An Exact Dual Adjoint Solution Method for Turbulent Flows on Unstructured Grids

Eric J. Nielsen
NASA Langley Research Center
Hampton, VA 23681

James Lu
Massachusetts Institute of Technology
Cambridge, MA 02139

Michael A. Park
NASA Langley Research Center
Hampton, VA 23681

David L. Darmofal
Massachusetts Institute of Technology
Cambridge, MA 02139
An Exact Dual Adjoint Solution Method for Turbulent Flows on Unstructured Grids

Eric J. Nielsen†, James Lu‡, Michael A. Park‡, and David L. Darmofal§

An algorithm for solving the discrete adjoint system based on an unstructured-grid discretization of the Navier-Stokes equations is presented. The method is constructed such that an adjoint solution exactly dual to a direct differentiation approach is recovered at each time step, yielding a convergence rate which is asymptotically equivalent to that of the primal system. The new approach is implemented within a three-dimensional unstructured-grid framework and results are presented for inviscid, laminar, and turbulent flows. Improvements to the baseline solution algorithm, such as line-implicit relaxation and a tight coupling of the turbulence model, are also presented. By storing nearest-neighbor terms in the residual computation, the dual scheme is computationally efficient, while requiring twice the memory of the flow solution. The scheme is expected to have a broad impact on computational problems related to design optimization as well as error estimation and grid adaptation efforts.

Introduction

The field of computational fluid dynamics (CFD) is increasingly important in the design and validation of new aerodynamic concepts. The use of computational tools can greatly reduce the need for more costly alternatives, such as wind tunnel experiments or flight testing. CFD techniques based on the Navier-Stokes equations have matured into reliable tools for the analysis of complex geometries. However, design optimization using these high-fidelity methods has been greatly inhibited by their relatively high expense and lack of robustness.

In an effort to alleviate the high cost of gradient-based design methodologies, recent work has focused on the use of adjoint methods. 1-20 Adjoint techniques generally fall into one of two categories based on the order in which the discretization and differentiation processes are performed. The two approaches are termed continuous or discrete adjoint methods. For a single output or constraint function, these schemes allow the computation of design sensitivities at the cost of solving a single additional linear problem and a subsequent computation of a matrix-vector product dimensioned by the number of design variables.

In addition to its role in design optimization problems, the solution of the adjoint equation can be used to provide error estimates for an output of engineering interest, as well as grid adaptation information that can be used to improve solution accuracy. 21-29 Traditional methods for grid adaptation have typically relied on ad hoc feature-based quantities. 30-33 In these approaches, features such as shocks, boundary layers, and other regions of interest are characterized by large gradients or curvatures in the solution. The adaptation algorithm then attempts to improve the solution by adding additional grid points in these regions. Unfortunately this approach can yield grids of unwieldy dimensions and even incorrect results. 33 The local flow feature of interest is often over-resolved, while smooth regions of the flowfield are essentially neglected.

The adjoint approach to grid adaptation seeks to minimize the uncertainty or error in some specified output function. In this approach, a local adaptation parameter is obtained by combining flow and adjoint solutions, where the nonlinearities in these solutions are weighted with the local residual error. This adaptation technique implicitly targets the flow features having the highest impact on the output of interest. The test cases examined in Refs. 24, 25, and 27-29 clearly demonstrate the potential of using such an approach to grid adaptation. Comparisons with feature-based techniques are shown and results equivalent to those of uniform grid-refinement studies are obtained at a fraction of the cost. The process terminates when a user-specified error tolerance is achieved.

Unfortunately the solution of the adjoint system of equations for realistic problems has proven to be a formidable task. In Ref. 4, a Krylov method was used to solve the adjoint system for turbulent flows using relatively coarse grids over simple geometries. This work successfully demonstrated the accuracy of the method. However, the computational time required to solve the equations was as much as ten times the cost of the analysis problem, and the scheme failed to converge for many problems. By employing a more extensive preconditioner for the Krylov algorithm, Ref. 5 demonstrated improved performance. However, this preconditioning strategy was shown to require approximately five times the memory of the baseline analysis scheme. This approach proved infeasible on available computers for large-scale problems that require grids containing several million mesh points.

The focus of the current work is to develop a new solution algorithm for the discrete adjoint system described in Refs. 4 and 5. The work is largely based on the recent contributions of Giles et al., 18-20 in which adjoint solutions for the Euler and Navier-Stokes equations are computed by using an explicit Runge-Kutta scheme combined with multigrid using an exact dual method. Elliott recently used a similar approach for two-dimensional turbulent flows on overset structured grids using multigrid and approximate factorization.

Improvements to the implicit solution technique of Refs. 35 and 36 are described, and an exact dual scheme is developed for the resulting algorithm. Care has been taken to ensure that the implementation yields identical values for linear functionals at each time step, thereby guaranteeing identical asymptotic convergence rates for the primal and dual systems. Results are shown for several small test problems and two large-scale configurations. Convergence of a typical cost function and its derivatives are examined, and computational speed and memory requirements are discussed.

Flow Solution Method

The governing equations are the three-dimensional Reynolds-averaged Navier-Stokes equations. For turbulent flows, the one-equation model of Spalart and Allmaras 38 is used. The flow solver used in the current work is described at length in Refs. 1, 35, and 36. The code uses an implicit, upwind, finite-volume discretization in which the dependent variables are stored at the mesh vertices.
Inviscid fluxes at cell interfaces are computed by using the upwind schemes of Roe or Van Leer. Viscous fluxes are formed by using an approach equivalent to a central-difference Galerkin procedure. For steady-state flows, temporal discretization is performed by using a backward Euler time-stepping scheme. A highly scalable parallelization scheme is achieved through domain decomposition and MPI communication.

An approximate solution of the linear system of equations formed at each time step is obtained through several iterations of a point-iterative scheme in which the nodes are updated in an even-odd fashion, resulting in a Gauss-Seidel-type method. This scheme is augmented with a line-relaxation algorithm in the current work.

In Refs. 1–5, 35, and 36, the turbulence model is integrated all orders of magnitude over the fast and turbulence equations has been solved by using a preconditioned GMRES algorithm. The focus of the current work is an alternate solution method based on an exact dual formulation.

Tightly Coupled Turbulence Model

The loosely coupled implementation of the turbulence model in the baseline solver was originally chosen for its convenience in accommodating additional models as they became available. Furthermore, the loose formulation allows for straightforward implicit enforcement of positivity on the eddy viscosity by using M-type matrices and positive operators as described in Ref. 38.

Although the loose formulation has proven satisfactory for engineering-level analysis, it often results in stalled convergence or limit-cycle oscillations that can be detrimental to subsequent adjoint computations. In the current work, a tightly coupled algorithm is used to obtain more robust convergence behavior. The scheme includes the linearizations of the governing flow equations with respect to the turbulence model dependent variable $\nu'$, as well as the linearizations of the turbulence model with respect to the conserved variables $Q$, in the computation of the solution updates $\Delta Q$ and $\Delta \nu$. The approach taken in Ref. 38 to guarantee positivity on $\nu$ becomes prohibitively difficult to impose for the tightly coupled system, so that an update clip at $\nu = 0$ is necessary to preclude non-physical behavior as the solution rapidly develops from its initial freestream conditions. This procedure occasionally admits transients in the early stages of a computation which may result in divergence of the solution; these transients are overcome by using small time steps as the initial solution sets up. Although not discussed in the present paper, limited success has been achieved through modifications as suggested in Ref. 41, in which an analysis of the linearized form of the turbulence source terms suggests a similar addition to the diagonal elements of the system.

To demonstrate the effect of coupling on solution convergence, transonic flow over an ONERA M6 wing is computed by using both the loosely and tightly coupled formulations. The grid is shown in Fig. 1 and contains 359,536 nodes and 2,074,955 tetrahedra. The freestream Mach number is 0.84, the angle of attack is 3.06°, and the Reynolds number is 5 million based on the mean aerodynamic chord. For both cases, the CFL number for the flow and turbulence equations has been linearly ramped from 10 to 200 over the first 50 iterations. The convergence histories for density and turbulence for the two solution methods are shown in Fig. 2. The loosely coupled scheme results in stalled convergence, whereas the tightly coupled scheme steadily reduces the residuals by six orders of magnitude over the course of the computation. Similar results have been observed in other cases, and the tightly coupled algorithm is employed for the remainder of the current work.

![Figure 1. Surface grid for viscous ONERA M6 wing.](image1)

![Figure 2. Effect of turbulence model coupling on ONERA M6 flow solution.](image2)

Line-Implicit Relaxation Scheme

When used in conjunction with the baseline point-implicit scheme used for the flow solution, a line-implicit scheme can result in improved convergence rates for viscous flows. The benefits of line-implicit techniques on highly stretched grids are well-known. The central idea is that the coupling of the discrete equations is considerably higher in the direction normal to the grid stretching, so that a relaxation scheme can be constructed to more efficiently solve the equations associated with lines in these directions.

**Line Construction Method**

On structured grids the line-relaxation direction is typically well-defined because a grid coordinate direction can be readily employed. For unstructured grids, this inherent direction is not available and must be constructed prior to performing any computations. In Ref. 44, implicit lines have been constructed based on neighboring prismatic and hexahedral elements within the boundary layer. In the current work, only tetrahedral elements are employed; thus, an alternative line construction strategy must be used.

To construct a line for implicit relaxation through the boundary layer, an initial point is chosen on a viscous boundary surface. A surface normal is constructed at this point, and the direction of edges connected to this node are compared with the normal using an inner product. The edge with the maximum positive inner product is selected to form the first line segment. The surface normal is
then replaced with the direction of the chosen edge, and the process is repeated at the endpoint of successively selected edges, forming the line along which to relax. The process is terminated when the ratio of the length of the longest edge connected to the current node to that of the selected edge falls below a predefined value; a value of 5 is used in the current work. This criterion serves to identify the "inviscid" region of the grid, where elements revert to an isotropic distribution. Construction of a line is also terminated if an edge direction differing by less than 20 degrees from the previous segment cannot be found. This second stopping criterion ensures that line segments remain normal to the original boundary surface. The result of applying this line construction algorithm to a simple wingtip geometry is shown in Fig. 3. A direction suitable for line-implicit relaxation through the boundary layer has been formed at each boundary node, such that approximately 30 grid points are contained in each line.

Another important consideration is the mesh partitioning strategy for parallel processing. To efficiently perform relaxation along any given line, the line should wholly lie within a grid partition. To avoid partition boundaries cutting across relaxation lines, edge weighting is employed in the partitioning phase. This procedure minimizes the number of implicit lines that are split across partition boundaries. In the event that an implicit line is cut by the mesh partitioning, the line is terminated at the partition boundary; no attempt is made to perform line-relaxation across processors. Finally, a multiconstraint version of the partitioning algorithm is used to ensure that load balancing is achieved within the line-implicit and point-implicit regions of the grid.

Line Relaxation Algorithm

Once the linear system of equations at the current time step has been assembled, the unknowns associated with each implicit line are computed exactly by using Gaussian elimination of a local block-tridiagonal system of equations. This procedure is repeated for a series of sweeps over the implicit lines; the initial decomposition of the coefficient matrix is stored so that subsequent sweeps merely consist of a forward/backward substitution procedure. Once a suitable level of convergence is obtained for the unknowns, the remainder of the domain is relaxed by using several sweeps of the baseline point-implicit scheme; adjacent unknowns determined by the line-implicit scheme are taken as known quantities and moved to the right-hand side of the equations.

To demonstrate the benefits of the line-implicit scheme, fully turbulent flow over the wing-body configuration shown in Fig. 4 is computed on twenty-two 2.2 GHz Pentium IV processors using the baseline point-implicit scheme as well as the line-implicit algorithm. The freestream Mach number is 0.75, the angle of attack is 0°, and the Reynolds number is 3 million based on the mean aerodynamic chord. The grid contains 1,641,452 nodes and 9,650,684 tetrahedra, and the line construction algorithm places 1,069,238 nodes in the line-implicit region. For this test, 15 sweeps through the line- and point-implicit regions are used for both computations. Convergence of the density and turbulence residuals for both solution schemes is shown in Fig. 5. The line-implicit strategy ultimately results in a computational savings of approximately 20% over the baseline algorithm. As shown in Fig. 6, lift and drag more rapidly approach their steady-state values with the line-implicit scheme. This behavior is attributed to the rapid development of the boundary layer region, and has been observed in a number of cases. For comparison, the results are plotted in Fig. 7 against experimental values from Ref. 45. Similar to many of the computations reported in Ref. 48, the lift is slightly overpredicted; however, the correlation with the experimental drag polar is good.

Derivation of the Dual Algorithm

Consider the following form of the steady-state nonlinear governing equations, where \( \mathbf{D} \) and \( \mathbf{Q} \) represent the vector of design variables and the corresponding dependent variables, respectively, and \( R_2 \) represents a second-order accurate discretization of the spatial residual.

\[
R_2(\mathbf{Q}, \mathbf{D}) = 0. \tag{1}
\]

Note that in practice, there is also an implicit dependency on the computational grid; the current approach includes these terms, however they are omitted here to simplify the underlying analysis.
An iterative algorithm based on a backward Euler integration scheme for the solution of the nonlinear system of equations given by Eq. 1 can be written as

$$\frac{V}{\Delta t}((Q^{n+1} - Q^n) + R_2(Q^{n+1}, D) = 0,$$  \hspace{1cm} (3)

where $\Delta Q^n = Q^{n+1} - Q^n$. Note the iteration superscript $n$ is omitted from the Jacobian matrix $\frac{\partial R_2}{\partial Q}$ for clarity; unless otherwise noted, these derivatives will all be evaluated at time step $n$.

Note that because the Jacobian used in Eq. 3 is an exact linearization of $R_2$, then for an infinite time step $\Delta t$, the iterative scheme corresponds to Newton’s method, and $Q^{n+1}$ will converge quadratically to the steady-state value $Q^*$ corresponding to $D$. However, the use of an exact Jacobian $\frac{\partial R_2}{\partial Q}$ is often prohibitively expensive for realistic problems, so that an approximate form of Eq. 3 is typically used:

$$\left(\frac{V}{\Delta t} + \frac{\partial R_2}{\partial Q}\right)\Delta Q^n + R_2(Q^n, D) = 0,$$  \hspace{1cm} (4)

where the exact Jacobian has been replaced with a linearization of some first-order approximation to the spatial residual. The drawback to such an approach is that the asymptotic convergence of the algorithm becomes linear in nature.

Equation 4 is a linear system of equations for $\Delta Q^n$, which in principle can be solved exactly. In practice, however, the system is usually solved approximately by using an iterative method. Therefore, Eq. 4 can be restated as

$$\Delta Q^n + P R_2(Q^n, D) = 0,$$  \hspace{1cm} (5)

where $P$ is an approximation to the inverse of $\frac{V}{\Delta t} + \frac{\partial R_2}{\partial Q}$, typically achieved through some iterative scheme such as a Gauss-Seidel iteration. In this manner, the asymptotic rate of convergence is governed by the largest eigenvalue of the matrix $I - P\frac{\partial R_2}{\partial Q}$.

### Direct Differentiation Algorithm

With minor modifications, the iterative algorithm outlined above can be used to determine the sensitivity derivatives of an output function $f = f(Q^*, D)$ with respect to a design variable. Application of the chain rule yields the following:

$$\frac{df}{dD} = \frac{\partial f}{\partial Q^*} \frac{\partial Q^*}{\partial D} \frac{df}{dQ^*}.$$  \hspace{1cm} (6)

Linearizing Eq. 1 with respect to $D$ gives the linear residual equations for the sensitivity derivatives $\frac{\partial Q^*}{\partial D}$:

$$\frac{\partial R_1}{\partial Q} \frac{\partial Q^*}{\partial D} + \frac{\partial R_2}{\partial D} = 0.$$  \hspace{1cm} (7)

Applying the algorithm of Eq. 3 to this expression gives an iterative scheme for $\frac{\partial Q^*}{\partial D}$:

$$\left(\frac{V}{\Delta t} + \frac{\partial R_2}{\partial Q}\right)\Delta Q^n + \frac{\partial R_1}{\partial Q} \frac{\partial Q^*}{\partial D} + \frac{\partial R_2}{\partial D} = 0.$$  \hspace{1cm} (8)

Finally, by replacing the exact Jacobian with the approximate Jacobian as before and performing an approximate inverse gives the final form of the direct differentiation iterative algorithm for the sensitivity derivatives $\frac{\partial Q^*}{\partial D}$:

$$\Delta Q^n + P R_2(Q^n, D) = 0.$$  \hspace{1cm} (9)

Note the asymptotic rate of convergence is again dictated by the largest eigenvalue of $I - P\frac{\partial R_2}{\partial Q}$, and therefore will be equivalent to that of the scheme used to attain $Q^*$. After $N$ iterations of Eq. 9, the derivative of the output $f$ with respect to the design variables $D$ may be approximated as

$$\frac{df}{dD} = \frac{\partial f}{\partial Q^*} \left(\frac{\partial Q^*}{\partial D}\right)^N + \frac{\partial f}{\partial D}.$$  \hspace{1cm} (10)
Exact Dual Algorithm

In this section, an iterative solution of the dual problem is constructed in a manner which guarantees the functional estimate is identical at every iteration to that obtained from the direct differentiation algorithm. Using an adjoint approach, the sensitivity derivative of the output can be written as

\[ \frac{df}{dD} = -\Lambda^T \frac{\partial R_2}{\partial Q} + \frac{\partial f}{\partial D}. \]  

(11)

where \( \Lambda \) is an adjoint variable that satisfies

\[ \frac{\partial R_2}{\partial Q}^T \Lambda - (\frac{\partial f}{\partial Q})^T = 0. \]  

(12)

Since Eq. 12 is independent of the vector \( D \), the adjoint approach is particularly attractive for aerodynamic design problems with a large number of design variables and relatively few constraints. By examining Eqs. 7 and 12, it can be seen that

\[ \frac{\partial f}{\partial Q^*} = -\Lambda^T \frac{\partial R_2}{\partial D}. \]  

(13)

Therefore, the values of \( df/dD \) calculated from Eqs. 6 and 11 are identical. However, since an iterative algorithm is used to estimate \( \partial Q^*/\partial D \), this relationship will not hold in general. Following Giles’ exact dual approach, an iterative adjoint solution algorithm is sought which satisfies

\[ \frac{\partial f}{\partial Q^*} = -\Lambda^T \frac{\partial R_2}{\partial D}. \]  

(14)

where \( N \) is some iteration at which the equality is to be enforced. However, the resulting exact dual algorithm will not depend on \( N \), and the expression given by Eq. 14 will be valid at every iteration. In this sense, the adjoint iterative algorithm will be exactly dual to the direct differentiation algorithm.

Introducing Lagrange multipliers, the left-hand side of Eq. 14 after \( N \) iterations can be written as

\[ \frac{\partial f}{\partial Q^*} \left( \frac{\partial Q^*}{\partial D} \right)^N = \frac{\partial f}{\partial Q^*} \left( \frac{\partial Q^*}{\partial D} \right)^N \]  

(15)

\[ -\sum_{n=0}^{N-1} (\Lambda^n)^T \left\{ \frac{\partial Q^*}{\partial D} \right\} + \frac{\partial f}{\partial Q^*} \left( \frac{\partial Q^*}{\partial D} \right)^N \]  

Then, by defining

\[ \Lambda^n = \sum_{m=n}^{N-1} P^n \Lambda^{n+1}. \]  

(16)

and performing the manipulations shown in the appendix, the exact duality condition requires that

\[ \begin{align} 
\Lambda^n &+ P^n \left[ \frac{\partial R_2}{\partial Q}^T \Lambda^n - \left( \frac{\partial f}{\partial Q} \right)^T \right] = 0 
\end{align} \]  

(17)

with an initial condition \( \Lambda^0 = 0 \). As mentioned previously, Eq. 17 is independent of \( N \) and thus the exact duality condition holds for all values of \( N \). This condition guarantees an asymptotic rate of convergence for the dual problem which is governed by the largest eigenvalue of \( I - P^n \left[ \frac{\partial R_2}{\partial Q} \right] \) and is therefore ultimately equivalent to that of the solution for \( Q^* \).

Relationship Between \( P \) and \( P^T \)

In the current work, the approximate inverse \( P \) is the result of applying multiple iterations of a fixed-point method, which can be written in the following manner:

\[ Mx^{j+1} = b + Nx^j \]  

(18)

or

\[ x^{j+1} = M^{-1}(b - Ax^j). \]  

(19)

where \( A = M - N \). Here, \( M \) is some matrix that is easily invertible and approximates \( A \).

Given some initial estimate \( x^0 \), after \( J \) iterations the scheme results in the following:

\[ x^{J+1} = [I - (M^{-1}J)^{-1}A^{-1}][b - Ax^0]. \]  

(20)

Therefore, with respect to Eq. 9, the approximate inverse \( P \) using \( J \) iterations of the fixed-point method is

\[ P = I - (M^{-1}J)^{-1}A^{-1}. \]  

(21)

Forming the transpose of \( P \) yields

\[ P^T = I - A^{-T}[(M^{-1}N)^T]'. \]  

(22)

Then, by using the algorithm given by Eq. 5, the adjoint solve

\[ y^{j+1} = M^{-T}(g - A^Ty^j). \]  

(24)

For a point-Jacobi scheme, the transpose operator applied to the iteration matrix \( M \) is of no consequence, other than to imply that the diagonal blocks are to be transposed. However, for a Gauss-Seidel iterative method such as the one used in the current work, the transpose operation transforms an upper-triangular iteration matrix into a lower-triangular form, and vice versa. Therefore, the transpose implies that even-odd sweeps through the system of equations must be performed in a reverse manner. The same argument holds for the line-implicit operator used to relax the boundary layer region, although the solution within each line can be obtained in the same manner as the baseline scheme because the inversion is exact for the set of local equations within the line.

Implementation of the Dual Algorithm

The majority of the effort associated with developing an adjoint solver is in forming the exact linearizations of the second-order residual. For the codes used in the current study, this differentiation has been previously performed and the accuracy has been established in Refs. 1, 3, and 5. The implementation of the exact dual algorithm is primarily a high-level task that involves the manipulation of existing routines; this process is made easier through the programming practices established in Ref. 49.

After the flow solution has completed a series of \( n \) time steps using the algorithm given by Eq. 5, the adjoint solve can be performed in an analogous fashion according to Eq. 17. Note that the dual iterative scheme is essentially identical to the baseline analysis scheme so that little extra coding is required.

According to Eq. 17, the exact linearization of the second-order spatial residual is required for the adjoint computation. However, since these terms only appear in a matrix-vector product, they can be formed on a term-by-term basis and therefore do not require the extensive storage typically associated with the higher-order stencil. The transpose of the first-order Jacobian matrix \( \left( \frac{\partial^2 f}{\partial R/\partial Q} \right) \) is used to solve the system of adjoint equations, so that the storage required by the exact dual scheme is roughly equivalent to that of the baseline analysis code. This requirement is in contrast to the solution algorithm outlined in Ref. 5, in which the complete linearization of the second-order residual \( \left[ \frac{\partial R/\partial Q} \right] \) was used as a preconditioner. The terms arising from this larger stencil resulted in a memory requirement approximately
five times that of the scheme outlined in Refs. 35 and 36. Finally, since the Jacobian matrix $\frac{\partial R}{\partial Q^*}$ is constant for an adjoint computation, the block and tridiagonal-block decompositions used for the point- and line-implicit iterations need only be performed once and stored.

The solution strategy outlined in Refs. 1 and 4 required a frequent computation of the form $[\partial R/\partial Q^*] \Delta \lambda$ for the Krylov method being used. The viscous terms arising from the nearest-neighbor discretization were stored, so that only the inviscid terms involving the larger stencil were recomputed for each new Krylov vector. In the current work, an analogous approach can be used in the formation of the adjoint residual component $[\partial R/\partial Q^*] \Delta \lambda$ if additional memory is available. Furthermore, if sufficient memory is available for the higher-order terms as used for the preconditioner in Ref. 5, the complete linearization may be stored. In this manner, the residual update at time step $n$ reduces to an explicit matrix-vector product. Since all terms comprising the residual are stored in this approach, the solution time for the adjoint problem could be reduced considerably. Solution times and memory requirements for each of these strategies will be shown in a subsequent section.

**Validation Cases**

A series of small test cases involving inviscid, laminar, and turbulent flow over the ONERA M6 wing geometry is presented to verify that the exact dual algorithm has been implemented correctly. Each of the viscous tests has been run using the line-implicit relaxation scheme. For each example, the property of Eq. 14 has been verified to machine accuracy, and the asymptotic convergence rates for density (and turbulence where appropriate) are shown to be equal to that of the corresponding adjoint equations. For visual clarity, the convergence histories of the momentum and energy equations and their adjoint counterparts are not shown in the figures that follow, but have been found to closely reflect that of the density equation. The cost function for each test case is based on lift, and the CFL number for the flow solution has been linearly ramped from 10 to 200 over the first 50 iterations. Unless otherwise stated, each of the computations shown here has been performed on sixteen 2.2 GHz Pentium IV processors.

**Inviscid Wing**

The first test case is inviscid flow, with a freestream Mach number equal to 0.84 and an angle of attack of $2^\circ$. The grid for this test contains 357,900 nodes and 2,000,034 tetrahedra. The convergence histories of the density equation for the flow and adjoint solutions are shown in Fig. 8; the convergence rates are equivalent.

**Laminar Wing**

The first viscous case is for laminar flow with a freestream Mach number of 0.3, an angle of attack of $2^\circ$, and a Reynolds number of 1000 based on the mean aerodynamic chord. The grid shown in Fig. 1 is used for this example. The density convergence histories for the flow and adjoint solutions are shown in Fig. 8, and the convergence rates are identical.

**Turbulent Wing**

The final validation case is for turbulent flow over the ONERA M6 wing on the same grid that was used in the previous example. For this case, the freestream Mach number is 0.84, the angle of attack is $3.06^\circ$, and the Reynolds number is 5 million based on the mean aerodynamic chord. The convergence histories for the flow and adjoint equations shown in Fig. 9 exhibit similar asymptotic rates.

To further demonstrate the asymptotic nature of the flow and adjoint solution algorithms, the turbulent flow test case is also used to examine the convergence of a cost function and its derivatives. For this test, the wing has been parameterized by using the method of Ref. 50. Fig. 10 shows the error in the lift coefficient as the flow solution converges; it also shows the error in the derivatives of lift with respect to freestream Mach number, angle of attack, and several shape design variables at the midspan location as the adjoint solution converges. For this case, the error is defined as the differ-
ence between the current value and the final converged result. The
errors are reduced at a similar rate for each computation.

The turbulent flow test case is also used to evaluate the lineariza-
tion storage strategies described above. Tests are performed on six-
teen 250 MHz R10000 processors of a Silicon Graphics Origin
2000 system to simplify memory monitoring and to eliminate the
effects of network traffic. The turbulent flow validation case has
been repeated using each of the storage strategies; memory and
wallclock statistics are shown in Table 1.

Recomputing all of the necessary linearizations for the adjoint
residual yields a memory requirement roughly equivalent to that of
the flow solution, while the wallclock time is approximately 17%
less. The discrepancy in solution times can be attributed to several
factors. Forming the flux Jacobians for the flow solution is roughly
equivalent in cost to a residual evaluation for the adjoint equations,
with a slight penalty for the computation of the second-order invisi-
cid terms required by the latter. However, in the current implementa-
tion these second-order terms merely require an application of the
chain rule during computation of the first-order inviscid lineariza-
tions. Note that the flux Jacobians used for the left-hand side are
fixed during an adjoint solution, so that the initial cost of forming
these terms and performing the block-decompositions and block-
tridiagonal decompositions a priori is amortized over the duration
of the computation. In addition to the flux Jacobians required at
each time step of a flow solution, a residual evaluation is necessary,
which requires updated solution gradients and other solution-de-
dependent terms such as edge weights. Finally, the flow solution also
incurs a small amount of overhead at each time step in the computa-
tion of current forces and moments.

Alternatively, by storing the nearest-neighbor adjoint residual
contributions that arise from the linearization of the viscous and
turbulent terms, the wallclock time for an adjoint solution can be re-
duced to approximately half that of the flow solution while dou-
bled the memory requirement. Finally, by storing the complete lin-
erization of the second-order residual, the computational time is
reduced to roughly one-third of the baseline flow solution; how-
ever, the memory requirement for this approach is a factor of 6
higher than the baseline flow solution algorithm. Based on these re-
results, the viscous and turbulent terms are stored throughout the re-
mainder of the current work, and the inviscid contributions are re-
computed as needed.

Large-Scale Test Cases

Two large-scale test cases are used to evaluate the solution algo-
rithm on realistic configurations. Each of the grids has been gen-
erated by using the method described in Ref. 51. In the interest of
comparing asymptotic convergence behavior, solutions have been
converged considerably beyond the usual requirements for engi-
neering level answers; the stated run times do not represent time re-
quired to obtain a sufficiently accurate solution.

<table>
<thead>
<tr>
<th>Solver</th>
<th>Memory (GB)</th>
<th>Wallclock Time (Hours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flow Solver</td>
<td>1.5</td>
<td>50.4</td>
</tr>
<tr>
<td>Adjoint Solver:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Explicit computation of all</td>
<td>1.6</td>
<td>42.0</td>
</tr>
<tr>
<td>residual terms</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| Explicit computation of invisi-
   cid residual terms; viscous, turbu-
   lent residual terms stored    | 3.1         | 25.9                   |
| All residual terms stored      | 10.8        | 17.1                   |

Table 1. Memory and CPU requirements for flow solver and
various linearization storage strategies used for computing adjoint
residual.

High-Lift Configuration

Turbulent flow over the high-lift configuration shown in Fig.
11 is computed using eighteen 1.7 GHz Pentium IV processors. The
grid contains 846,863 nodes and 4,879,086 cells, and the implicit
line construction algorithm places 418,523 nodes in the line-im-
plex region. Although only a single plane is shown, symmetry
plane boundary conditions are used on both sides of the configura-
tion. The freestream Mach number is 0.2, the angle of attack is
12°, and the Reynolds number is 9 million based on the stowed
chord length. The CFL numbers for the flow equations and turbu-
lence model are linearly ramped from 10 to 200 and 1 to 20, respec-
tively, over the first 200 iterations. For this case, the objective func-
tion is based on the lift coefficient. The flow solver requires
approximately 3.5 GB of memory and 36.2 hours of wallclock time
for the current example; the adjoint solver requires roughly 7.1 GB
and 19.2 hours. The convergence histories for the flow and adjoint
solutions are shown in Fig. 12; the asymptotic rates are similar.

Modern Transport Configuration

For this test, transonic flow over the modern transport configura-
tion shown in Fig. 13 is considered. The grid for this case contains
1,731,262 nodes and 10,197,838 cells. The implicit line construc-
tion algorithm places 1,220,567 nodes in the line relaxation region.
The freestream Mach number for this case is 0.84, the angle of att-
ack is 2.25°, and the Reynolds number is 3 million based on the
mean aerodynamic chord. The computations shown here are per-
formed on twenty-three 2.4 GHz Pentium IV processors, and the
cost function is based on the drag coefficient.
The convergence histories for the flow and adjoint solutions are shown in Fig. 14. The CFL numbers for the flow equations and turbulence model are linearly ramped from 10 to 200 and 1 to 20, respectively, over the first 200 iterations. The CFL number used for the turbulence model is increased to 200 after the first 1000 iterations. As shown in Fig. 14, the asymptotic rates for both computations are similar. For this example, the flow solution requires 7.4 GB of memory and 41.8 hours of wallclock time; the adjoint solution requires 15.0 GB of storage and 32 wallclock hours.

Summary and Future Work

An adjoint solution algorithm that preserves discrete duality for turbulent flows on unstructured grids has been developed and implemented. The scheme is based on a backward-Euler discretization of the Reynolds-averaged Navier-Stokes equations with a tightly coupled one-equation turbulence model, where the linear problem at each time step is solved with a line-implicit relaxation in the boundary layer and a point-implicit technique through the remainder of the domain.

Results for several cases show that the exact dual scheme achieves asymptotic convergence behavior equivalent to that of the flow problem. By storing the viscous and turbulent contributions to the adjoint residual, the memory required for the algorithm is approximately twice that of the flow solution, but with execution times roughly 25-50% less for an equivalent number of iterations.

Efforts are currently underway to develop a multigrid implementation for improved convergence rates and to extend the discretization to include mixed-element grids. Real gas physical models from existing hypersonic structured-grid solvers are also being added, with the long-term goal of achieving a self-adapting analysis and design capability valid across the speed range.54

Acknowledgements

The authors wish to thank Drs. James Thomas, Kyle Anderson, Steve Allmaras, and Peter Gnoffo for many helpful discussions pertaining to the current work. Dr. Jamshid Samareh and Elizabeth Lee-Rausch are gratefully acknowledged for providing the geometric parameterization and high-lift grid, respectively. The study would not have been possible without the computational resources provided by Sally Viken and Bil Kleb, and Drs. Mark Carpenter, Veer Vatsa, and Dick Wilmoth.

Appendix: Derivation of Exact Dual Outer Iteration

In this appendix, the expression given by Eq. 15 is manipulated to give the exact dual iterative scheme as shown in Eq. 17. The derivation shown here is identical to that of Ref. 18.

Using the discrete form of integration by parts, namely

\[ \sum_{n=0}^{N-1} a_n^{n+1} (b_n^{n+1} - b_n^n) = a_N b_N - a_0 b_0 - \sum_{n=0}^{N-1} (a_n^{n+1} - a_n^n) b_n. \]  

Eq. 15 can be expanded as

\[ \frac{\partial f}{\partial Q} \left( \frac{\partial Q^*}{\partial D} \right)^N = \frac{\partial f}{\partial Q} \left( \frac{\partial Q^*}{\partial D} \right)^N - (\lambda_N)^T \left( \frac{\partial Q^*}{\partial D} \right)^N + (\lambda_0)^T \left( \frac{\partial Q^*}{\partial D} \right)^0 \]

\[ + \sum_{n=0}^{N-1} \left\{ (\lambda_n^{n+1} - \lambda_n^n)^T \left( \frac{\partial Q^*}{\partial D} \right)^n \right\} \]

\[ + (\lambda_n^{n+1})^T P \left[ \frac{\partial R}{\partial Q} \left( \frac{\partial Q^*}{\partial D} \right)^n + \frac{\partial R}{\partial D} \right] \]

Rearranging terms and applying the initial condition \[ \left( \frac{\partial Q^*}{\partial D} \right)_0 = 0 \] gives
\[
\frac{\partial f}{\partial Q} \frac{dQ}{dD} = \left[ \frac{\partial f}{\partial Q} \right] - \lambda^* + \sum_{\alpha=0}^{n-1} \left( \lambda^{*+1} - \lambda^* \right) \left( \frac{\partial R}{\partial Q} \right) \left( P \lambda^{*+1} - P \lambda^* \lambda^* \right) \left( \frac{\partial Q}{\partial \lambda^*} \right)
\]

By applying the condition \( \lambda^* = \left( \frac{\partial f}{\partial Q} \right)^{-1} \) and the variable substitution shown in Eq. 16, the exact dual iteration takes the form of Eq. 17 through the elimination of the terms involving \( dQ/dD \).

References

