On the iterative solution of adjoint equations

Michael B. Giles
Oxford University Computing Laboratory

ABSTRACT This paper considers the iterative solution of the adjoint equations which arise in the context of design optimisation. It is shown that naive adjoining of the iterative solution of the original linearised equations results in an adjoint code which cannot be interpreted as an iterative solution of the adjoint equations. However, this can be achieved through appropriate algebraic manipulations. This is important in design optimisation because one can reduce the computational cost by starting the adjoint iteration from the adjoint solution obtained in the previous design step.

1 Introduction

In computational fluid dynamics (CFD), one is interested in solving a set of nonlinear discrete flow equations of the form

$$N(U,X) = 0. \tag{1.1}$$

Here X is a vector representing the coordinates of a set of computational grid points, U is the vector of flow variables at those grid points and $N(U,X)$ is a differentiable vector function of the same dimension as U.

To solve these equations, many CFD algorithms use iterative methods which can be written as

$$U^{n+1} = U^n - R(U^n,X)\, N(U^n,X), \tag{1.2}$$

where R is a non-singular square matrix which is a differentiable function of its arguments. If R were defined to be L^{-1} where $L = \partial N/\partial U$ is the non-singular Jacobian matrix, this would be the Newton-Raphson method which converges quadratically in the final stage of convergence. However, in the iterative methods used in CFD, R is a poor approximation to L^{-1} and therefore they exhibit linear convergence asymptotically, with the final rate of convergence being given by the magnitude of the largest eigenvalue of the matrix $I - R(U,X)\, L(U,X)$, where I is the identity matrix.

In design optimisation [1, 5, 9], the grid coordinates X depend on a set of design parameters α, and one wishes to minimise a scalar objective function $J(U, X)$ by varying α. To achieve this using gradient-based optimisation methods [10], at each step in the optimisation process one needs to determine the linear sensitivity of the objective function to changes in α.

For a single design parameter, linearising the objective function gives

$$\frac{\mathrm{d}J}{\mathrm{d}\alpha} = \frac{\partial J}{\partial U} u + \frac{\partial J}{\partial \alpha}. \tag{1.3}$$

Here $u \equiv \mathrm{d}U/\mathrm{d}\alpha$ is the linear sensitivity of the flow variables to changes in the design parameter, which is obtained by linearising the flow equations to give

$$Lu = f, \tag{1.4}$$

where

$$f = -\frac{\partial N}{\partial X} \frac{\mathrm{d}X}{\mathrm{d}\alpha}, \tag{1.5}$$

and L and f are both functions of the nonlinear solution (U, X) for the current value of the design parameter α.

Using forward mode automatic differentiation tools such as ADIFOR or Odyssée (treating $R(U, X)$ as a constant for maximum efficiency since its linearisation is unnecessary because it is multiplied by $N(U, X)$ which is zero [13]) one can automatically generate a code for the iterative solution of these linear flow equations. This will use the same iterative procedure as the nonlinear equations and will correspond to the iteration

$$u^{n+1} = u^n - R\,(Lu^n - f), \tag{1.6}$$

starting from zero initial conditions. R is again a function of the nonlinear solution (U, X), and the linear convergence rate for this will be exactly the same as the asymptotic convergence rate of the nonlinear iteration as it is controlled by the same iteration matrix $I - RL$. However, if there are many design parameters, each one gives rise to a different vector f and linear flow perturbation u. Thus, the computational cost increases linearly with the number of design variables.

To avoid this increasing cost, one can use adjoint methods. The evaluation of the second term on the r.h.s. of Equation (1.3),

$$\frac{\partial J}{\partial \alpha} = \frac{\partial J}{\partial X} \frac{\mathrm{d}X}{\mathrm{d}\alpha},$$

is straightforward and inexpensive, so we focus attention on the first term, which we choose to write as an inner product $(\overline{u}, u) \equiv \overline{u}^T u$, by defining

$$\overline{u} \equiv \left(\frac{\partial J}{\partial U} \right)^T .$$

Since $Lu - f = 0$, simple algebraic manipulation yields

$$(\overline{u}, u) \;=\; (\overline{u}, u) - (\overline{f},\, Lu - f) \;=\; (\overline{f}, f) - (L^T \overline{f} - \overline{u},\, u) \;=\; (\overline{f}, f)$$

when $\overline{f}$ satisfies the adjoint equation

$$L^T \overline{f} = \overline{u}. \tag{1.7}$$

The advantage of the adjoint approach is that the calculation of F and the evaluation of the inner product $(\overline{f}, f)$ for each design variable is negligible compared to the cost of determining the single adjoint solution $\overline{f}$, and so the total cost is approximately independent of the number of design variables.

The issue to be addressed in this paper is how to obtain the adjoint solution $\overline{f}$ as the limit of a fixed point iteration which is the natural counterpart to that used for the nonlinear and linear equations, and which therefore has exactly the same rate of iterative convergence. It will be shown that the naive application of adjoint methodology to the iterative solution of the linear equations results in an algorithm in which the working variables do not correspond to the adjoint variables $\overline{f}$. However, with a slight reformulation it can be cast into the desired form.

The benefit of the adjoint calculation being formulated as a fixed point iteration is that one can obtain very significant computational savings if one can provide a good initialisation for the adjoint variables. This is possible in nonlinear design optimisation, since the adjoint variables computed for one step in the design optimisation can be used to initialise the computation of the adjoint variables for the next step. Indeed, it is usually found that the computational cost of the entire optimisation process is minimised by not fully converging the nonlinear and adjoint flow calculations at each design step, and instead letting the nonlinear and adjoint flow variables as well as the design parameters all evolve towards the optimum solution [11].

This issue of the iterative solution of adjoint equations has been investigated previously by Christianson [3, 4], but the context for his work is more abstract; the references should be consulted for further information.

2 Continuous equations

The iterative solution methods used in CFD are often based on an unsteady evolution towards the solution of a steady system of equations. Therefore, we begin by considering the unsteady solution $u(t)$ of the coupled system of differential equations

$$\frac{\mathrm{d}u}{\mathrm{d}t} = -P\,(L\,u - f), \tag{1.8}$$

for some constant preconditioning matrix P, subject to the initial conditions $u(0) = 0$. The functional of interest is the inner product $(\overline{u}, u(T))$, with the final time T chosen to be sufficiently large that $\mathrm{d}u/\mathrm{d}t$ is very small and therefore $u(T)$ is very close to being the solution of the steady equations.

Introducing the unsteady adjoint variable $\overline{u}_u(t)$, and using integration by parts, the unsteady adjoint formulation is given by

$$\begin{aligned}
(\overline{u}, u(T)) &= (\overline{u}, u(T)) - \int_0^T \left(\overline{u}_u,\, \frac{\mathrm{d}u}{\mathrm{d}t} + P\,(L\,u - f) \right) \mathrm{d}t \\
&= (\overline{u} - \overline{u}_u(T), u(T)) \\
&\quad - \int_0^T \left(-\frac{\mathrm{d}\overline{u}_u}{\mathrm{d}t} + L^T P^T \overline{u}_u,\, u \right) - (P^T \overline{u}_u,\, f)\ \mathrm{d}t \\
&= \int_0^T (P^T \overline{u}_u, f)\ \mathrm{d}t, \qquad (1.9)
\end{aligned}$$

where $\overline{u}_u(t)$ satisfies the differential equation

$$\frac{\mathrm{d}\overline{u}_u}{\mathrm{d}t} = L^T P^T \overline{u}_u, \qquad (1.10)$$

which is solved backwards in time subject to the final condition $\overline{u}_u(T) = \overline{u}$.

With the equations in this form, one would obtain the correct value for the functional, exactly the same value as one would obtain from the unsteady linear equations over the same time interval, but it is not immediately clear how the working variables $\overline{u}_u(t)$ are related to the steady adjoint solution $\overline{f}$.

To obtain the link with the steady adjoint equation, we define

$$\overline{f}(t) = \int_t^T P^T \overline{u}_u\ \mathrm{d}t, \qquad (1.11)$$

so that the functional is $(\overline{f}(0), f)$ and $\overline{f}(t)$ satisfies the differential equation

$$\begin{aligned}
-\frac{\mathrm{d}\overline{f}}{\mathrm{d}t} &= P^T \overline{u}_u \\
&= P^T \left(\overline{u} - \int_t^T \frac{\mathrm{d}\overline{u}_u}{\mathrm{d}t}\ \mathrm{d}t \right) \\
&= P^T \left(\overline{u} - \int_t^T L^T P^T \overline{u}_u\ \mathrm{d}t \right) \\
&= -P^T \left(L^T \overline{f} - \overline{u} \right), \qquad (1.12)
\end{aligned}$$

subject to the final condition $\overline{f}(T) = 0$.

In this form, the connection with the iterative solution of the steady adjoint equations becomes apparent. $\overline{f}(t)$ evolves towards the steady adjoint solution, and if T is very large then $\overline{f}(0)$ will be almost equal to the steady adjoint solution.

3 Discrete equations

Having considered the continuous equations to gain insight into the issue, we now consider the discrete equations and their iterative solution. As described in the Introduction, many standard iterative algorithms for solving the linearised equations can be expressed as

$$u^{n+1} = u^n - R\,(L\,u^n - f). \tag{1.13}$$

After performing N iterations starting from the initial condition $u^0 = 0$, the functional $(\overline{u}, u^N)$ is evaluated using the final value u^N.

Proceeding as before to find the discrete adjoint algorithm yields

$$\begin{aligned}
(\overline{u}, u^N) &= (\overline{u}, u^N) - \sum_{n=0}^{N-1} \left(\overline{u}_u^{n+1}, u^{n+1} - u^n + R\,(L\,u^n - f)\right) \\
&= (\overline{u} - \overline{u}_u^N, u^N) \\
&\quad - \sum_{n=0}^{N-1} \left\{ -(\overline{u}_u^{n+1} - \overline{u}_u^n, u^n) + (L^T R^T \overline{u}_u^{n+1}, u^n) - (R^T \overline{u}_u^{n+1}, f) \right\} \\
&= (\overline{u} - \overline{u}_u^N, u^N) \\
&\quad + \sum_{n=0}^{N-1} \left\{ (\overline{u}_u^{n+1} - \overline{u}_u^n - L^T R^T \overline{u}_u^{n+1}, u^n) + (R^T \overline{u}_u^{n+1}, f) \right\},
\end{aligned}$$

in which we have used the following identity which is the discrete equivalent of integration by parts,

$$\sum_{n=0}^{N-1} a^{n+1}\,(b^{n+1} - b^n) = a^N b^N - a^0 b^0 - \sum_{n=0}^{N-1} (a^{n+1} - a^n)\; b^n.$$

Consequently, if $\overline{u}_u$ satisfies the difference equation

$$\overline{u}_u^n = \overline{u}_u^{n+1} - L^T R^T \overline{u}_u^{n+1}, \tag{1.14}$$

subject to the final condition $\overline{u}_u^N = \overline{u}$, then the functional is equal to $(\overline{f}^{\,0}, f)$ where $\overline{f}^{\,0}$ is defined to be the accumulated sum

$$\overline{f}^{\,0} = \sum_{m=0}^{N-1} R^T \overline{u}_u^{m+1}. \tag{1.15}$$

The above description of the discrete adjoint algorithm corresponds to what would be generated by reverse mode automatic differentiation tools such as Odyssée [6], ADJIFOR [2] or TAMC [7]. As it stands, it is not clear what the connection is between the adjoint solution $\overline{f}$ and either the sum $\overline{f}^{\,0}$ or the working variable $\overline{u}_u^n$.

As with the continuous equations, it is preferable to cast the problem as a fixed point iteration towards the solution of the discrete adjoint equations. To do this we define $\overline{f}^n$ for $0 \leq n < N$ to be

$$\overline{f}^n = \sum_{m=n}^{N-1} R^T \overline{u}_u^{m+1}, \tag{1.16}$$

with $\overline{f}^N = 0$. The difference equation for $\overline{f}^n$ is

$$\begin{aligned} \overline{f}^n - \overline{f}^{n+1} &= R^T \overline{u}_u^{n+1} \\ &= R^T \left(\overline{u} - \sum_{m=n+1}^{N-1} (\overline{u}_u^{m+1} - \overline{u}_u^m) \right) \\ &= R^T \left(\overline{u} - \sum_{m=n+1}^{N-1} L^T R^T \overline{u}_u^{m+1} \right) \\ &= -R^T \left(L^T \overline{f}^{n+1} - \overline{u} \right), \end{aligned} \tag{1.17}$$

showing that the new working variable $\overline{f}^n$ evolves towards the solution of the adjoint equations. The rate of convergence is exactly the same as for the linear iteration since it is governed by the matrix $I - R^T L^T$ whose eigenvalues are the same as its transpose $I - LR$ and hence also $I - RL$, since if v is an eigenvector of the former then $L^{-1}v$ is an eigenvector of the latter with the same eigenvalue.

4 Applications

In applying the theory presented above to formulate adjoint algorithms, the key is to first express the nonlinear and linear iterative method in the correct form to determine the matrix R, and thereby determine the matrix R^T for the adjoint iterative scheme. Note that not all algorithms can be expressed in the above way with a constant matrix R. In the conjugate gradient algorithm, for example, the matrix R changes from one iteration to the next.

Reference [8] applies the theory to two kinds of iterative solver. The first is a quite general class of Runge-Kutta methods which is used extensively in CFD, and includes both preconditioning and partial updates for viscous and smoothing fluxes. Putting the linear iterative solver into the correct form requires a number of manipulations, and having then determined R^T further manipulations are necessary to express the adjoint algorithm in a convenient form for programming implementation. The correctness of the analysis has been tested with a simple MATLAB program which can solve either a simple scalar o.d.e. or an upwind approximation to the convection equation with a harmonic source term. In the latter case, the theory presented in this paper has been extended to include problems with complex variables, by replacing all vector and matrix transposes by their complex conjugates. In either case, it is verified that an identical number of iterations of either the linear problem or its adjoint yields identical values for the functional of interest.

Reference [8] also briefly considers the application of the theory to preconditioned multigrid methods [12], in which a sequence of coarser grids is used to accelerate the iterative convergence on the finest grid. Provided the smoothing algorithm used on each grid level within the multigrid solver is of the form given in Equation (1.6), it shown that all is well if the restriction operator for the adjoint solver is the transpose of the prolongation operator for the linear solver, and *vice versa*. This feature has been tested in unpublished research in developing a three-dimensional adjoint Navier-Stokes code using unstructured grids. Again, identical values have been obtained for the functional of interest after equal number of multigrid cycles with either the linear solver or its adjoint counterpart.

5 Conclusions

In this paper we have shown that the naive application of adjoint methods to the iterative solution of a linear system of equations produces an algorithm which does not correspond to the iterative solution of the corresponding adjoint system of equations. However, with some algebraic manipulations it can be transformed into an algorithm in which the working variables do converge to the solution of the adjoint equations.

Mathematically, the two approaches produce identical results if the second calculation starts from zero initial conditions. The advantage of the second formulation is that the computational costs can be greatly reduced if one has a good initial estimate for the solution. This happens in nonlinear design optimisation in which the adjoint solution for one step in the optimisation can be used as the initial conditions for the adjoint calculation in the following step.

This has implications for the use of automatic differentiation software in generating adjoint programs. The AD tools can still be used to generate the subroutines which construct the adjoint system of linear equations but to achieve the maximum computational efficiency it appears it is necessary to manually program the higher-level fixed point iterative solver.

Acknowledgments

The author is very grateful to Prof. Andreas Griewank for his considerable help in the writing of this paper. This research was supported by EPSRC under grant GR/L95700, and by Rolls-Royce plc and BAe Systems plc.

6 References

[1] W.K. Anderson and D.L. Bonhaus. Airfoil design on unstructured grids for turbulent flows. *AIAA J.*, 37(2):185–191, 1999.

[2] A. Carle, M. Fagan, and L.L. Green. Preliminary results from the application of automated code generation to CFL3D. AIAA Paper 98-4807, 1998.

[3] B. Christianson. Reverse accumulation and attractive fixed points. *Opt. Meth. and Software*, 3(4):311–326, 1994.

[4] B. Christianson. Reverse accumulation and implicit functions. *Opt. Meth. and Software*, 9(4):307–322, 1998.

[5] J. Elliott and J. Peraire. Practical 3D aerodynamic design and optimization using unstructured meshes. *AIAA J.*, 35(9):1479–1485, 1997.

[6] C. Faure. Splitting of algebraic expressions for automatic differentiation. Proceedings of the second SIAM Int. Workshop on Computational Differentiation, 1996.

[7] R. Giering and T. Kaminski. Recipes for adjoint code construction. *ACM Trans. Math. Software*, 24(4):437–474, 1998.

[8] M.B. Giles. On the use of Runge-Kutta time-marching and multigrid for the solution of steady adjoint equations. Technical Report NA00/10, Oxford University Computing Laboratory, 2000.

[9] M.B. Giles and N.A. Pierce. An introduction to the adjoint approach to design. *European Journal of Flow, Turbulence and Control*, to appear, 2000.

[10] P. Gill, W. Murray, and M. Wright. *Practical optimization.* Academic Press, 1981.

[11] A. Jameson and J. Vassberg. Studies of alternate numerical optimization methods applied to the brachistochrone problem. OptiCON '99 Conference, 1999.

[12] N.A. Pierce and M.B. Giles. Preconditioned multigrid methods for compressible flow calculations on stretched meshes. *J. Comput. Phys.*, 136:425–445, 1997.

[13] L.L. Sherman, A.C. Taylor III, L.L. Green, P.A. Newman, G.W. Hou, and V.M. Korivi. First and second order aerodynamic sensitivity derivatives via automautic differentiation with incremental iterative methods. *J. Comput. Phys.*, 129(2):307–331, 1996.