Fixed-Point solution of discrete adjoint for SIMPLE-type incompressible solvers

Siamak Akbarzadeh*
School of Engineering and Materials Science (SEMS)
Queen Mary, University of London, Mile End Rd, London, E1 4NS, UK
Email: s.akbarzadeh@qmul.ac.uk

Jens-Dominik Müller
School of Engineering and Materials Science (SEMS)
Queen Mary, University of London, Mile End Rd, London, E1 4NS, UK
Email:j.mueller@qmul.ac.uk

Summary

The derivation of incompressible adjoint CFD codes through automatic differentiation (AD) is discussed with focus on SIMPLE-like incompressible solvers. Based on the algebraic form of the segregated SIMPLE scheme, a fixed-point form of the discrete adjoint is derived. Using the coefficient matrix of the original linearised flow equations (primal) and writing the adjoint driver code as close as possible to the primal, to improve the performance. Moreover, one can use the new approach in a one-shop optimisation method in which the adjoint solution iteration can be started from the adjoint solution obtained in the previous design step. Using Tapenade[1] as an AD tool, the approach is implemented in the in-house incompressible flow solver GPDE Final results will also demonstrate the implementation of the method withe application of dco/c++[2] to the open-source CFD package, OpenFOAM.

Keywords: *CFD, SIMPLE algorithm, discrete adjoint, shape optimisation.*

1 Introduction

Adjoint solvers are an essential ingredient for sensitivity applications with CFD: shape and topology optimisation, parameter estimation, mesh adaptation and uncertainty quantification. Continuous formulations have been pioneered by Jameson,[3] where the adjoint equations are derived and then discretised. Major issues with the continuous approach are a) the significant effort in keeping the adjoint up to date with its complementary flow code, b) the lack of understanding of the stability of the discretised adjoints and c) the fact that due to the re-discretisation the sensitivities are not guaranteed to vanish when the flow discretisation (the primal) predicts a minimum for the objective function.

As an alternative discrete formulations have been rising in popularity where the discretised equations of the flow solver are differentiated and transposed.[4,5] The discrete approach guarantees convergence if the primal is convergent which is a major attraction of the approach. This however has to be weighed against the fact that for complex industrial configurations the flow typically does not fully converge with standard iterative approaches, which may lead to a divergent adjoint,[6] hence stronger solvers are often required.

The discrete adjoint approach can be automated using AD tools,[1,2] which bears the aspiration that the generation of the adjoint code may be fully automated. However, in practice, developing discrete adjoint codes using AD often is no less painful and labour-intensive than hand-differentiated discrete codes: the preparation of the code may be significant to make it parse-able with AD tools and achieve acceptable runtime performance and especially memory footprint. Significant advances however have been achieved with compressible codes that use fully-coupled discretisations[5] and manually constructed fixed-point time-stepping loops that minimise the memory requirements.

Using the source-transformation AD tool Tapenade,[1] the discrete adjoint for the incompressible SIMPLE algorithm[7] has been presented in the literature.[8,9] However, their 'brute-force' application of AD which submits the entire inner iterative loop to the AD tool leads to a significant overhead in memory and runtime. Moreover, due to the accumulation of sensitivities from zero adjoint the approach cannot make use of solutions from previous design steps in 'one-shot' optimisation algorithms, but has to accumulate starting from a zero adjoint solution.

A continuous adjoint formulation for SIMPLE-type incompressible flow discretisations, has been introduced and implemented in OpenFOAM[10] which is being widely used for industrial application with topology optimisation.[11] The discrete adjoint of OpenFOAM has been presented by Towara et al.[12,13] Since OpenFOAM is written in advanced C++ language with use of high-level syntax, no source-transformation AD tool is currently able to differentiate it. Due to this reason, Towara et al. use the operator-overloading (O-O) tool dco/c++ of RWTH Aachen[2] which is adapted to parse and differentiate the code but constrained by high memory demand due to the nature of o-o tools which do reading and writing memory operations ('taping').

Stück et al.[14] have proposed a hybrid of continuous and discrete approaches where they reuse some of the operators of the primal, but derive and re-discretise boundary conditions. Here we derive a fixed-point form[15] of the SIMPLE algorithm that allows 'hot-starts' with arbitrary adjoint solutions. This formulation uses the transpose of the operators and coefficient matrices from the primal, as shown by Stück,[14] to solve the adjoint equation in a fixed-point iteration algorithm that can be regarded as SIMPLE adjoint loop. In contrast to Stück approach in which the boundary conditions and Adjoint Transpose Convection (ATC) are treated continuously, here we compute those terms by either application of AD or hand differentiation.

The method is implemented in an in-house SIMPLE-type incompressible solver code called GPDE which is written in FORTRAN language. Since the code can be differentiated by source transformation AD tools, Tapenade is used to differentiate the cost function, the internal and boundary fluxes to compute the right hand side of momentum and pressure equations. Then, the transpose of coefficient matrices and operators from primal and the differentiated subroutines are combined to write the fixed-point adjoint driver code close to the primal SIMPLE loop.

In the final application to OpenFOAM such straight forward steps might not be doable. Due to nature of o-o tool like dco/c++, all differentiated functions and subroutines are written on the memory tape and accessing and reusing them in a different driver would be difficult. The solution might be the following steps: a)apply dco/c++ to compute the sensitivity of objective function with respect to flow variables b)transpose the matrix coefficients and operators from primal c)hand differentiate all internal and boundary fluxes for ATC c)assemble them in adjoint driver.

The paper presents the formulation in Sec. 2, followed by Sec. 3 which presents the discrete fixed-point adjoint approach. Comparisons of the brute-force and the fixed-point approaches for validation are shown in Sec. 4.

2 Mathematical Overview

2.1 *Incompressible Flows*

The Navier-Stokes(NS) governing equations for steady, laminar incompressible Newtonian fluid flow can be written as

$$\bigtriangledown . \rho \boldsymbol{U} = 0 \tag{1}$$

$$\bigtriangledown . (\rho \boldsymbol{U}\boldsymbol{U}) = -\bigtriangledown p + \bigtriangledown \cdot (\mu \boldsymbol{U}) \tag{2}$$

Using a Picard linearisation, the governing equations can be expressed in a discretised form by representing the variables at the control volume faces in terms of cell centre values. The discretised continuity and momentum equations can be formulated in matrix view as

$$\underbrace{\begin{bmatrix} \mathbf{F} & \mathbf{B} \\ \mathbf{C} & 0 \end{bmatrix}}_{\mathbf{A}} \cdot \underbrace{\begin{bmatrix} \boldsymbol{u} \\ p \end{bmatrix}}_{\boldsymbol{W}} = \underbrace{\begin{bmatrix} b_1 \\ b_2 \end{bmatrix}}_{b} \tag{3}$$

where $\mathbf{F}$, $\mathbf{B}$ and $\mathbf{C}$ are the convection-diffusion, gradient and divergence operators. In matrix notation the SIMPLE algorithm can be summarised as

$$\mathbf{M}\boldsymbol{y} = b \quad ; \quad \underbrace{\begin{bmatrix} \mathbf{F} & 0 \\ 0 & \mathbf{S} \end{bmatrix}}_{\mathbf{M}} \begin{bmatrix} \boldsymbol{u}^* \\ p' \end{bmatrix} = \begin{bmatrix} b_1 - \mathbf{B}p' \\ b_2 - \mathbf{C}\boldsymbol{u}^* \end{bmatrix} \tag{4}$$

$$\boldsymbol{W} = \mathbf{L}\boldsymbol{y} \quad ; \quad \begin{bmatrix} \boldsymbol{u} \\ p \end{bmatrix} = \underbrace{\begin{bmatrix} \mathbf{I} & -\mathbf{D}^{-1}\mathbf{B} \\ 0 & \mathbf{I} \end{bmatrix}}_{\mathbf{L}} \begin{bmatrix} \boldsymbol{u}^* \\ p' \end{bmatrix} \tag{5}$$

here $\mathbf{I}$ is the identity matrix, $\mathbf{D}^{-1}$ is the inverse of diagonal matrix of the operator $\mathbf{F}$ ($\mathbf{D}^{-1} = \tilde{\mathbf{F}}^{-1}$) and $\mathbf{S}$ is the Schur complement:

$$\mathbf{S} = -\mathbf{C}\mathbf{D}^{-1}\mathbf{B} \tag{6}$$

As it is clear in Eqn. 4, the gradient of pressure in the momentum equation and the velocity divergence in the pressure equation are treated explicitly on the right hand side alongside boundary terms. The SIMPLE method can be considered a right block preconditioner with $\mathbf{M}$ and $\mathbf{L}$ as the preconditioning matrices. Having described the matrix view of SIMPLE algorithm, the algebraic formula of it as a fixed-point iteration can be obtained as

$$\boldsymbol{W}^{n+1} = \boldsymbol{W}^n - \mathbf{L}\mathbf{M}^{-1}(\mathbf{A}\boldsymbol{W}^n - b), \tag{7}$$

where $\boldsymbol{W}$ represents the flow variables, $\mathbf{L}\mathbf{M}^{-1}$ is the preconditioner and the term $(\mathbf{A}\boldsymbol{W}^n - b)$ is the spatial residual, $\mathbf{R}$, that is driven to zero by the flow solver.

2.2 *Tangent Linear Differentiation*

In the context of shape optimisation, the general discrete form of NS equations that is solved is of the form

$$\mathbf{R}(\boldsymbol{W},\boldsymbol{X}(\alpha)) = 0, \tag{8}$$

where $\boldsymbol{X}$ is the vector of the computational grid points coordinates and α are a set of design parameters in the design process to control the coordinates of grids to optimise a scalar objective function, J,

$$\alpha \quad \longrightarrow \quad \boldsymbol{X} \quad \longrightarrow \quad \boldsymbol{W} \quad \longrightarrow \quad J$$

To use any gradient-based shape optimisation, the sensitivity of objective function with respect to the design variables is needed in each design step,

$$\frac{\partial J}{\partial \alpha} = \frac{\partial J}{\partial \boldsymbol{W}}\frac{\partial \boldsymbol{W}}{\partial \boldsymbol{X}}\frac{\partial \boldsymbol{X}}{\partial \alpha}. \tag{9}$$

In CFD, the flow solution $\boldsymbol{W}$ is implicitly dependent on the grid coordinates through a set of nonlinear equations and regarding Eq. 8 one can write

$$\frac{\partial \mathbf{R}}{\partial \alpha} = \frac{\partial R}{\partial \boldsymbol{W}}\dot{\boldsymbol{W}} + \frac{\partial \mathbf{R}}{\partial \boldsymbol{X}}\dot{\boldsymbol{X}} = 0, \tag{10}$$

where

$$\dot{\boldsymbol{W}} = \frac{\partial \boldsymbol{W}}{\partial \boldsymbol{X}}\dot{\boldsymbol{X}}, \quad \dot{\boldsymbol{X}} = \frac{\partial \boldsymbol{X}}{\partial \alpha}$$

Although the sensitivity of flow variables, $\dot{\boldsymbol{W}}$, can be computed with AD tools and in forward mode but the calculation needs to be repeated for every single design variable in each design step. That's why, in gradient-based shape optimisation applications, in which the number of design variables is relatively large, the tangent linear is not an affordable option for gradient calculation. The corresponding tangent linear differentiation of SIMPLE algorithm can be formulated as

$$\dot{\boldsymbol{W}}^{n+1} = \dot{\boldsymbol{W}}^{n} - \mathbf{L}\mathbf{M}^{-1}(\mathbf{A}\dot{\boldsymbol{W}}^{n} - f), \tag{11}$$

where

$$f = -\frac{\partial \mathbf{A}}{\partial \boldsymbol{W}}\dot{\boldsymbol{W}}^{n-1}\boldsymbol{W} - \frac{\partial \mathbf{A}}{\partial \boldsymbol{X}}\dot{\boldsymbol{X}} + \frac{\partial b}{\partial \alpha}$$

2.3 *Discrete Adjoint Differentiation*

Regarding the Eq. 9 and Eq. 10, the cost function sensitivity calculation can be rewritten in a different way which is much cheaper computationally compared to tangent linear

$$\frac{\partial J}{\partial \alpha} = -\frac{\partial J}{\partial \boldsymbol{W}}\left[\frac{\partial \mathbf{R}}{\partial \boldsymbol{W}}\right]^{-1}\frac{\partial \mathbf{R}}{\partial \boldsymbol{X}}\dot{\boldsymbol{X}} \tag{12}$$

and an AD tool can compute the sensitivity in adjoint/reverse mode

$$\frac{\partial J}{\partial \alpha} = -\dot{\boldsymbol{X}}^{T}\left[\frac{\partial \mathbf{R}}{\partial \boldsymbol{X}}\right]^{T}\left[\frac{\partial \mathbf{R}}{\partial \boldsymbol{W}}\right]^{-T}\left[\frac{\partial J}{\partial \boldsymbol{W}}\right]^{T}. \tag{13}$$

In this way any change in design parameter only affects the left-most term which is evaluated last. Moreover, the other three terms need only be evaluated once unless there is a change in functional J or a change in flow field $\boldsymbol{W}$. Introducing the transpose of adjoint variable as

$$v^{T} = \frac{\partial J}{\partial \boldsymbol{W}}\left[\frac{\partial \mathbf{R}}{\partial \boldsymbol{W}}\right]^{-1},$$

and the adjoint equation is obtained as

$$\left[\frac{\partial \mathbf{R}}{\partial \boldsymbol{W}}\right]^{T} v = \frac{\partial J}{\partial \boldsymbol{W}} \tag{14}$$

2.4 *Discrete Adjoint of SIMPLE algorithm (brute-force)*

It can be shown[15] that the automatic differentiation of Eq. 7 in reverse mode leads to

$$\begin{aligned}\overline{\boldsymbol{W}}^{n} = {} & \overline{\boldsymbol{W}}^{n+1} - \mathbf{A}^{T}\mathbf{M}^{-T}\mathbf{L}^{T}\overline{\boldsymbol{W}}^{n+1} \\ & - \left[\frac{\partial \mathbf{A}}{\partial \boldsymbol{W}}\right]^{T}\mathbf{M}^{-T}\mathbf{L}^{T}\overline{\boldsymbol{W}}^{n+2}\boldsymbol{W}^{T} \\ & + \left[\frac{\partial b}{\partial \boldsymbol{W}}\right]^{T}\mathbf{M}^{-T}\mathbf{L}^{T}\overline{\boldsymbol{W}}^{n+2}\end{aligned} \tag{15}$$

and subject to the final condition which is

$$\overline{\boldsymbol{W}}^{N} = \frac{\partial J}{\partial \boldsymbol{W}} = g$$

the adjoint variables can be accumulated using the following formula[15] :

$$v = \sum_{m=0}^{N-1}\mathbf{M}^{-T}\mathbf{L}^{T}\overline{\boldsymbol{W}}^{m+1} \tag{16}$$

The general structure of an SIMPLE-type incompressible solver code which is differentiated with an AD tool can be summarised as following steps

```
call  $Initialise\ the\ flow\ (\boldsymbol{W})$
call  $Metrics\ (Normal\ vectors,\ volumes,\ etc.)$
call  $SIMPLE$ :
            $Do\ \ nIter = 1, mIt$
                call  $Momentum\ Eqn.$
                call  $Pressure\ Eqn.$
                call  $Update$
            $End\ DO$
call  $Cost\_Func: \ J$
call  $\overline{Cost\_Func}: \ \frac{\partial J}{\partial \boldsymbol{W}} = \overline{\boldsymbol{W}}^N = g$
call  $\overline{SIMPLE}: \ \left[\frac{\partial \mathbf{R}}{\partial \boldsymbol{X}}\right]^T \sum \mathbf{M}^{-T}\mathbf{L}^T\overline{\boldsymbol{W}}$
            $Do\ \ nIter = 1, mIt$
                call  $\overline{Update}$
                call  $\overline{Pressure\ Eqn.}$
                call  $\overline{Momentum\ Eqn.}$
            $End\ DO$
call  $\overline{Metrics}: \ \dot{\boldsymbol{X}}^T \left[\frac{\partial \mathbf{R}}{\partial \boldsymbol{X}}\right]^T v$
```

The adjoint variable can be accumulated during the reverse SIMPLE loop ()with a small code modification.

Using AD in a brute-force fashion, which was illustrated above, would generate a sensitivity code that is not the counterpart of primal fixed-point iteration. Due to this reason, it can not be used in one-shot optimisation method which converges flow, gradient and design variables simultaneously, resulting in further efficiency gains. Moreover, because of taping/check-pointing of primal which is needed by AD tools the memory and runtime demand is high when running large scale industrial cases.

3 Fixed-Point Discrete Adjoint of SIMPLE algorithm

Alternatively, using AD tool and a new form of manually assembling the differentiated blocks of code, it is possible to obtain the discrete adjoint solution of SIMPLE-like algorithms as a fixed-point iteration. Then the derived discrete adjoint formulation is proved to have the same asymptotic convergence rate as the primal code.[5] because the rate of convergence in adjoint equation is governed by matrix $\mathbf{I}-\mathbf{M}^{-T}\mathbf{L}^T$ which its eigenvalues are the same as its counterpart matrix for primal, $\mathbf{I}-\mathbf{L}\mathbf{M}^{-1}$. Considering the adjoint accumulation formula, Eq. 16, with $v^N = 0$, the adjoint difference can be written as

$$\begin{aligned}
v^n - v^{n+1} &= \mathbf{M}^{-T}\mathbf{L}^T\overline{\boldsymbol{W}}^{n+1} \\
&= \mathbf{M}^{-T}\mathbf{L}^T\overline{\boldsymbol{W}}^{n}(\overline{\boldsymbol{W}}^{N} - \sum_{m=n+1}^{N-1}(\overline{\boldsymbol{W}}^{m+1} - \overline{\boldsymbol{W}}^{m})) \\
&= \mathbf{M}^{-T}\mathbf{L}^T(g - \sum_{m=n+1}^{N-1}(\mathbf{A}^T\mathbf{M}^{-T}\mathbf{L}^T\overline{\boldsymbol{W}}^{n+A1} \\
&\qquad + (\frac{\partial \mathbf{A}}{\partial \boldsymbol{W}})^T\mathbf{M}^{-T}\mathbf{L}^T\overline{\boldsymbol{W}}^{n+2}\boldsymbol{W}^T \\
&\qquad - (\frac{\partial b}{\partial \boldsymbol{W}})^T\mathbf{M}^{-T}\mathbf{L}^T\overline{\boldsymbol{W}}^{n+2})) \\
&= -\mathbf{M}^{-T}\mathbf{L}^T(\mathbf{A}^T v^{n+1} - (g - (\frac{\partial \mathbf{A}}{\partial \boldsymbol{W}})^T v^{n+2}\boldsymbol{W}^T \\
&\qquad + (\frac{\partial b}{\partial \boldsymbol{W}})^T v^{n+2}))
\end{aligned} \tag{17}$$

Looking at Eq. 17, the block matrix $\mathbf{A}^T$ is just the transpose of the matrix $\mathbf{A}$ which has already been calculated in primal. It means one can use the benefit of having the matrix $\mathbf{A}$ and reassemble its transpose for the adjoint equation. This saves the memory usage needed for check-pointing matrix $\mathbf{A}$. The term that has been subtracted from g is the lagged adjoint variable due to Picard Linearisation. To solve Eq. 17, the SIMPLE algorithm can be adapted to be used as a preconditioner for the adjoint equation.[14]

The method is implemented in the in-house flow solver code called GPDE which is written in FORTRAN language based on cell-centred SIMPLE scheme. The subroutine *spd_conv_diff* in GPDE calculates the internal and boundary fluxes to build the matrix coefficient for momentum equation in primal. To compute the ATC of the adjoint momentum equation, this subroutine is differentiated with TAPENADE in reverse mode (*spd_conv_diff_b*) to be used in the SIMPLE adjoint driver. Moreover, the objective function subroutine is differentiated to calculate the term $\frac{\partial J}{\partial \boldsymbol{W}}$. Then, in the adjoint driver code, the subroutine *objective_b* is called just before the SIMPLE adjoint loop to add term 'g' to the right hand side both momentum and pressure adjoint equations.

4 RESULTS

4.1 Cavity Flow

To aid understanding, the method first was implemented in a validated basic pressure-correction in-house code written in the C-language. The source-transformation AD tool Tapenade[1] is used for differentiation. As a test case, the well-known lid-driven cavity flow with $Re = 1000$ is considered. The momentum source term is perturbed at a specific point and the objective function is defined as the average of squared velocity magnitude over the whole domain.

A sample of velocity adjoint over centre of cavity, $x = 0.5$, is tabulated in table 1 in both methods. As it is clear,

Fixed-Point	Tapenade brute-force
0.0	0.0
-0.000005168	-0.000002527
-0.000043189	-0.000046429
-0.000084630	-0.000085385
...	...
0.000363607	0.000365744
0.000317863	0.000319150
0.000262002	0.000258612
0.000081122	0.000082703
0.0	0.0

Table 1: Velocity adjoint distribution at x = 0.5

the adjoint velocity values gained by the new approach is in agreement with the brute-force differentiation of the solver by 4-5 digits after decimal point.

4.2 S-Bended Duct Flow

To verify the method in more complex problems, an internal laminar flow through a s-bended duct is examined. The velocity adjoint field on a plane in the middle of the duct for both methods is shown in figures 1 and 2. The quantitative comparison will be demonstrated in the final work.

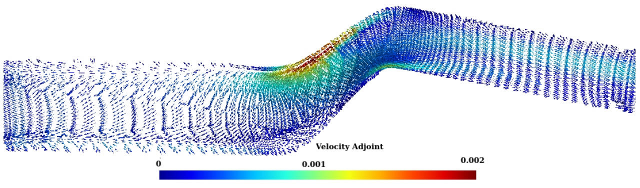

Figure 1: Velocity adjoint field gained by fixed-point approach

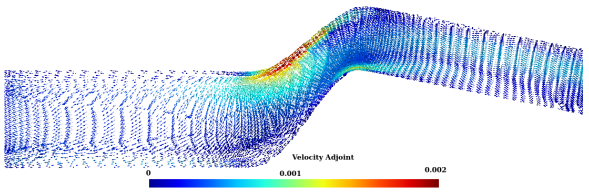

Figure 2: Velocity adjoint field gained by brute-force approach

5 Conclusion

In this paper it was shown that with some algebraic manipulation the brute-force differentiation of SIMPLE-like solver can be transformed to the fixed-point iteration solution of adjoint equation. The bulk of the adjoint equation was built by using the transpose of the primal matrices. The method was tested in an in-house code. Some subroutines of the primal driver code were selectively differentiated by TAPENADE AD tool and were assembled in the adjoint driver which was written very close to the primal code. The method was verified with the result of two simple test cases.

The final paper will demonstrate the fixed-point approach applied to OpenFOAM, with appropriate parts differentiated either by hand or where possible using the `dco++` AD tool.[2] The resulting discrete adjoint solver will be applied to relevant industrial testcases from the About Flow project.

6 Acknowledgements

This work is conducted within the About Flow project (http://aboutflow.sems.qmul.ac.uk) which has received funding from the European Union's Seventh Framework Programme for research, technological development and demonstration under Grant Agreement No. 317006.

References

[1] Hascoët, L. and Pascual, V. The Tapenade Automatic Differentiation tool: Principles, Model, and Specification. *ACM Transactions On Mathematical Software* **39**(3) (2013).

[2] Lotz, J., Leppkes, K., and Naumann, U. dco/c++ – efficient derivative code by overloading in c++. Tech. rep. aib-2011-05, RWTH Aachen, (2011). http://aib.informatik.rwth-aachen.de/2011/2011-05.pdf.

[3] Jameson, A., Martinelli, L., and Pierce, N. Optimum aerodynamic design using the Navier-Stokes equations. *Theor. Comp. Fluid. Dyn.* **10**, 213–237 (1998).

[4] Nielsen, E. J. *Aerodynamic Design Sensitivities on an Unstructured Mesh Using the Navier-Stokes Equations and a Discrete Adjoint Formulation.* PhD thesis, Virginia Polytechnic Institute and State University, December (1998).

[5] Giles, M. B., Duta, M. C., Müller, J.-D., and Pierce, N. A. Algorithm developments for discrete adjoint methods. *AIAA Journal* **41**(2), 198–205 (2003).

[6] Xu, S., Radford, D., Meyer, M., and Müller, J.-D. Stabilisation of discrete steady adjoint solvers. **submitted** (2014).

[7] Patankar, S. V. and Spalding, D. A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flow. *Int. J. of Heat and Mass Transfer* **15**(10), 1787–1806 (1972).

[8] Nemili, A., Özkaya, E., Gauger, N., Carnarius, A., and Thiele, F. Automatic generation of discrete adjoints for unsteady optimal flow control. In ECCOMAS CFD & Optimization, Tuncer, T., editor, (2011). ISBN 978-605-61427-4-1, No. 2011-043.

[9] Jones, D. and Müller, J.-D. Cfd development with automatic differentiation. *50th AIAA Aerospace Sciences Meeting* **AIAA-2012-573** (2012).

[10] Othmer, C. A continuous adjoint formulation for the computation of topological and surface sensitivities of ducted flows. **58**(8), 861–877 (2008).

[11] Othmer, C. Adjoint methods for car aerodynamics. *J. Math. Ind.* **14**(6) (2014).

[12] Towara, M., Sen, A., and Naumann, U. An effective discrete adjoint model for openfoam. In OPT-i: An International Conference on Engineering and Applied Sciences Optimization, Papadrakakis, M., Karlaftis, M., and Lagaros, N., editors, 1994. National Technical University of Athens, (2014).

[13] Sen, A., Towara, M., and Naumann, U. A discrete adjoint version of an unsteady incompressible solver for openfoam using algorithmic differentiation. In 6th European Conference on Computational Fluid Dynamics, nate, E. O., Oliver, J., and Huerta, A., editors, 5014–5023. ECCOMAS, (2014).

[14] Stück, A. and Rung, T. Adjoint complement to viscous finite-volume pressure-correction methods. *Journal of Computer Physics* **248**, 402 – 419 (2013).

[15] Giles, M. On the iterative solution of adjoint equations. In Automatic Differentiation of Algorithms, Corliss, G., Faure, C., Griewank, A., Hascoët, L., and Naumann, U., editors, 145–151. Springer-Verlag New York, Inc., New York, NY, USA (2002).