Automatic Differentiation of a Time-Dependent CFD Solver for Optimisation of a Synthetic Jet

M. Tadjouddine[*1], **S.A.Forth**[1], and **N. Qin**[2]

[1] Applied Maths & Operational Research Group, Engineering Systems Department, Cranfield University, Shrivenham Campus, SN68LA, UK
[2] Department of Mechanical Engineering, The University of Sheffield, Mappin Street, S1 3JD, U.K.

Received 29/6/2005

Key words Automatic Differentiation, adjoint, synthetic jet flow
Subject classification 65D25,47A05,76G25

We use Automatic Differentiation to linearise a time-dependent CFD solver that is written in Fortran 95 and features mesh movement. The CFD code simulates a small device that generates a jet-like flow by oscillating fluid in a chamber connected to its surroundings via an orifice. Using the resulting derivative code, we discuss a case study aimed at optimising the kinetic energy of the upwards motion of the jet by controlling the period and the amplitude of the oscillation.

© 2005 WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim

1 Introduction

Synthetic jet actuators are small scale devices generating a jet-like motion by oscillating fluid in a chamber connected to an external flow via an orifice. They may be embedded into an aircraft wing to control flow separation or they can serve as a propulsion system for microfluid systems [2, 7]. The associated fluid mechanics is governed by the unsteady Navier-Stokes equations that are solved using a finite volume approach on a moving mesh [7]. Figure 1 shows the vorticity contours using a two-dimensional geometry, see [7] for further details.

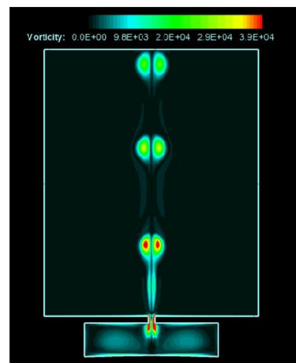

Fig. 1 Vorticity contours of the synthetic jet flow

In this study, we consider the synthetic jet actuator modelled in [7] and aim to maximise the kinetic energy of the jet in the upwards movement for a given fixed amount of work that cannot be exceeded by the system. The optimisation is carried out using a gradient-based algorithm from the MATLAB optimisation toolbox. Finite-differencing (FD) is a popular way of calculating derivatives. It is easy to implement but incurs truncation errors that may affect robustness of algorithms that use derivatives. An alternative is to use the algorithms and tools of Automatic Differentiation (AD) [5, 3], also known as Algorithmic or Computational Differentiation by which derivatives of a function represented by a computer program are computed without the truncation errors associated with finite-differencing. This facilitates the generation of the gradient of the objective function represented by some 4500 lines of Fortran 95 code. Performance of the AD-generated code is enhanced using manual modifications. Preliminary results of an optimisation test case aiming at maximising the kinetic energy of the upstream jet flow using the MATLAB optimisation routine `fmincon` have shown improved solution and convergence as compared to using FD.

* Corresponding author: e-mail: M.Tadjouddine@cranfield.ac.uk, Phone: +00 44 1793 785889, Fax: +00 44 1793 784196

© 2005 WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim

2 The Time-Dependent CFD Solver

In previous work [7], the unsteady Navier-Stokes were discretised using a cell-centered finite volume approach and solved on a moving mesh. Figure 2 shows the chamber containing the air flow. The diaphragm is forced to move in a sinusoidal oscillation with given period p and amplitude (as a percentage of its length) a. Boundary and internal mesh points $\mathbf{x}_i$ are moved into position $\mathbf{x}_i^{new}=\mathbf{x}_i+\Delta\mathbf{x}_i$, where $\Delta\mathbf{x}_i$ is prescribed for boundary mesh points and for internal mesh points $\Delta\mathbf{x}_i$ is given by,

$$\Delta\mathbf{x}_i = \frac{1}{D_i}\sum_{j\in\text{Nbr}(i)}\frac{1}{|\mathbf{x}_i-\mathbf{x}_j|}\Delta\mathbf{x}_j, \quad \text{with } D_i = \sum_{j\in\text{Nbr}(i)}\frac{1}{|\mathbf{x}_i-\mathbf{x}_j|}, \tag{1}$$

where $\text{Nbr}(i)$ represents the neighbouring points of i. This "smoothing" is repeated 50 times to ensure converged mesh movement.

In the numerical simulation, the residual $\mathbf{R}_i(\mathbf{q},\mathbf{x})$ for cell i is obtained using a finite-volume semi-discretisation of the Navier-Stokes equations in the form of $\partial V_i\mathbf{q}_i/\partial t = \mathbf{R}_i(\mathbf{q},\mathbf{x})$ where V_i and q_i are respectively the volume and conserved variables at cell i. By using a backward Euler scheme and introducing a pseudo-time step τ, we obtain the following nonlinear system for $\mathbf{q}^{n+1}$

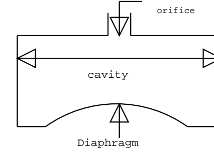

Fig. 2 Schematic of a synthetic jet flow

$$V_i^{n+1}\frac{\partial\mathbf{q}_i^{n+1}}{\partial\tau} = \mathbf{R}_i(\mathbf{q}_i^{n+1},\mathbf{x}^n) - \frac{V_i^{n+1}\mathbf{q}_i^{n+1}-V_i^n\mathbf{q}_i^n}{\Delta t}. \tag{2}$$

This system is solved by matching to steady state in τ using low-storage 4-stage Runge-Kutta scheme. The objective function is defined as the time averaged specific kinetic energy $F=\frac{1}{T_{max}}\int_{t=0}^{t=T_{max}}(u_k^2+v_k^2)dt$ where k is the index of a nominated cell in the freestream above the cavity, (u_k,v_k) is the cell's velocity vector and T_max is taken on 5 periods of the diaphragm motion.

3 Differentiation of the CFD Solver

Differentiating a code representing a function $\mathbf{F}:\mathbf{R}^n\mapsto\mathbf{R}^m$ can be viewed as a program transformation in which the original code's statements that calculate real valued variables are augmented with additional statements to calculate their derivatives. This is carried out using the techniques of Automatic Differentiation that regards the original computer code as a composition of elementary functions having at least a first derivative and then uses the chain rule to automatically evaluate the function represented by the given code as well as its derivative. Assuming $\text{d}x$ is the derivative associated with a variable x and x_1, x_2 the variables with respect to which we desire derivatives, the following statement can be transformed as follows:

$$v = v + x_1x_2 \qquad \begin{aligned}\text{d}v &= \text{d}v + 2(x_2\text{d}x_1 + x_1\text{d}x_2)\\ v &= v + 2x_1x_2\end{aligned} \tag{3}$$

In AD terminology, we define the *independent* variables to be those input variables with respect to which we need to compute the derivatives, the *dependent* variables to be those outputs whose derivatives are desired, and the *intermediate* variables to be those whose value depends on an independent and affects a dependent variable.

We distinguish at least two standard AD algorithms: the forward mode and the reverse or adjoint mode. The forward mode propagates directional derivatives along the control flow of the original program. The adjoint mode is composed of two passes: a forward pass that computes the function and a reverse pass that calculates the sensitivities of the dependent variables with respect to the intermediate and independent variables in the reverse order to their calculation in the function. In the reverse pass, we may need to recompute intermediate values required by the differentiation process or extract them from a storage data structure called the tape (essentially a LIFO stack) if they have been stored during the forward pass. The sensitivities of the dependent to the independent variables give the desired derivatives. The complexity analysis of these two algorithms can be found in [3]. The forward and adjoint modes AD are implemented in various AD tools such as ADIFOR 3.0, TAF, or TAPENADE 2.1, with variant strategies for performance purposes (see www.autodiff.org/ for references and details). Here, we have used TAPENADE 2.1 to calculate the gradient of the objective function F defined in Section 2.

© 2005 WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim

3.1 Results and Discussion

To use TAPENADE 2.1 [4] for our Fortran 95 code, we have 'cleaned up' the code using a sed script as in [6], since certain Fortran 95 features were not handled by TAPENADE 2.1. Namely, derived types are transformed to a set of arrays, static allocation is used in lieu of dynamic allocation and modules are transformed into common blocks. Furthermore, kernel routines of the input code were transformed by removing any overwriting of variables, arguments of routines made explicit, and we use local variables instead of accessing global array sections in a common block. The runtime of the resulting input code (indicated as "new" in Table 1) after these manual changes was decreased by about 40%. Then we calculated the gradient of the objective function $F : \mathbf{R}^2 \mapsto \mathbf{R}$ using the two equivalent input codes.

The obtained codes were compiled with maximum compilation optimisations and run on a Sun Blade 1000 machine. Performance results for a small mesh size 654 nodes and 582 cells are shown in Table 1.

Table 1 Performance of the AD-generated Code, the timings are in (s)

Method	CPU(∇F) (s)	Tape_size	$\frac{\text{CPU}(\nabla F)}{\text{CPU}(F)}$	Difference
TAPENADE 2.1(adj)	7774.6	1.585 GB	254.9	$\mathcal{O}(10^{-11})$
One-sided FD (new)	57.6	—	3.1	$\mathcal{O}(10^{-6})$
TAPENADE 2.1(new, fwd)	75.2	—	4.0	0.0
TAPENADE 2.1(new, adj)	450.8	0.12GB	24.2	$\mathcal{O}(10^{-11})$
TAPENADE 2.1(new, adj,FP)	384.2	0.09GB	20.7	$\mathcal{O}(10^{-6})$

Considering the TAPENADE 2.1(fwd) results as being correct, Table 1 shows the difference between AD and FD results is about 10^{-5} and FD has used 3 function evaluations to calculate the gradient. TAPENADE 2.1(new, adj) is 10 times faster than TAPENADE 2.1(adj) showing that simple changes to the input code can result in a huge performance improvement of the AD-generated code. A profile of the TAPENADE 2.1(adj)-generated code showed that about 90% of the CPU time is spent storing or retrieving data from the tape. This percentage dropped to 36% with TAPENADE 2.1(new, adj). However, TAPENADE 2.1(new, adj) is 5 times slower than TAPENADE 2.1(new, fwd). We believe this is due to the overheads incurred by the taping required by the adjoint mode AD. Note that TAPENADE 2.1 adjoint uses a recursive checkpointing strategy performed at subroutine levels to trade-off off between storage and recomputation of intermediate values. The last line of Table 1 shows the improvement in performance obtained by hand-modifying TAPENADE 2.1 adjoint code to store the converged nonlinear system and adjoining it [1]. Here, we have used the same number of iterations in the reverse pass as for the forward pass. The resulting derivative values are as precise as those of FD and the discrepancies with other AD results is, we believe, due to the finite convergence tolerance used in the nonlinear iteration at each backward Euler step. The runtime and memory requirement performance of the mechanical adjoining used by default in TAPENADE 2.1 are improved respectively by a percentage of 15% and 25%.

In AD theory [3], gradients are cheaply calculated using the adjoint mode when the number of independent variables is fairly larger. Here, we have two independents and the computation of the adjoint is dominated by the taping routines. The adjoint approach will be competitive when we have a large number of independent e.g., geometric design of a complex actuator.

4 Application to an Optimisation Test Case

We consider a case study aiming at maximising the kinetic energy of the upwards motion in a given cell of the mesh above the chamber by controlling the period p and the amplitude a of the oscillation. We constrained this optimisation problem by setting a maximum work that the system cannot exceed. This work $C(p,a)=\frac{1}{T_{max}}\int_0^{T_{max}}\int_{\Gamma=diaphragm} P(t)n(t).\frac{d\mathbf{x}}{dt}d\Gamma dt$ is obtained by integrating the surface force over the time steps. Our optimisation problem is,

$$\max F(p,a) \quad \text{subject to} \quad \begin{cases} C(p,a) \le 1 \\ 10^{-3} \le p \le 10^{-1} \\ 10^{-2} \le a \le 10^{-1} \end{cases} \tag{4}$$

© 2005 WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim

We used MATLAB's `fmincon` function to solve the above optimisation. The results are shown in Table 2. At the initial guess, we have $F(p,a)=2.65D-4$ and $C(p,a)=8.73D-3$. We observed `fmincon` using the gradients supplied by AD converged to a better solution in 11 iterations than that obtained using FD in 13 iterations. By taking the solution obtained by `fmincon` using FD as an initial guess, the optimiser converged in 5 iterations to the optimal solution obtained by `fmincon`(AD). We believe fast and accurate first derivatives provided by AD in the optimisation has increased robustness of the `fmincon` routine.

Table 2 Optima calculated by MATLAB's `fmincon`

	Initial Guess		Optimum		Opt. Value	
Method	p	a	p	a	F(p,a)	Iter.
`fmincon`	1.D-2	1.D-2	0.0024	0.0229	0.0389	13
`fmincon`(AD)	1.D-2	1.D-2	0.01	0.1	1.4535	11

5 Conclusion and Future Work

We have presented preliminary results in using AD to produce an adjoint solver for a time-dependent CFD code. This shows that AD is a useful tool as it facilitates the automatic generation of adjoint code and can enhance robustness of an optimisation algorithm that requires derivatives. Furthermore, we have shown that simple changes of the input code prior to differentiation can greatly enhance the efficiency of the AD-generated code. The "better" the function is coded, the faster the AD calculated derivative will be.

Future work include optimising the geometry of the chamber. This will imply increasing the number of independent variables and therefore the AD adjoint mode will be of great benefit. We will also investigate checkpointing schemes that can further enhance the performance of the adjoint code and then run the resulting adjoint solver using finer meshes.

Acknowledgements This work is supported by EPSRC under grant GR/R85358/01.

References

[1] B. Christianson. Reverse accumulation and attractive fixed points. *Optimization Methods and Software*, 3:311–326, 1994.

[2] Q. Gallas, G. Wang, M. Papila, M. Sheplak, and L. Cattafesta. Optimization of synthetic jet actuators. *AIAA paper*, (0635), 2003.

[3] A. Griewank. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Number 19 in Frontiers in Appl. Math. SIAM, Philadelphia, PA, 2000.

[4] L. Hascoët and V. Pascual. *The TAPENADE 2.1 user's guide*. Inria Sophia Antipolis, Tropics Project, 2004, Route des Lucioles, 09902 Sophia Antipolis, France, Jan 2005. http://www-sop.inria.fr/tropics/.

[5] L. B. Rall. *Automatic Differentiation: Techniques and Applications*, volume 120 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1981.

[6] M. Tadjouddine, S. A. Forth, and A. J. Keane. Adjoint differentiation of a structural dynamics solver. In M. Bücker, G. Corliss, P. Hovland, U. Naumann, and B. Norris, editors, *AD2004: Proceedings of the 4th International Conference on Automatic Differentiation*, Lecture Notes in Computational Science and Engineering, page To appear. Springer, 2005.

[7] H. Xia and N. Qin. Dynamic grid and unsteady boundary conditions for synthetic jets flow. *43rd AIAA Aerospace Sciences Meeting and Exhibit, Reno, Nevada*, (AIAA 2005-106), 2005.

© 2005 WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim