Efficient Hessian Calculation using Automatic Differentiation

Devendra P. Ghate* and Michael B. Giles†

Oxford University Computing Laboratory, Oxford, Oxfordshire, OX1 3QD, UK

This paper outlines the basic formulation for Hessian calculation for a functional of interest for aerodynamic applications. Complete analysis of the order of the computational cost with the number of independent variables is also presented. An efficient implementation strategy using automatic differentiation packages is outlined. Finally, results are presented for a two dimensional airfoil code and compared with finite difference Hessian calculation. Comparison of quadratic extrapolation of a functional of interest with the adjoint corrected linear extrapolation is also presented.

I. Introduction

An algorithm to calculate the Hessian of a functional of interest is outlined here. In the context of aerodynamics, the examples of functionals of interest are lift, drag or total pressure loss. Typically these quantities are calculated after an iterative solution of nonlinear partial differential equations. This makes it difficult to calculate the gradients and the Hessian. The most widely used method of finite difference is sensitive to step-size selection and is computationally expensive.

We have already demonstrated the effective use of automatic differentiation[1,2] to automate the process of Jacobian calculation in our previous publication.[3] Automatic differentiation helps in keeping the linearised version of the nonlinear codes in-sync with the continuous changes made in the nonlinear code. The Jacobian obtained by this method is theoretically accurate to machine precision.[2] The proposed method for Hessian calculation is a natural extension of this. We now show how automatic differentiation can also be used to compute the Hessian.

The Hessian thus obtained has various applications in optimisation algorithms, Monte Carlo simulations, surrogate modelling and uncertainty analysis.

II. Background

The idea of calculating the Hessian matrix using automatic differentiation is not new and the automatic differentiation community has been addressing the issue of calculating higher order derivatives for a number of years. In one of the earliest papers, Christianson[4] describes an algorithm for Hessian calculation using reverse accumulation. There are two commonly used methods for calculating Hessians using automatic differentiation: forward-on-forward and forward-on-reverse. Forward-on-forward is a straightforward double application of automatic differentiation to the original code in forward mode. If there are n independent variables and a single functional of interest, then the computational cost of this approach is $O(n^2)$. Similarly in forward-on-reverse mode, the code is differentiated first in the reverse mode and then in the forward mode. The computational cost for a functional of interest of dimension m with respect to n independent variables is $O(m \times n)$.

*DPhil Candidate

†Professor of Scientific Computing

Presently, most of the automatic differentiation tools (ADOL-C, ADIFOR, Tapenade) can be used to calculate the Hessian using forward-on-forward or forward-on-reverse modes. These packages also provide in-built driver routines that calculate the Hessian or Hessian-vector products. However the computational cost of direct application of automatic differentiation as a black-box is unacceptable for large iterative solution codes.

In the Aerospace community, the method described here was initially investigated by Taylor *et al*[5] along with various other algorithms, but the publication does not go into the implementation details for a generic fluid dynamics code. Here we aim to demonstrate the method in detail with the help of a two dimensional airfoil code. The order of computational cost, efficient automatic differentiation implementation and the mathematical background behind the idea are presented.

III. Basic Formulation

We are interested in the Hessian of a functional of interest $j(\alpha) = J(\alpha, w(\alpha)),\ j \in \mathbb{R}^m$ with respect to independent variables $\alpha \in \mathbb{R}^n$ such that $w(\alpha) \in \mathbb{R}^p$ satisfies the state equation

$$R(\alpha, w) = 0. \tag{1}$$

Henceforth w will be referred to as the intermediate variable. To take an example from the fluid dynamics code,

$$w = [x, u]$$

where,

x are the grid variables which change according to the design variables, and

u are referred to as the state variables which are obtained by solving the state equation, e.g. the discretised Navier-Stokes equations.

$R(\alpha, w)$ refers to such state equations augmented by the grid generation equations. Typically equation (1) is a set of nonlinear equations and is solved using some fixed point iteration method which is computationally expensive.

For simplicity consider that j is uni-dimensional, *i.e.* $m = 1$. The derivative of j with respect to one individual component of α is given by

$$\frac{\partial j}{\partial \alpha_i} = \frac{\partial J}{\partial \alpha_i} + \frac{\partial J}{\partial w}\ \frac{\partial w}{\partial \alpha_i}. \tag{2}$$

Differentiating equation (2) again gives us

$$\begin{aligned}\frac{\partial^2 j}{\partial \alpha_i \partial \alpha_j} = \frac{\partial^2 J}{\partial \alpha_i\ \partial \alpha_j} + \frac{\partial^2 J}{\partial \alpha_i \partial w}\left(\frac{\partial w}{\partial \alpha_j}\right) + \frac{\partial^2 J}{\partial \alpha_j \partial w}\left(\frac{\partial w}{\partial \alpha_i}\right) \\ + \frac{\partial^2 J}{\partial w^2}\left(\frac{\partial w}{\partial \alpha_i}\ \frac{\partial w}{\partial \alpha_j}\right) + \frac{\partial J}{\partial w}\left(\frac{\partial^2 w}{\partial \alpha_i\ \partial \alpha_j}\right)\end{aligned} \tag{3}$$

The above equation tells us that the calculation of the Hessian requires the linear sensitivities of w. Also the last term on the right hand side of the equation requires the second order sensitivity of w. The computational cost of this calculation is

- one baseline nonlinear solution w,
- $O(n)$ linear solutions of $\frac{\partial w}{\partial \alpha_i}$,
- $O(n^2)$ second derivatives of $\frac{\partial^2 w}{\partial \alpha_i \alpha_j}$, and
- $O(n^2)$ evaluations of the right-hand side of equation (3).

If the intermediate variables w are an explicit function of the design variables, then this is a simple task using automatic differentiation. The original routines can be differentiated twice in the forward mode to propagate the second order sensitivities. The calculation of the linear and second order sensitivities of the intermediate variables will be computationally inexpensive.

However, we know that in case of fluid dynamics, the intermediate variables are an implicit function of the design variables and they require an iterative procedure for the solution. This makes the calculation of the linear and second order sensitivities of the intermediate variables w computationally expensive. A different formulation is presented henceforth to reduce this computational cost.

Equation (3) can be rearranged as

$$\frac{\partial^2 j}{\partial\alpha_i \partial\alpha_j} = \frac{\partial J}{\partial w}\ \frac{\partial^2 w}{\partial\alpha_i \partial\alpha_j} + D^2_{i,j}J, \tag{4}$$

where

$$D^2_{i,j}J = \frac{\partial^2 J}{\partial\alpha_i\ \partial\alpha_j} + \frac{\partial^2 J}{\partial\alpha_i \partial w}\left(\frac{\partial w}{\partial\alpha_j}\right) + \frac{\partial^2 J}{\partial\alpha_j \partial w}\left(\frac{\partial w}{\partial\alpha_i}\right) + \frac{\partial^2 J}{\partial w^2}\left(\frac{\partial w}{\partial\alpha_i}\ \frac{\partial w}{\partial\alpha_j}\right). \tag{5}$$

Differentiating the state equation (1) gives

$$\frac{\partial R}{\partial\alpha_i} + \frac{\partial R}{\partial w}\ \frac{\partial w}{\partial\alpha_i} = 0. \tag{6}$$

Differentiating again we get,

$$\frac{\partial R}{\partial w}\ \frac{\partial^2 w}{\partial\alpha_i \partial\alpha_j} + D^2_{i,j}R = 0, \tag{7}$$

where $D^2_{i,j}R$ is similarly defined as $D^2_{i,j}J$ in equation (5).

Now substituting for $\frac{\partial^2 w}{\partial\alpha_i \partial\alpha_j}$ in equation (4) from equation (7) we get

$$\begin{aligned} \frac{\partial^2 j}{\partial\alpha_i \partial\alpha_j} &= -\frac{\partial J}{\partial w}\left(\frac{\partial R}{\partial w}\right)^{-1} D^2_{i,j}R + D^2_{i,j}J \\ &= v^T D^2_{i,j}R + D^2_{i,j}J. \end{aligned} \tag{8}$$

Here v is the adjoint solution associated with the functional of interest J defined by the adjoint equation

$$\left(\frac{\partial R}{\partial w}\right)^T v + \left(\frac{\partial J}{\partial w}\right)^T = 0. \tag{9}$$

These are the same adjoints that are widely used in aerodynamic gradient calculation and optimisation. Various methods for calculation of the adjoint solutions are available.[6] Equation (8) is used to calculate the complete Hessian. The entire formulation presented here is also valid for a multi-dimensional functional of interest.

Now let us look at the computational cost for calculating the entire Hessian. First we need the solution of the state equation (1) to calculate the baseline value of the intermediate variables w^0 corresponding to the values of the design variables α^0 at which we need the Hessian. Then looking at equation (8) it is clear that we need a single adjoint solution $v(w^0)$ corresponding to J. Also, equation (5) tells us that we need calculation of the n linear solutions $\frac{\partial w}{\partial\alpha_i}(w^0)$ corresponding to the n independent variables α_i. If these solutions are available then we only need to evaluate $D^2_{i,j}J$ and $D^2_{i,j}R$ for each entry of the Hessian, *i.e.* we evaluate these functions for each pair of α_i and α_j.

Hence the total computational cost of the entire Hessian calculation is:

- Single baseline nonlinear solution w^0,
- $O(n)$ linear flow solutions $\frac{\partial w}{\partial \alpha_i}(w^0)$,
- single adjoint solution $v(w^0)$,
- $O(n^2)$ evaluations of $D^2_{i,j}J,\ D^2_{i,j}R$, and
- $O(n^2)$ dot products for $v^T D^2_{i,j}R$.

As a single evauluation is much cheaper than an iterative solution, the cost of the last two items is negligible and hence the computational cost of calculating the entire Hessian is approximately $O(n)$.

The entire argument presented above for a single dimensional functional of interest J also holds true for the general case of m dimensions. The net computational cost would be of the order $O(n+m)$ because of the $O(m)$ adjoint solutions corresponding to all the functionals of interest.

The basic concept behind the Hessian calculation for a general case has been explained in this section. But the mathematical formulation used in our implementation is slightly different and is presented in the next section.

IV. Formulation for Fluid Mechanics

We are interested in the Hessian of a functional of interest $j(\alpha) = J(\alpha, x(\alpha), u(\alpha)),\ j \in \mathbb{R}^m$ with respect to the independent variables $\alpha \in \mathbb{R}^n$ such that $x(\alpha)$ and $u(\alpha)$ satisfy the state equation

$$R(\alpha, x(\alpha), u(\alpha)) = 0. \tag{10}$$

α are the design variables and we are interested in the Hessian of the functional of interest with respect to these design variables.

x are the grid variables which change according to the design variables. u are referred to as the flow variables.

For simplicity consider that j is uni-dimensional, i.e. $m = 1$. The derivative of j with respect to one individual component of α is given by

$$\frac{\partial j}{\partial \alpha_i} = \frac{\partial J}{\partial \alpha_i} + \frac{\partial J}{\partial x}\ \frac{\partial x}{\partial \alpha_i} + \frac{\partial J}{\partial u}\ \frac{\partial u}{\partial \alpha_i}. \tag{11}$$

Differentiating equation (11) again gives us

$$\frac{\partial^2 j}{\partial \alpha_i \partial \alpha_j} = \frac{\partial J}{\partial u}\ \frac{\partial^2 u}{\partial \alpha_i \partial \alpha_j} + D^2_{i,j}J \tag{12}$$

where,

$$\begin{aligned} D^2_{i,j}J = \frac{\partial^2 J}{\partial \alpha_i\ \partial \alpha_j} &+ \frac{\partial J}{\partial x}\ \frac{\partial^2 x}{\partial \alpha_i \partial \alpha_j} \\ &+ \frac{\partial^2 J}{\partial \alpha_i \partial u}\left(\frac{\partial u}{\partial \alpha_j}\right) + \frac{\partial^2 J}{\partial \alpha_j \partial u}\left(\frac{\partial u}{\partial \alpha_i}\right) + \frac{\partial^2 J}{\partial u^2}\left(\frac{\partial u}{\partial \alpha_i}\ \frac{\partial u}{\partial \alpha_j}\right) \\ &+ \frac{\partial^2 J}{\partial \alpha_i \partial x}\left(\frac{\partial x}{\partial \alpha_j}\right) + \frac{\partial^2 J}{\partial \alpha_j \partial x}\left(\frac{\partial x}{\partial \alpha_i}\right) + \frac{\partial^2 J}{\partial x^2}\left(\frac{\partial x}{\partial \alpha_i}\ \frac{\partial x}{\partial \alpha_j}\right) \end{aligned} \tag{13}$$

Similarly, the entire process can be repeated for the state equation (10) to give

$$\frac{\partial R}{\partial u}\ \frac{\partial^2 u}{\partial \alpha_i \partial \alpha_j} + D^2_{i,j}R = 0, \tag{14}$$

where, $D^2_{i,j}R$ is similarly defined as $D^2_{i,j}J$ in equation (13).

Now substituting for $\frac{\partial^2 u}{\partial \alpha_i \partial \alpha_j}$ in equation (12) from equation (14) we get

$$\begin{aligned} \frac{\partial^2 j}{\partial \alpha_i \partial \alpha_j} &= -\frac{\partial J}{\partial u}\left(\frac{\partial R}{\partial u}\right)^{-1} D^2_{i,j}R + D^2_{i,j}J \\ &= v^T D^2_{i,j}R + D^2_{i,j}J. \end{aligned} \tag{15}$$

where v is the adjoint solution associated with the functional of interest J defined by the adjoint equation

$$\left(\frac{\partial R}{\partial u}\right)^T v + \left(\frac{\partial J}{\partial u}\right)^T = 0. \tag{16}$$

Equation (15) is used to calculate the complete Hessian. It should be noted here that only the second derivative of flow variables is replaced here by the flow adjoint solution. Typically, the grid generation process is computationally much cheaper than the iterative flow solutions. Also, because of the involvement of the CAD packages in the grid generation process, it is difficult to develop adjoint codes for the grid generation process. Hence in our implementation we do not use grid adjoint solutions. The linear and second derivatives of the grid variables $x(\alpha)$ are calculated in the forward mode and then passed on to the routines in the flow solver.

However if the grid adjoint solution capability exists or is desirable then the earlier more generic formulation can be used. The entire formulation presented here is also valid for a multi-dimensional functional of interest. The order of the computational cost remains the same as the earlier generic formulation assuming that the second order sensitivities of grid variables can be calculated cheaply. The next section discusses some of the related implementation issues.

V. Implementation

Hessian code development process is a natural extension of the linear and adjoint code development reported in our earlier work.[3] The entire nonlinear code has to be written in a modular fashion with all the nonlinear bits separated out from the time integration loop. Each of these functions containing nonlinear bits are then double differentiated in the forward mode using the automatic differentiation software. For example the original nonlinear wall flux calculation subroutine is

`flux_wall(x1,x2,q,res)`,

where, `x1` and `x2` are the nodes defining the wall edge, `q` is the state vector of the interior cell and `res` is the residue vector. This subroutine is differentiated in forward mode using an automatic differentiation software with `x1`,`x2` and `q` as the independent variables and `res` as the dependent variable. The differentiated subroutine is

`flux_wall_d(x1,x1d,x2,x2d,q,qd,res,resd)`,

where all the variables appended with d are the perturbation variables. This subroutine is again differentiated in the forward mode with `x1`, `x1d`, `x2`, `x2d`, `q` and `qd` as the independent variables while `res` and `resd` are the dependent variables.

```
flux_wall_d2(x1,x1d0,x1d,x1dd,x2,x2d0,x2d,x2dd,
             q,qd0,qd,qdd,res,resd0,resd,resdd)
```

Effectively each of the original variable gets linearised twice and we also have the second order perturbation in the variables appended by `dd`. These variables correspond to the complete second order derivative of the original variable, *i.e.* `resdd` is the complete second order derivative given by an expression similar to equations (12) and (13) if `x1d`, `x2d`, `qd` are initialised with perturbations with respect to α_i and `x1d0`, `x2d0`, `qd0` are initialised with perturbations with respect to α_j. Now if we set `qdd` $= 0$, then essentially we are calculating the $D^2_{i,j}$ operator applied to `res`. These perturbations have to be carried forward throughout the solver code in a similar fashion to calculate the complete $D^2_{i,j}R$ over the entire grid. $D^2_{i,j}J$ is evaluated in a similar fashion.

VI. Validation Checks

Although automatic differentiation by definition removes all the user intervention and associated errors, it is always a good practice to introduce validation checks to ensure the correctness of the implementation. Validation checks for Hessian calculation are developed on the similar lines as discussed in our previous publication for the linear and adjoint code development.[3] Unfortunately, it is not so straightforward as the linear and adjoint code development. Still some simple checks can be introduced.

Given fully converged nonlinear u and linear $\frac{\partial u}{\partial \alpha_i}$ solutions the following equations should be satisfied in the entire domain to machine precision

$$R(x,u) = 0, \quad \text{and} \quad \frac{\partial R}{\partial x}\frac{\partial x}{\partial \alpha_i} + \frac{\partial R}{\partial u}\frac{\partial u}{\partial \alpha_i} = 0.$$

These checks ensure that the converged nonlinear and linear solutions are being correctly introduced in the Hessian code. Also, it is desirable to ensure that the adjoint solution is consistent with the linear solutions using the checks discussed in our previous publication.[3]

Finally, it should be tested that the calculated Hessian is symmetric, *i.e.*

$$\frac{\partial^2 j}{\partial \alpha_i \partial \alpha_j} = \frac{\partial^2 j}{\partial \alpha_j \partial \alpha_i}.$$

This identity should be satisfied to machine precision. This calculation would require n^2 evaluations of the $D^2_{i,j}J$ and $D^2_{i,j}R$ functions. Once this has been validated then only the upper diagonal Hessian matrix can be calculated requiring $\frac{n(n+1)}{2}$ evaluations.

However the final validation has to be the comparison with finite difference results. These are presented in the next section.

VII. Results

A two dimensional airfoil code is used to demonstrate these ideas. The nonlinear solver using finite difference discretisation with two step predictor-corrector time integration is developed. Tapenade[7] developed by Laurent Hascoët and co-workers at Inria, France is used here for automatic differentiation. A consistent linear and adjoint codes were developed for the baseline nonlinear airfoil code using Tapenade. The entire source code along with a driver program can be downloaded from our website.[8]

Three modes of artificial perturbation are introduced to the baseline geometry. Figure 1 shows the three modes of perturbation namely: thickness, angle-of-attack and leading edge shape.

A `Makefile` is written to calculate all the relevant linear, nonlinear and adjoint versions of the code using Tapenade. A separate program `air_hes` is written which calls all the appropriate double differentiated subroutines to calculate the $D^2_{i,j}R$ and $D^2_{i,j}J$ functions, and finally the entire Hessian with respect to these three modes. Despite the fact that the Hessian is symmetric, all nine components are calculated. The Hessian thus calculated is compared with the central finite difference approximation given by

$$\frac{\partial^2 j}{\partial \alpha_i \partial \alpha_j}(\alpha^0) = \frac{1}{2\Delta\alpha_j}\left(\frac{\partial j}{\partial \alpha_i}(\alpha^0 + \Delta\alpha_j) - \frac{\partial j}{\partial \alpha_i}(\alpha^0 - \Delta\alpha_j)\right).$$

This expression is used instead of the second order finite difference of the nonlinear solution to minimise the sensitivity to the step-size. Also, the linear perturbations calculated using a linear solver are accurate to machine precision.

The far-field conditions in non-dimensional units are:

- Pressure = 1,
- Density = 1,

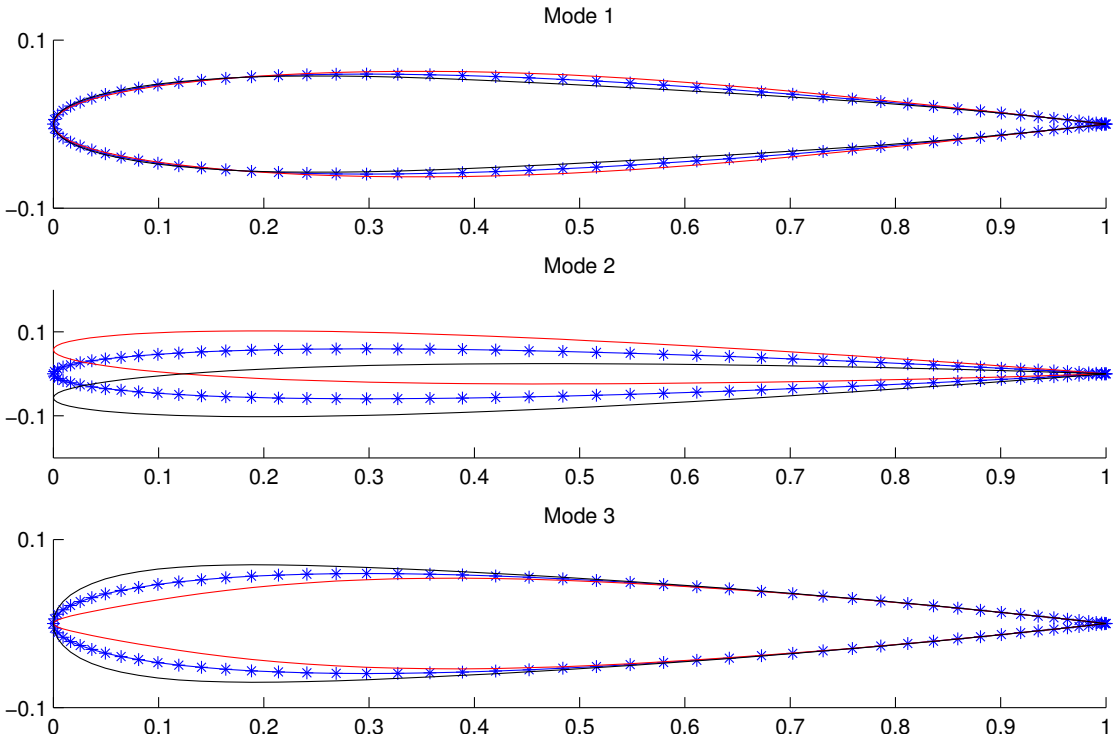

Figure 1. Artificial modes of perturbation introduced in NACA0012 airfoil

- Mach = 0.4, and
- Flow Angle = 3^o.

Table 1 shows the comparison between the Hessian calculated directly and the finite difference calculations using appropriate stepsize. Good agreement between these two is the final verification of the correctness of the Hessian code implementation.

Modes	Finite Difference	Direct
1 - 1	$-\mathbf{3.111203}625702499E-07$	$-\mathbf{3.111203}862791910E-07$
1 - 2	$-\mathbf{2.097600}811599999E-06$	$-\mathbf{2.097600}748629300E-06$
1 - 3	$-\mathbf{9.959223}201885120E-07$	$-\mathbf{9.959223}212828186E-07$
2 - 1	$-\mathbf{2.0976007}47610895E-06$	$-\mathbf{2.0976007}48629318E-06$
2 - 2	$-\mathbf{2.1596874}23428786E-04$	$-\mathbf{2.1596874}24802269E-04$
2 - 3	$-\mathbf{1.7465378}57481162E-04$	$-\mathbf{1.7465378}59860203E-04$
3 - 1	$-\mathbf{9.95922}904915369E-07$	$-\mathbf{9.95922}3212828262E-07$
3 - 2	$-\mathbf{1.7465378}61210817E-04$	$-\mathbf{1.7465378}59860204E-04$
3 - 3	$-\mathbf{1.9709370}36569875E-05$	$-\mathbf{1.9709370}34187627E-05$

Table 1. Comparison between direct Hessian calculation and finite difference calculation for Lift. (Bold digits are matching digits)

VIII. Extrapolation

One of the major applications of the Hessian thus obtained is in extrapolation. It is interesting to compare the performance of the quadratic extrapolation using linear and Hessian solutions with the linear extrapolation using adjoint correction.[9] For the sake of completeness, the two expressions are given here. The quadratic

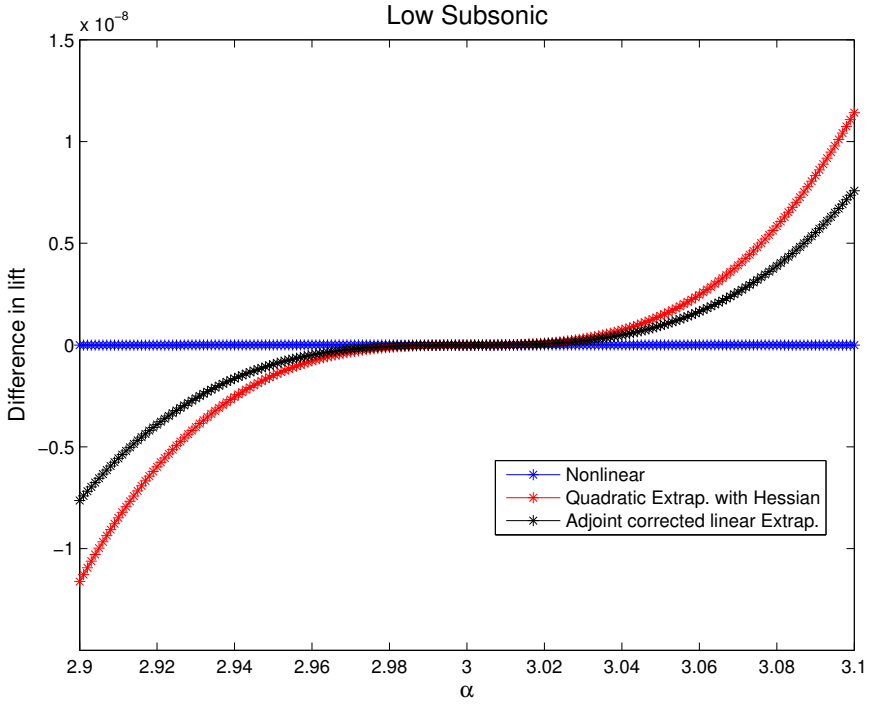

Figure 2. Comparison between the errors from the cubic fit for the quadratic and the adjoint corrected linear extrapolation for low subsonic case (Mach $= 0.4$ and AOA $= 3^o$)

extrapolation is given by

$$j_\alpha = j_{\alpha_0} + \frac{\partial j}{\partial \alpha}(\alpha - \alpha_0) + \frac{1}{2}\frac{\partial^2 j}{\partial \alpha^2}(\alpha - \alpha_0)^2,$$

while adjoint corrected linear extrapolation is

$$j_\alpha = j_{\alpha_0} + \frac{\partial j}{\partial \alpha}(\alpha - \alpha_0) - v(\alpha_0)^T R\left(x(\alpha), u(\alpha_0) + \frac{\partial u}{\partial \alpha}(\alpha - \alpha_0)\right).$$

The adjoint corrected linear extrapolation thus obtained has a third order leading error (i.e. $O\left((\Delta\alpha)^3\right)$). Extrapolation about a base angle-of-attack (AOA) is carried out using the second mode of perturbation. A set of nonlinear simulations converged to full machine precision is carried out by varying α from 2.9^o to 3.1^o in the steps of 0.001^o. The nonlinear lift thus obtained is compared with the two methods of extrapolation described above. Figure 2 plots the difference between these extrapolations and a cubic fit of the lift (calculated using nonlinear simulations) for the low subsonic range (Mach $= 0.4$ and AOA $= 3^o$) against the angle-of-attack. Both the approaches look equally accurate in this case. As the perturbation in α is very small, we see extremely small error with respect to the cubic fit of the nonlinear lift.

Similarly, Figure 3 shows comparison for a higher subsonic test case with Mach $= 0.65$ and AOA $= 10^o$. The behaviour of the nonlinear lift is not smooth with some pronounced kinks. Further investigation of this curious behaviour revealed that these kinks are arising because of the fundamental non differentiability of the underlying function.

A key quantity Adt which is area of a cell divided by the local time-step in the nonlinear calculations is calculated as

$$Adt = \frac{\sum_i |u\ dy_i - v\ dx_i| + c\ \sqrt{dx_i^2 + dy_i^2}}{CFL}, \tag{17}$$

Here, u and v are the velocity components in x and y directions respectively, while dx and dy are the projections of the length of an edge. Local speed of sound is denoted by c. CFL is the Courant-Friedrichs-Lewy number. The summation is over the four edges forming a cell.

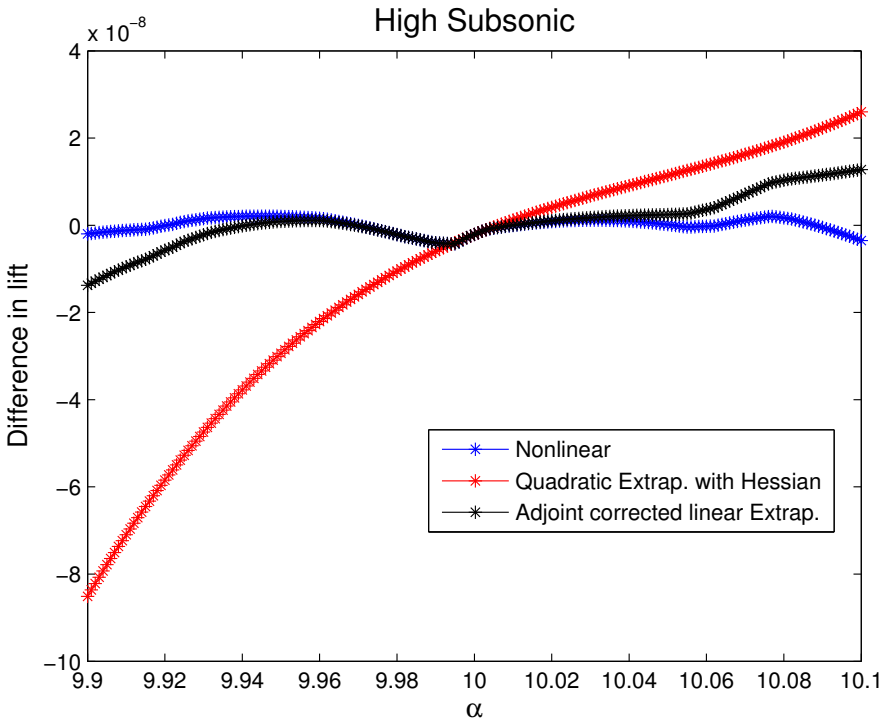

Figure 3. Comparison between quadratic and adjoint corrected linear extrapolation for high subsonic case (Mach $= 0.65$ **and AOA** $= 10^o$**)**

Because of the term with the absolute function, though this is C_0 continuous, it is not C_1 continuous. To elaborate the problem more clearly, consider a generic function

$$g(x) = |f(x)|$$

with $f(x)$ being infinitely differentiable. Now $g(x)$ is continuous on the entire real line but there is a discontinuity in $g'(x)$ at all x with $f(x) = 0$ and $f'(x) \neq 0$. Investigation of a relatively large kink in Figure 3 at $\alpha = 9.995^o$ revealed that $|u\ dy - v\ dx|$ changes its sign in multiple cells between $\alpha = 9.994$ and $\alpha = 9.995$. These small perturbations accumulated over multiple cells account for the kink in the nonlinear lift value. If the Hessian is calculated near such a sensitive point then it introduces error.

It should be noted here that these errors are extremely small and in comparison to the convergence and discretisation errors in the real-life applications, these are not significant. However, such errors accumulated from multiple sources for highly complex codes may create significant errors. The current example of a two dimensional inviscid solver is too simplistic to assess the true extent of errors that might be introduced.

Also it should be noted that the adjoint corrected linear extrapolation performs better than the quadratic extrapolation in this case. It closely follows the trend of the underlying nonlinear solution.

To confirm that this was the only source of error for the non-smooth behaviour, we modified the time-step calculation slightly to avoid the sign change. The time-step calculation as given in equation 17 was modified to

$$Adt = \frac{1}{CFL} \sum_i c\ ds_i \left(\sqrt{(m_x \tfrac{dy_i}{ds_i} - m_y \tfrac{dx_i}{ds_i})^2 + \epsilon^2}\ + 1 \right),$$

where $ds_i = \sqrt{dx_i^2 + dy_i^2}$, $m_x = \frac{u}{c}$, $m_y = \frac{v}{c}$ and $\epsilon = 0.1$. ϵ is chosen to ensure that the derivative of Adt is always defined. The entire simulations are carried out again and the results were presented in Figure 4. The smooth behaviour of the nonlinear lift confirms the hypothesis.

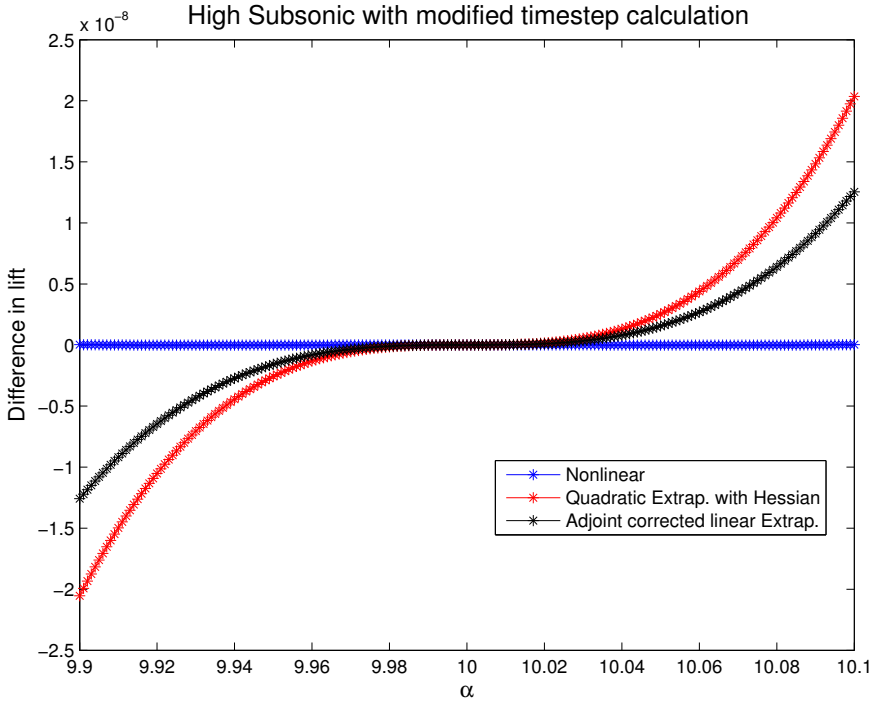

Figure 4. Comparison between quadratic and adjoint corrected linear extrapolation for high subsonic case (Mach = 0.65 and AOA = 10^o) with the modified time-step calculation

IX. Conclusion

Black-box use of automatic differentiation on fluid mechanics codes for Hessian calculation is still not computationally acceptable. An alternative computationally cheaper formulation for Hessian calculation has been described. Successful use of automatic differentiation has been demonstrated for generating all the necessary double differentiated routines. A `Makefile` can be generated to automatically keep in-sync with changes in the original nonlinear code. A set of checks have been introduced which at least point out any obvious inconsistencies in the nonlinear, linear and adjoint solutions used during the Hessian calculation.

A conscious effort is required while developing any nonlinear solvers to avoid inherently non-differentiable component functions. It will be increasingly difficult to track down these problems in complex codes. Adjoint corrected linear extrapolation seems to perform at least as well as the quadratic extrapolation. In non-smooth regions, adjoint corrected linear extrapolation closely follows the nonlinear trend in contrast to the quadratic extrapolation using the Hessian which may introduce relatively large errors.

Acknowledgments

This research was performed as part of the MCDO project funded by the UK Department for Trade and Industry and Rolls-Royce plc, and coordinated by Yoon Ho, Leigh Lapworth and Shahrokh Shahpar.

We are very grateful to Laurent Hascoët for making Tapenade available to us, and for being so responsive to our queries.

References

[1] "http://www.autodiff.org," .

[2] Greiwank, A., *Evaluating Derivatives*, SIAM, Frontiers in Applied Mathematics, 2000.

[3] Giles, M. B., Ghate, D., and Duta, M., "Using Automatic Differentiation for Adjoint CFD Code Development," *Indo-French Workshop*, Dec. 2005, Also available as NA05/25.

[4] Christianson, B., "Automatic hessians by reverse accumulation," *IMA Journal of Numerical Analysis*, Vol. 12, 1992, pp. 135–150.

[5] Sherman, L. L., Taylor III, A. C., Green, L. L., and Newman, P. A., "First- and Second-Order Aerodynamic Sensitivity Derivatives via Automatic Differentiation with incremental Iterative Methods," *Journal of Computational Physics*, Vol. 129, 1996, pp. 307–331.

[6] Giles, M. B., Duta, M. C., Muller, J. D., and Pierce, N., "Algorithm developments for discrete adjoint methods," *AIAA Journal*, Vol. 42(2), 2003.

[7] Hascoët, L., "http://www-sop.inria.fr/tropics," .

[8] Ghate, D. and Giles, M. B., "Source code for airfoil testcase for Hessian calculation using Tapenade http://www.comlab.ox.ac.uk/devendra.ghate/hessian/," .

[9] Giles, M. B. and Pierce, N. A., "Adjoint Recovery of Superconvergent Functionals from PDE Approximations," *SIAM Review*, Vol. 42(2), 2000, pp. 247–264.