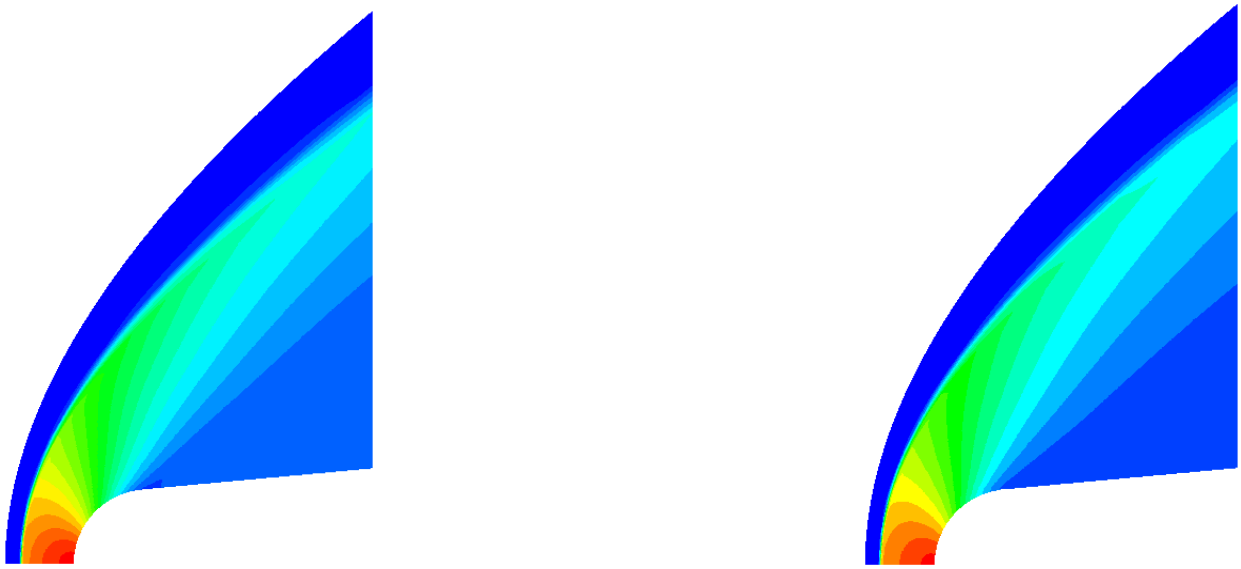
Run times for the base line MFNK and semi-implicit solver are compared to the S1 strategy, with different switch switch values, for both the inviscid and laminar problem in tables 3 and 4. Nonlinear convergence behaviors are presented in figure 4. Just as in the previous case, a dramatic boost in solver performance is realized by using strategy S1 for both the inviscid and laminar viscous cases.

Table 3. Inviscid flow over blunt wedge with Ma=3.0.

Strategy	**CPU** sec.
MFNK/ILU(0)	3,140
SI	2,776
S1_2	678
S1_3	553
S1_4	583

Table 4. Laminar flow over blunt wedge with Ma=3.0 and Rex=2.4×10^5.

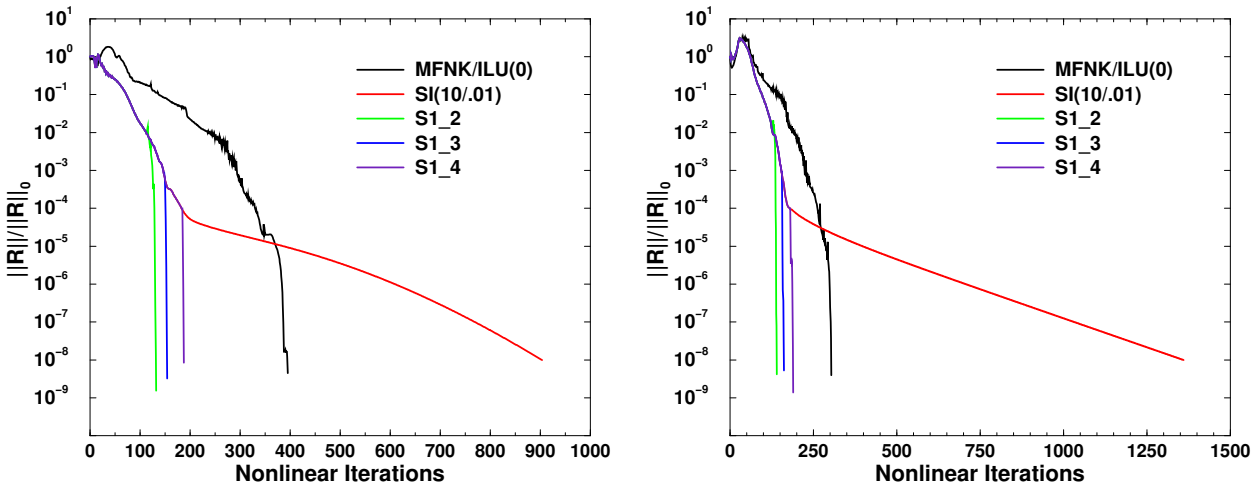
Strategy	**CPU** sec.
MFNK/ILU(0)	2,837
SI	7,835
S1_2	1,042
S1_3	1,085
S1_4	1,292



(a) Inviscid flow values range from 40,000 to 420,000.

(b) Laminar flow values range from 400 to 3,400.

Figure 3. Pressure contours for blunt wedge flow.



(a) Inviscid flow: MFNK base, SI base, and S1 strategies.

(b) Laminar flow: MFNK base, SI base, and S1 strategies.

Figure 4. Nonlinear convergence behavior for supersonic blunt wedge flow.

V. Conclusions

In this work, we have set out to design and develop intelligent nonlinear solvers with the goal in mind of using established algorithms and libraries in new ways and combinations to improve upon the performance and reliability of the solution methods. We have identified several solver strategies that hold promise and demonstrated their potential on several shock capturing problems. While no claim of optimality can be made, and minimal "tuning" was involved, dramatic performance increases have been realized when compared to the two baseline solvers. The next step toward reaching our goal is to start making these solvers more autonomous.

VI. Appendix

First-order accuracy is achieved by assuming a piecewise constant state vector within control volumes. Interface state values are taken as node values themselves,

$$\mathbf{U}_I^+ \equiv \mathbf{U}_I \tag{24}$$

$$\mathbf{U}_J^- \equiv \mathbf{U}_J \ . \tag{25}$$

For higher-order spatial accuracy, a piecewise linear variation within cells is assumed. Reconstruction of the interface states (+,-) use a multi-dimensional MUSCL extrapolation[18,19] along with the average nodal gradient. The Green-Gauss volume-averaged gradient is obtained by solving the surface integral

$$\int_{\mathcal{V}} \nabla\phi \, d\mathcal{V} = \delta\mathcal{V}\nabla\phi = \oint_{\Gamma} \phi\mathbf{n} \, d\Gamma \ .$$

The discrete form is given by the edge-based formulation, including boundary face contributions,

$$\nabla\phi_I \approx \frac{1}{\delta\mathcal{V}_I} \sum_{J\in NE(I)} \left(\frac{\phi_I + \phi_J}{2}\right) \mathbf{n}_{IJ} \left|\delta\Gamma_{IJ}\right| \tag{26}$$

$$\nabla\phi_I^b \approx \frac{1}{\delta\mathcal{V}_I} \sum_{J\in NE(I)} \left(\frac{\phi_I^b + \phi_J}{2}\right) \mathbf{n}_{IJ} \left|\delta\Gamma_{IJ}\right| + \phi_I^b \mathbf{n}_I^b \left|\delta\Gamma_I^b\right| \ . \tag{27}$$

where $\nabla\phi_I^b$ is on a boundary node. An alternative to Green-Gauss construction is weighted least-squares. Haselbacher[4] has summarized the least-squares formulation for arbitrary meshes. The three components of the gradient at each node I, can be written as a weighted sum of differences between edge IJ state data;

$$(\phi_x)_I = \sum_{J\in N(I)} \omega_{IJ}^x \left(\phi_J - \phi_I\right) \tag{28}$$

$$(\phi_y)_I = \sum_{J\in N(I)} \omega_{IJ}^y \left(\phi_J - \phi_I\right) \tag{29}$$

$$(\phi_z)_I = \sum_{J\in N(I)} \omega_{IJ}^z \left(\phi_J - \phi_I\right) \tag{30}$$

where $\nabla\phi_I = \{\phi_x, \phi_y, \phi_z\}_I^T$ denote Cartesian derivative components and $(\omega_{IJ}^x, \omega_{IJ}^y, \omega_{IJ}^z)$ denote Cartesian component weights. The gradients require that six weights be stored on each node and are assembled by looping over all edges including virtual edges as defined by Haselbacher.[4]

Left (+) and right(-) interface states are computed using MUSCL extrapolation,

$$\mathbf{U}_I^+ = \mathbf{U}_I + \Phi(\Delta_I^+, \Delta_I^-) \left(\nabla\mathbf{U}_I \cdot \frac{\Delta\mathbf{r}}{2}\right) \tag{31}$$

$$\mathbf{U}_J^- = \mathbf{U}_J - \Phi(\Delta_J^+, \Delta_J^-) \left(\nabla\mathbf{U}_J \cdot \frac{\Delta\mathbf{r}}{2}\right) \ .$$

where $\Delta\mathbf{r} = \mathbf{x}_J - \mathbf{x}_I$ is a displacement vector, and $\Phi = [0,1]$ is a gradient limiter. Gradients are limited by the van Albada limiter, similar to that used by Luo *et al.*[5] which is computed locally on each edge. The van Albada limiter is written as,

$$\Phi(a,b) = \frac{ab + |ab| + \epsilon}{a^2 + b^2 + \epsilon} \tag{32}$$

and the arguments are defined by,

$$\begin{aligned} \Delta_I^+ &= \mathbf{U}_J - \mathbf{U}_I \\ \Delta_I^- &= 2\nabla\mathbf{U}_I \cdot \Delta\mathbf{r} - (\mathbf{U}_J - \mathbf{U}_I) \\ \Delta_J^+ &= 2\nabla\mathbf{U}_J \cdot \Delta\mathbf{r} - (\mathbf{U}_J - \mathbf{U}_I) \\ \Delta_J^- &= \mathbf{U}_J - \mathbf{U}_I \ . \end{aligned} \tag{33}$$

The advection scheme that Premo currently uses is the approximate Riemann solver of Roe[20] with the entropy fix of Liou and van Leer[21] and the implementation closely follows that of Whitaker.[22]

The Roe flux function can be written as,

$$\mathbf{F}_{IJ}^c = \frac{1}{2}\left(\mathbf{F}(W_I^+, \mathbf{n}_{IJ}) + \mathbf{F}(W_J^-, \mathbf{n}_{IJ}) - |\tilde{A}|(W_J^- - W_I^+)\right)|\delta\Gamma_{IJ}| \tag{34}$$

where $A = \frac{\partial F}{\partial W}$ is the flux Jacobian matrix and $|\tilde{A}| = |A(\tilde{W}_I^+, \tilde{W}_J^-, \mathbf{n}_{IJ})|$ is evaluated using average states $(\tilde{\cdot})$ defined by Roe.[20]

In the next section, steady-state solution strategies will be developed which require the construction of Jacobian matrix. Linearization of the advective flux function with respect to the conservative variables including only edge node data (first-order) is evaluated using automatic differentiation (AD)[23,24] or a local edge-based finite difference, and can be represented as,

$$\begin{aligned} \frac{\partial F_I}{\partial \mathbf{W}_I} &= \mathbf{I}\frac{\delta\mathcal{V}}{\Delta t} + \frac{1}{2}\left(A(W_I, \mathbf{n}_{IJ}) + |\tilde{A}| - \frac{\partial|\tilde{A}|}{\partial \mathbf{W}_I}(W_J - W_I)\right)|\delta\Gamma_{IJ}| \\ \frac{\partial F_I}{\partial \mathbf{W}_J} &= \frac{1}{2}\left(A(W_J, \mathbf{n}_{IJ}) - |\tilde{A}| - \frac{\partial|\tilde{A}|}{\partial \mathbf{W}_J}(W_J - W_I)\right)|\delta\Gamma_{IJ}| \ . \end{aligned} \tag{35}$$

The Jacobian operators in the semi-implicit solver are based upon a modified flux function, here referred to as "symmetric dissipation" (**SD**), that was suggested by Luo *et al.*[15] Here, artificial viscosity is based on the magnitude of the spectral radius of the flux Jacobian,

$$\mathbf{F}_{IJ} = \frac{1}{2}\left(F(W_I, \mathbf{n}_{IJ}) + F(W_J, \mathbf{n}_{IJ}) - |\lambda_{IJ}|(W_J - W_I)\right)|\delta\Gamma_{IJ}| \ . \tag{36}$$

The Jacobian terms are,

$$\begin{aligned} \frac{\partial F_I^{\mathbf{SD}}}{\partial \mathbf{W}_I} &= \mathbf{I}\frac{\delta\mathcal{V}}{\Delta t} + \frac{1}{2}\left(A(W_I, \mathbf{n}_{IJ}) + |\lambda_{IJ}|\mathbf{I}\right)|\delta\Gamma_{IJ}| \\ \frac{\partial F_I^{\mathbf{SD}}}{\partial \mathbf{W}_J} &= \frac{1}{2}\left(A(W_J, \mathbf{n}_{IJ}) - |\lambda_{IJ}|\mathbf{I}\right)|\delta\Gamma_{IJ}| \end{aligned} \tag{37}$$

where,

$$|\lambda_{IJ}| = |\mathbf{u}_{IJ} \cdot \mathbf{n}_{IJ}| + c_{IJ} + \frac{2\mu}{\rho|\mathbf{n}_{IJ} \cdot \Delta\mathbf{r}_{IJ}|} \tag{38}$$

is the spectral radius of the flux Jacobian matrix and $\mathbf{I}$ is the identity matrix.

Evaluating the viscous flux vector requires reconstructing primitive variable gradients at edge midpoints. Gradients are reconstructed by blending edge node differences with nodal gradients,

$$\nabla U_{IJ} = \frac{(\nabla U_I + \nabla U_J)}{2} + \left(U_J - U_I - \frac{(\nabla U_I + \nabla U_J)}{2} \cdot \Delta\mathbf{r}_{IJ}\right)\frac{\Delta\mathbf{r}_{IJ}}{|\Delta\mathbf{r}_{IJ}|^2} \ . \tag{39}$$

Viscosity, conductivity and velocity components appearing in the energy dissipation terms are evaluated at the I and J nodes and then averaged. The support for these gradients include distance-two nodes.

Therefore, exact discrete Jacobians cannot be contained on an edge graph. Instead, only the node differences are included in the analytic representation of the Jacobian terms,

$$\nabla U_{IJ} \approx (U_J - U_I) \frac{\Delta \mathbf{r}_{IJ}}{|\Delta \mathbf{r}_{IJ}|^2} \ . \tag{40}$$

This results in a very efficient algorithm for evaluating the terms but is consistent with eq. 39 only on Cartesian grids.

In all cases, whether it be the evaluation the spatial residual equations eq. 4, or the nodal gradients or an approximation to the Jacobian operator, assembly only looping over edges and nodes. Boundary fluxes are assembled by looping over exterior faces of the primitive finite element mesh. Additional details can be found in Smith *et al.*[10]

References

[1]Kosko, B., *Fuzzy Engineering*, Prentice-Hall, Engelwood Cliffs, NJ, 1997.

[2]Manzano, L., Lassaline, J., Wong, P., and Zingg, D., "A Newton-Krylov Algoritm for the Euler Equations Using Unstructured Grids," *AIAA Paper 2003-0274*, January 2003.

[3]Edwards, H. C. and Stewart, J. R., "SIERRA : A Software Environment for Developing Complex Multiphysics Applications," *Proceedings of First MIT Conference on Computational Fluid and Solid Mechanics*, edited by K. J. Bathe, Elsevier Scientific, 2001.

[4]Haselbacher, A. and Blazek, J., "Accurate and Efficient discretization of Navier-Stokes Equations on Mixed Grids," *AIAA Journal*, Vol. 38, No. 11, November 2000, pp. 2094–2102.

[5]Luo, H., Baum, J. D., and Lohner, R., "Edge-Based Finite Element scheme for the Euler Equations," *AIAA Journal*, Vol. 32, No. 6, June 1994, pp. 1180–1190.

[6]Eisenstat, S. and Walker, J., "Choosing the forcing terms in an inexact Newton Method," *SIAM Journal of Scientific Computing*, Vol. 17, 1996, pp. 16–32.

[7]Saad, Y. and Schultz, M., "GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM Journal of Scientific and Statistical Computing*, Vol. 7, 1986, pp. 856–869.

[8]Brown, P. and Saad, Y., "Hybrid Krylov methods for nonlinear systems of equations," *SIAM Journal of Statistical Computing*, Vol. 11, 1990, pp. 450–481.

[9]Knoll, D. and Keyes, D., "Jacobian-Free Newton-Krylov methods: a survey of approaches and applications," *Journal of Computational Physics*, Vol. 193, 2004, pp. 357–397.

[10]Smith, T. M., Hooper, R. W., Ober, C. C., Lorber, A. A., and Shadid, J. N., "Comparison of Operators for Newton-Krylov Method for Solving Compressible Flows on Unstructured Meshes," *AIAA Paper 2004-0743*, January 2004.

[11] "Trilinos/AztecOO: Object-Oriented Aztec Linear Solver Package," http://software.sandia.gov/Trilinos/doc /aztecoo/doc/html/index.html.

[12] "NOX & LOCA: Object-Oriented Nonlinear Solver And Continuation Packages," http://software.sandia.gov/nox/.

[13]Chen, R. and Wang, Z., "Fast, Block Lower-Upper Symmetric Gauss-Seidel Scheme for Arbitrary Grids," *AIAA Journal*, Vol. 38, No. 12, 2000, pp. 2238–2244.

[14]Sharov, D., Luo, H., and Baum, J. D., "Implementation of Unstructured Grid GMRES+LU-SGS Method on Shared-Memory, Cache-Based Parallel Computers," *AIAA Paper 2000-0927*, January 2000.

[15]Luo, H., Baum, J. D., and Lohner, R., "A fast, matrix-free implicit method for compressible flows on unstructured grids," *Journal of Computational Science*, Vol. 146, 1998, pp. 664–690.

[16]Nielsen, E. J., Anderson, W. K., Walters, Robert, W., and Keyes, D. E., "Application of Newton-Krylov Methodology to a Three-Dimensional Unstructured Euler Code," *AIAA Paper 1995-1733*, 1995.

[17]Peery, K. and Imlay, S., "Blunt-Body Flow Simulations," *AIAA Paper 88-2904*, January 1988.

[18]van Leer, B., "Towards the Ultimate Conservation Difference Scheme III. Upstream-Centered Finite-Difference Schemes for Ideal Compressible Flow," *Journal of Computational Physics*, Vol. 23, 1977, pp. 263–275.

[19]Barth, T., "The Design and Application of Upwind Schemes on Unstructured Meshes," *AIAA Paper 89-0366*, January 1989.

[20]Roe, P. L., "Approximate Riemann Solvers, Parameters Vectors and Difference Schemes," *Journal of Computational Physics*, Vol. 43, 1981, pp. 357–372.

[21]Liou, M.-S. and van Leer, B., "Choice of Implicit and Explicit operators for the upwind differencing method," *AIAA Paper 1988-0624*, January 1988.

[22]Whitaker, D. L., "Three-Dimensional Unstructured Grid Euler Computations Using a Fully-Implicit, Upwind Method," *AIAA Paper 93-3337*, January 1993.

[23]Berggren, M., "Discrete adjoint equations and gradients for edge-based discretizations of the Euler equations," Tech. rep., Sandia National Laboratories, 2003, Unpublished.

[24]Griewank, A., *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, SIAM, 2000.