

# An Adjoint Solver for an Industrial CFD Code via Automatic Differentiation

Shaun A. Forth<sup>1</sup>, David W.F. Standingford<sup>2</sup> and Paul Dawson<sup>2</sup>

<sup>1</sup> Applied Mathematics & Operational Research, Engineering Systems Department, Cranfield University (Shrivenham Campus), Swindon SN6 8LA, UK.

<sup>2</sup> BAE SYSTEMS Advanced Technology Centre, Sowerby Building, PO Box 5, FPC 267 Filton, Bristol BS34 7QW, UK.

## 1 Introduction

Industrial use of CFD is now routine and the current challenge is to incorporate large scale CFD codes into the engineering design cycle via design optimisation. The associated use of gradients is non-trivial and is the subject of the monograph [6]. Aerodynamic designs may have 100's of design variables making adjoint methods, which compute gradients in time proportional to the (small) number of design objectives [4], attractive. Here we concentrate on discrete adjoint methods, in which the discretised flow solver is adjointed, since it may be generated and maintained by automatic differentiation (AD) [5] tools. Previously [7] we described using the Odyssee AD tool [3] to obtain and validate a discrete forward sensitivity version of the BAE SYSTEMS/AIRBUS UK unstructured mesh CFD code FLITE3D. In this paper we report on the generation, validation and performance optimisation of the corresponding adjoint solver.

## 2 The Flite3D CFD Package

Neglecting RK time-stepping and multigrid, FLITE3D's solver is described by (1-4).

$$\left. \begin{array}{l} \textbf{Surface Mesh Generator} \\ \text{Given design parameters } \underline{\beta} \\ \text{Surface mesh generation } \mathbf{x}_S = \mathbf{X}_S(\underline{\beta}) \end{array} \right\} \quad (1)$$

$$\left. \begin{array}{l} \textbf{Volume Mesh Generator} \\ \text{Volume Mesh generation } \mathbf{x} = \mathbf{X}(\mathbf{x}_S) \end{array} \right\} \quad (2)$$

$$\left. \begin{array}{l} \textbf{Geometric Pre-Processor} \\ \text{Calculate point and face surface normals } \mathbf{a}(\mathbf{x}_S, \underline{\beta}), \mathbf{n}(\mathbf{x}_S) \\ \text{Calculate Volume Mesh normals } \mathbf{w}(\mathbf{x}, \mathbf{x}_S) \end{array} \right\} \quad (3)$$

$$\left. \begin{array}{l} \textbf{Flow Solver} \\ \text{Initialise flow variables } \mathbf{u}^0 \\ \text{Do } n = 0, n_{max} - 1 \\ \quad \text{Finite volume residual } \mathbf{r}^n = \mathbf{R}(\mathbf{u}^n, \mathbf{x}, \mathbf{w}, \mathbf{n}) \\ \quad \text{If (converged) Exit Do Loop} \\ \quad \mathbf{u}^{n+1} = \mathbf{u}^n - \mathbf{P}_L(\mathbf{u}^n, \mathbf{x})\mathbf{r}^n \\ \quad \text{Aerodynamic Coefficients } [C_L, C_D, M_Y] = C(\mathbf{u}^n, \mathbf{x}, \mathbf{n}) \\ \text{EndDo} \end{array} \right\} \quad (4)$$

Given design parameters  $\underline{\beta}$ , for example B-spline coefficients, a surface mesh  $\mathbf{x}_S(\underline{\beta})$  is generated, followed by the volume mesh  $\mathbf{x}$  using Delauney techniques. A geometric pre-processor then calcu-

lates point  $\mathbf{a}(\mathbf{x}_S, \beta)$  and face surface normals  $\mathbf{n}(\mathbf{x}_S)$  followed by volume mesh normals  $\mathbf{w}(\mathbf{x}, \mathbf{x}_S)$ . For the flow-solver (4),  $\mathbf{P}_L(\mathbf{u}^n, \mathbf{x}) = -\Delta t(\mathbf{u}^n, \mathbf{x})/\mathbf{V}(\mathbf{x})$  is the diagonal matrix of negative time steps  $-\Delta t_i^n$  divided by control volumes  $V_i$  for each cell  $i$  in the mesh.

The principle goal of aerodynamic design optimisation, subject to structural and manufacturing constraints, is to reduce the drag  $C_D$  on an aircraft while keeping lift  $C_L$  fixed. Additionally the pitching moment  $M_Y$  is important to ensure aerodynamic stability. We therefore require an efficient means to calculate the derivatives  $\partial(C_L, C_D, M_Y)/\partial(\beta, \alpha)$  of objective drag  $C_D$  and constraints lift  $C_L$  and  $M_Y$  with respect to the design variables  $\beta$  and angle-of-attack  $\alpha$ . Since we may have many hundreds of design variables, an adjoint approach is desirable.

### 3 Adjoint Differentiation with Odyssee

We have initially adopted INRIA’s Odyssee tool [3] since it may be used commercially without restriction. In adjoint mode Odyssee utilises source-transformation via either a *syntax reversal* or *flow reversal* technique. Adjoint code generated via syntax reversal uses statically allocated local arrays to store data required for the reverse propagation of adjoints. Flow reversal utilises a taping mechanism that makes use of dynamic memory via a small C library. Since FLITE3D can use meshes of arbitrary size, global use of syntax reversal is precluded.

A decision was also taken, at least during the present proof-of-concept stage, to calculate derivatives with respect to surface  $\mathbf{x}_S$  and volume  $\mathbf{x}$  mesh coordinates and calculate surface mesh point normals  $\mathbf{a} = \mathbf{a}(\mathbf{x}_S)$  purely from the surface mesh ignoring curvature effects from the spline coefficients  $\beta$ . Then by moving the calculation of geometric quantities into the flow solver, effectively modifying the form of (4) to,

$$\left. \begin{array}{l}
 \mathbf{Modified\ Flow\ Solver} \\
 \text{Calculate point and face surface normals } \mathbf{a}(\mathbf{x}_S), \mathbf{n}(\mathbf{x}_S) \\
 \text{Calculate Volume Mesh normals } \mathbf{w}(\mathbf{x}, \mathbf{x}_S) \\
 \text{Initialise flow variables } \mathbf{u}^0 \\
 \text{Do } \quad n = 0, n_{max} - 1 \\
 \quad \text{Finite volume residual } \mathbf{r}^n = \mathbf{R}(\mathbf{u}^n, \mathbf{x}, \mathbf{w}, \mathbf{n}) \\
 \quad \text{If (converged) Exit Do Loop} \\
 \quad \mathbf{u}^{n+1} = \mathbf{u}^n - \mathbf{P}_L(\mathbf{u}^n, \mathbf{x})\mathbf{r}^n \\
 \quad \text{Aerodynamic Coefficients } [C_L, C_D] = C(\mathbf{u}^n, \mathbf{x}, \mathbf{n}) \\
 \text{EndDo}
 \end{array} \right\} \quad (5)$$

we obtained a program that calculates  $C_L$  and  $C_D$  in terms of the surface mesh  $\mathbf{x}_S$  and volume mesh  $\mathbf{x}$ .

#### 3.1 Code Preparation

Considerable time was spent preparing the code for differentiation, mostly giving explicit dimensions to assumed size (e.g. `real a(*)`) arrays so Odyssee’s taping mechanism could determine how many elements of such arrays needed to be stored to tape before being overwritten. The code within the `do` loop of (5) was differentiated independently of the code preceding it so a two-phase approach [5, p287] could be adopted for the adjoint solve: first converging the nonlinear solve; then converging the adjoint with taping required for only the one, converged flow state.

#### 3.2 Validation of Adjoint

The forward mode differentiated version of FLITE3D previously validated against finite-differencing [7] was used to validate the adjoint differentiation. All calculations were performed on a SUN BLADE with MHz CPU and MB Memory. Sensitivities of an ONERA M6 wing to angle-of-attack  $\alpha$  and Mach number  $M$  on a 753 point mesh at  $M = 0.5$  and  $\alpha = 5^\circ$  were calculated and are shown in Table 1. Data on the two rows labelled “AD(fwd)” was calculated using the forward

Method	forward sensitivities			$\frac{\text{CPU}(\partial c)}{\text{CPU}(c)}$
	$\partial C_L/\partial\alpha$	$\partial C_D/\partial\alpha$	$\partial M_Y/\partial\alpha$	
AD(fwd)	4.8460	$8.2895 \times 10^{-1}$	$-3.5575 \times 10^1$	3.1
AD(fwd) - rotation	4.8459	$8.2896 \times 10^{-1}$	$-3.5574 \times 10^1$	2.9
Method	$\partial C_L/\partial M$	$\partial C_D/\partial M$	$\partial M_Y/\partial M$	
AD(fwd)	$-2.3027 \times 10^1$	2.3371	$2.3050 \times 10^2$	7.7

Table 1: Forward Sensitivities of key integrated forces and moments to angle-of-attack  $\alpha$  and Mach number for the sharp trailing-edge ONERA M6 wing on a 753 point mesh

mode differentiated FLITE3D with derivatives of  $\alpha$  and  $M$  set to 1.0. Note that the boundary conditions for an angle-of-attack calculation are set by rotating the **far-field flow velocity** by the specified  $\alpha$ . Alternatively, for the row labelled “AD(fwd)-rotation” we set derivatives of **surface and volume meshes** to correspond to a rotation of the mesh by an angle  $\alpha$ . Comparison of the two sets of results enables validation of the mesh derivatives [7]. Table 1 also gives ratios  $\text{CPU}(\partial c)/\text{CPU}(c)$  of the run-times for the differentiated solver  $\text{CPU}(\partial c)$  to the nonlinear solver  $\text{CPU}(c)$ .

In a similar manner, Table 2 gives sensitivities and column “initial  $\text{CPU}(\partial c)/\text{CPU}(c)$ ” run-time ratios for the adjoint solver. Each row of the Table corresponds to one adjoint solution. Comparing the sensitivities of rows of Table 2 with columns of Table 1, we see excellent agreement between the sensitivities of the forward and adjoint sensitivity solvers. The efficiency of the adjoint solvers was disappointing being over 50 times the cost of the nonlinear solve.

Method	adjoint sensitivities		initial $\frac{\text{CPU}(\partial c)}{\text{CPU}(c)}$	optimized $\frac{\text{CPU}(\partial c)}{\text{CPU}(c)}$
	$\partial C_L/\partial\alpha$	$\partial C_L/\partial M$		
AD(rev)	4.8470	$-2.3045 \times 10^1$	55.5	10.8
AD(rev) - rotation	4.8459	$-2.3045 \times 10^1$	55.6	10.9
Method	$\partial C_D/\partial\alpha$	$\partial C_D/\partial M$		
AD(rev)	$8.2905 \times 10^{-1}$	2.3353	57.3	11.2
AD(rev) - rotation	$8.2895 \times 10^{-1}$	2.3353	57.4	11.2
Method	$\partial M_Y/\partial\alpha$	$\partial M_Y/\partial M$		
AD(rev)	$-3.5583 \times 10^1$	$2.3057 \times 10^2$	56.4	11.0
AD(rev) - rotation	$-3.5574 \times 10^1$	$2.3057 \times 10^2$	56.4	11.0

Table 2: Adjoint lift, drag and pitching moment sensitivities for the sharp trailing-edge ONERA M6 wing on a 753 point mesh

## 4 Improving Adjoint Performance

Profiling of the adjoint solver, via the UNIX `gprof` facility, showed that a tremendous amount of time was spent in Odyssee’s taping functions. Subroutines making most calls to the taping routines were those: that looped over the interior edges of the mesh calculating numerical fluxes and dissipations; and those looping around the boundary edges applying boundary conditions. For these we followed the approach of [1] for adjoining independent loops. To be precise, where the original solver contained residual assembly loops of the form to the left of Figure 1, we hand-coded adjoint loops of the form to the right. To generate `flux_ad`, the adjoint of the flux calculation subroutine `flux`, we used Odyssee in syntax-reversal form (since all arrays and loops are of fixed length) and thus taping is to local variables. Consequently we remove calls to Odyssee’s taping libraries completely from such loops. We note that this strategy is similar to the semi-automated

```

do edge=1,N
  il=vertex1(edge)
  i2=vertex2(edge)
  call flux(u(i1),u(i2),f)
  residual(i1)=residual(i1)+f
  residual(i2)=residual(i2)-f
enddo

do edge=N,1-1
  il=vertex1(edge)
  i2=vertex2(edge)
  f_ad=residual_ad(i1)
  &      -residual_ad(i2)
  call flux_ad(u(i1),u(i2),
  &      f,u_ad(i1),u_ad(i2),f_ad)
enddo

```

Figure 1: Nonlinear residual loop (left) and its hand-coded adjoint (right). An `_ad` denotes an adjoint variable or subroutine.

switchback strategy [2]. Additionally we suppressed the dependence of the local time step on active variables within the Runge-Kutta time-stepping algorithm, i.e.  $\mathbf{P}_L(\mathbf{u}^n, \mathbf{x}) = \mathbf{P}_L$  in (4) c.f.[5, p291].

The last column “optimized CPU( $\partial c$ )/CPU( $c$ )” of Table 2 shows the ratio of adjoint to function CPU times after performing these adjoint optimizations. A factor of 5 speed up is seen and adjoint calculations now take a respectable 10 times the cost of the nonlinear solve. The resulting sensitivities agreed with those of obtained for the un-optimized adjoint to at least 4 significant figures.

## 4.1 A More Realistic Calculation

Clearly the 753 point meshes of Sections 3.2 and 4 are unrealistically small for industrial use. In Table 3 we present results on a 14,148 point; the run-time is now comfortably under 10 times that of the nonlinear solve and memory requirements are only modestly increased from 226MB (nonlinear solve) to 257MB (adjoint solve).

Method	adjoint sensitivities		$\frac{\text{CPU}(\partial c)}{\text{CPU}(c)}$
	$\partial C_L / \partial \alpha$	$\partial C_L / \partial M$	
AD(rev)	$6.5545 \times 10^{-2}$	$8.6977 \times 10^{-1}$	8.8
AD(rev) - rotation	$6.5545 \times 10^{-2}$	$8.6977 \times 10^{-1}$	8.8
	$\partial C_D / \partial \alpha$	$\partial C_D / \partial M$	
AD(rev)	$1.1282 \times 10^{-2}$	$1.5079 \times 10^{-1}$	5.7
AD(rev) - rotation	$1.1285 \times 10^{-2}$	$1.5079 \times 10^{-1}$	5.7
	$\partial M_Y / \partial \alpha$	$\partial M_Y / \partial M$	
AD(rev)	$-3.3666 \times 10^{-2}$	$-5.9389 \times 10^{-1}$	8.1
AD(rev) - rotation	$-3.3658 \times 10^{-2}$	$-5.9389 \times 10^{-1}$	8.1

Table 3: Adjoint lift, drag and pitching moment sensitivities for the blunt trailing-edge ONERA M6 wing on a 14,148 point mesh

## 5 Conclusions and Outlook

We have generated an adjoint solver for a industrial unstructured mesh flow solver. The accuracy of the adjoint sensitivities has been validated against a previously validated forward sensitivity solver. We have demonstrated a 5-fold increase in efficiency of the adjoint solver by: hand-coding a few adjoint subroutines; using local variables instead of a tape-storage for the adjoint within key subroutines; and eliminating the dependency of the local time-step on the flow variables. Our full paper will present results from industrial-size meshes and some initial design optimisations.

**Acknowledgements:** We thank the UK Department of Trade and Industry, and both Murray Cross and Stefano Tursi of AIRBUS UK, for supporting this project under the CAST program. We thank INRIA for the use of the Odyssee tool.

## References

- [1] F. Courty, A. Dervieux, B. Koobus, and L. Hascoet. Reverse automatic differentiation for optimum design: from adjoint state assembly to gradient computation. *Optimization Methods and Software*, 18(5):615–627, 2003.
- [2] M. Fagan and A. Carle. Switchback: Profile-driven recomputation for reverse mode. In P. M. Soot, C. K. Tan, J. J. Dongarra, and A. G. Hoekstra, editors, *Computational Science - ICCS 2002*, volume II of *Lecture Notes in Computer Science; Vol 2330*, pages 1029–1038. Springer-Verlag, 2002.
- [3] C. Faure and Y. Papegay. Odyssee User’s Guide. Version 1.7. Rapport technique RT-0224, INRIA, Sophia-Antipolis, France, Sept. 1998. See [www.inria.fr/RRRT/RT-0224.html](http://www.inria.fr/RRRT/RT-0224.html), and [www.inria.fr/safir/SAM/Odyssee/odyssee.html](http://www.inria.fr/safir/SAM/Odyssee/odyssee.html).
- [4] M. B. Giles and N. A. Pierce. An introduction to the adjoint approach to design. *Flow, Turbulence and Combustion*, to appear.
- [5] A. Griewank. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Number 19 in *Frontiers in Appl. Math.* SIAM, Philadelphia, Penn., 2000.
- [6] B. Mohammadi and O. Pironneau. *Applied Shape Optimization for Fluids*. Numerical Mathematics and Scientific Computation. Oxford Science Publications, Clarendon Press, Oxford, 2001.
- [7] D. W. F. Standingford and S. A. Forth. A discrete sensitivity solver for an industrial CFD code via automatic differentiation. In S. Armfield, P. Morgan, and K. Srinivas, editors, *Computational Fluid Dynamics 2002: Proceedings of the Second International Conference in Computational Fluid Dynamics, ICCFD, Sydney, Australia, 15-19 July 2002*, pages 82–87. Springer-Verlag, 2003.