Sensitivity computation for ducted flows using adjoint of implicit pressure-velocity coupled solver based on Foam

Arindam Sen*
Software and Tools for computational Engineering (STCE)
RWTH Aachen University, Süsterfeldstraße 65, Aachen, 52072, Germany
Email: sen@stce.rwth-aachen.de

Markus Towara
Software and Tools for computational Engineering (STCE)
Email: towara@stce.rwth-aachen.de

Uwe Naumann
Software and Tools for computational Engineering (STCE)
Email: naumann@stce.rwth-aachen.de

Summary

Adjoint based methods have become the focus of research in CFD optimization pertaining to their low computational cost compared to other traditional methods. In principle, adjoints are computed via two methods: Continuous and Discrete. In the continuous method, adjoints are computed by deriving the adjoint equations of corresponding state variables and boundary conditions and then are solved using similar schemes as that of the primal. The discrete approach using Algorithmic Differentiation (AD) [9] is implemented either by source transformation or by an operator overloading tool. For an object-oriented code in C++ like OpenFOAM [10], the use of an operator overloading tool becomes imperative. Discrete adjoints based on operator overloading have the distinct advantage of robustness and stability while computing accurate gradients. Our experience however shows that adjoints of traditional SIMPLE (semi implicit method for pressure linked equations) [13] like algorithms have high computational cost even after employing standard performance improvement techniques such as binomial checkpointing [4] and analytic treatment of the linear solver [3]. Thus in this paper we present the discrete adjoint solver based on *puCoupledFoam* [7], an implicit pressure-velocity coupled incompressible steady state solver in Foam-extend(v3.1) [14].The objective is to exploit the faster rate of convergence of the flow equations in terms of runtime and number of non-linear solves to compute fast gradients. The solver is tested on related test problems to demonstrate significant performance improvement.

Keywords: *Adjoints, CFD optimization, Discrete adjoints, implicit coupled solvers.*

1 Introduction

Computing sensitivities i.e derivative of a desired objective function with respect to useful geometric parameters are the building blocks of Topology and Shape optimization. Classical methods such as Finite Differences scale with the size of the design space and hence often are too expensive to be affordable. Adjoint methods on the other hand are independent of the size of the design space and therefore present a more suitable alternative. Adjoint based CFD optimizations find extensive applications in the fields of Aerospace, Turbomachinery and automotive applications among others [6] [2] [12] and thus have evolved into a lucrative research topic. Two of the major ways of obtaining sensitivities are via the continuous and discrete based methods. The continuous methods typically provide the adjoint solution at the expense of two solver runs, one each for the primal and adjoint set of equations often utilizing similar numerical schemes with some additional tuning for stability of the adjoint solution. However development of continuous adjoint methods are deemed to be tedious and lacks adaptability with respect to different governing equations, boundary conditions and objective functions. Stability especially for problems of industrial relevance remains an area of concern and in some cases, as with the k-omega turbulence model, issues of non-differentiability are well known. Discrete based adjoints present an essential alternative. Using Algorithmic Differentiation (AD), the

sequence of operations of the primal can be reversed more or less in an automatic fashion that facilitates a robust and stable adjoint solution. The major strength of such a method is the flexibility across different flow solvers and cost functions. Sensitivity computation using adjoint methods can be carried out in the discrete way either by using Source transformation tools like TAPENADE [5] or by using an operator overloading tool like dco/c++ [8], as in our case. Often the solver restricts the choice of the AD tool, for example TAPENADE is well equipped to deal with codes written in Standard C or Fortran however for codes written in C++ that extensively use features of object orientation like templates and classes as in the case of OpenFOAM, operator overloading almost becomes the automatic choice. Interest in development of adjoint based methods is obvious from the fact that almost all leading commercial CFD solvers like ANSYS Fluent, STAR CCM+ alongwith major open-source solvers like SU2, OpenFOAM now have their own adjoint solvers. However, most of them are still in their early stages and none at this point can claim to tackle the complexities of real world engineering problems. Here we present the discrete adjoint version of Foam-extend(v3.1) building upon our experience with discrete adjoint OpenFOAM[15] [1]. Foam-extend is a kind of fork of OpenFOAM that retains many of its structural features with some additional extensions. Like OpenFOAM, it is a finite volume based solver written in C++. Some of the additional features include Mixing plane boundary for turbomachinery applications, density based roe-flux solver, dynamic mesh capability and GPU support via cuda solvers. In this paper however, we intend to focus on adjoints of *puCoupledFoam*, an implicit pressure-velocity coupled incompressible steady state solver based on explicit use of Rhie-Chow interpolation. We are also focussing on ducted flows where rather than computing sensitivities with respect to points on a parametrized surface, sesitivities are computed in the entire volume space. The section 2 gives a brief desciption of the physics of the problem, following which the black-box discrete adjoint formulation is underlined in some detail. The methodology is then tested on the *backwardFacingstepbend* test case where we see the temporal and memory complexities of the black-box approach. The results are verified using the continuous adjoint formulation. In section 3, standard performance improvement techniques such as binomial checkpointing [4] and symbolic treatment of the linear solver [7] is adapted to our implementations. These powerful improvements enable us to apply our implementation to a mid-size industrial test case, the S shaped airconditioning duct in section 4. We summarize our results in section 5.

2 The primal solution and the black-box approach

2.1 Governing Equations

The governing equation solved to demonstrate the functionality of discrete adjoint Foam-extend is incompressible steady state Navier-Strokes equation with an additional ressistance term. It needs to be mentioned here though that the functionality can be seamlessly extended to all kinds of flow solvers whether it be for incompressible or compressible flows, laminar or turbulent flows, steady or unsteady flows. The capability of discrete adjoint OpenFOAM to compute adjoints from unsteady flows has been previously demonstrated [1].

$$(v.\nabla)v = -\nabla p + \nabla.(2\nu D(v)) - \alpha v \ \textbf{(momentum)} \quad (1)$$

$$\nabla.v = 0 \ \textbf{(continuity)} \quad (2)$$

Here v denotes the velocity vector, p the pressure, ν the kinematic viscosity, and D(v) is the rate of strain tensor. α is the porosity. The value of α lies between 0 and 1. If α is zero for an individual cell, it implies that it is permeable and the flow passes through the cell without any hindrance, whereas if α is 1, it means the cell completely blocks the flow. We use α as our basic design parameter. Thus the volume sensitivities we compute are basically the gradient of our objective function with respect to α. This approach has been discussed in considerable detail in [11]. Two methods to solve the governing equations (1) and (2) are considered in this paper. One is the standard SIMPLE algorithm and the other is the coupled velocity-pressure approach that directly solves multiple equations and couplings in one matrix system.

The cost function considered in this paper is the total power loss between the inlet and outlet,

$$J = \int_{\tau} p + 0.5v^2 \, d\tau \quad (3)$$

To obtain the adjoints, we employ two basic methods. We then apply the discrete adjoint method to compute the adjoint sensitivities. These results are then verified against the continuous approach.

1. The discrete adjoint approach by Operator overloading: The basic steps of the method involves overloading all the basic datatype of Foam, i.e *scalar* with the appropriate dco adjoint data type, for example, dco::ga1s for first order adjoints. The following step involved writing adjoint drivers based on basic foam solvers such as *simpleFoam*, *puCoupledFoam* etc. These steps are described in more detail in section 3.

2. The continuous approach: The official version of OpenFOAM has the basic implementation of the continuous adjoint equations derived from the steady state Navier Strokes equation for internal flows. The application is adjointShapeOptimizationFoam where alongside the equations of momentum and continuity, we additionally solve their adjoint complements:

$$(v \cdot \nabla)u = -\nabla q + \nabla \cdot (2\nu D(u)) - \alpha u \ \textbf{(adjoint momentum)} \quad (4)$$

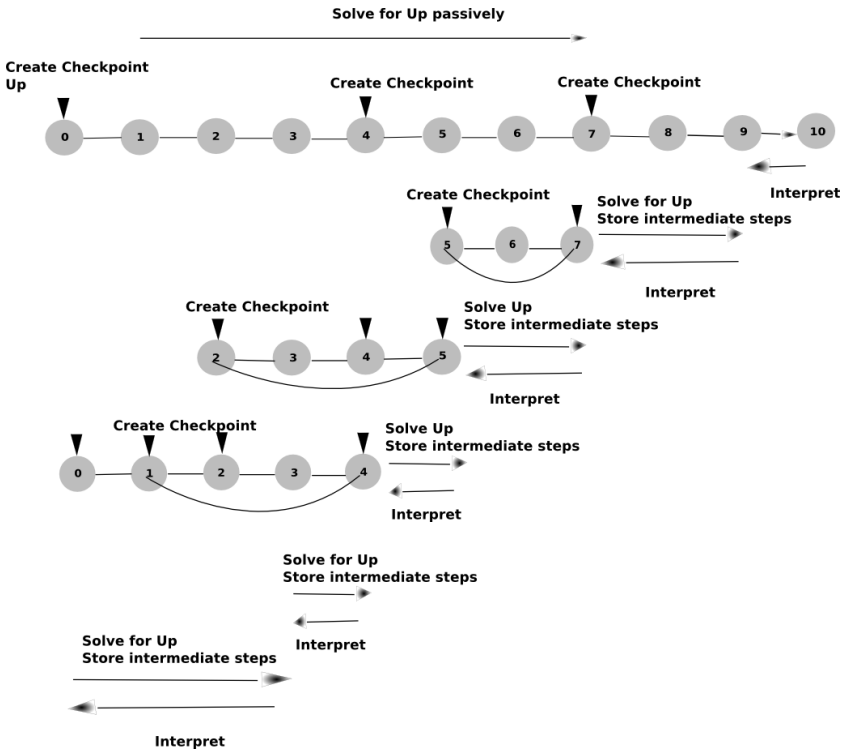

Figure 1: Binomial checkpointing applied to *puCoupledFoam*

$$\nabla \cdot u = 0 \quad \textbf{(adjoint continuity)} \tag{5}$$

where u denotes the adjoint velocity vector, q denotes the adjoint pressure. Other symbols remain the same as in equation (1) and (2).

2.2 Discrete adjoint Foam-extend

The basic steps of overloading the generic foam datatype and recompiling includes:
1. Put **dco** in foam/src.
2. Overloading operators, basic mathematical functions and data types
3. Include dco header files and linking to all the dco files
4. Replace the custom Foam datatype, `scalar` with the generic dco adjoint datatype (`dco::ga1s<double>`)

After performing the mentioned changes, theoretically Foam just needs to be recomplied. However, overloading with AD tools on such a large size code rarely, if ever, works out of the box. We need to adapt to the functionalities of AD tool such as removing unions in this case, removal of explicit namespacing and explicit casts do deal with assignment operations.

We then write the black-box adjoint driver based on *puCoupledFoam*. The generic set of guidelines for writing an adjoint solver based on *any* solver in Foam remains relatively similar.

1. Modify the momentum equation, UEqn.H to pluck in the porosity term as shown in Equation 1.

```
fvVectorMatrix UEqn
( fvm::div(phi, U) + turbulence->divDevReff(U)
+ fvm::Sp(alpha, U) );
```

2. Create tape and allocate memory for the tape

```
dco::ga1s<double>::global_tape =
dco::ga1s<double>::tape_t::create();
```

3. Initialize cost function

```
scalar J = 0;
```

4. Register the individual entries of alpha as inputs

```
for(int i=0; i<N; i++)
```

Solver	Primal RAM	Primal Time	Adjoint RAM	Adjoint Time
Decoupled without LST	139 MB	75s	906 MB	778s
Decoupled with LST	139 MB	75s	283 MB	758s
Coupled without LST	175 MB	11s	13GB	77s
Coupled with LST	175 MB	11s	303MB	62s

Table 1: Memory and Runtime for the *backwardFacingstepbend* case for coupled and decoupled solvers with and without Linear solver treatment

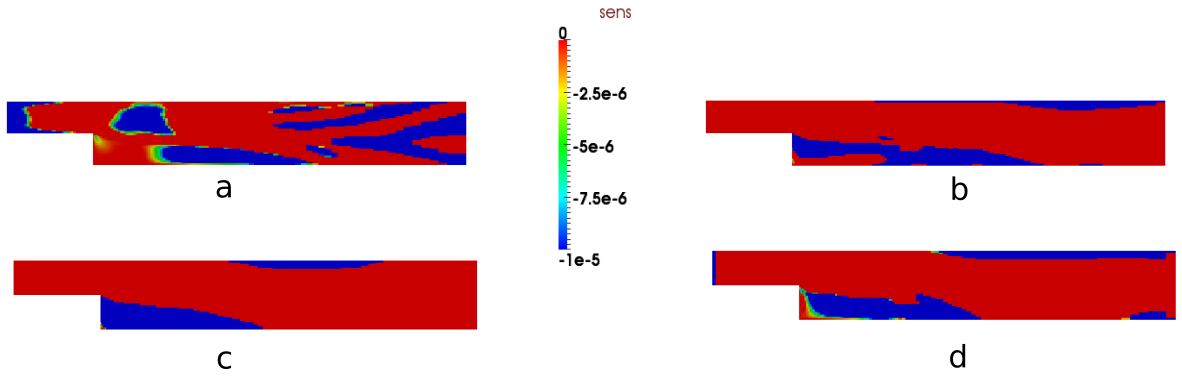

Figure 2: Sensitivities for the 'backwardFacingStepBend' test case, **a)** Continuous adjoints **b)** Decoupled discrete adjoint with Linear Solver treatment (LST), **c)** Blackbox coupled discrete adjoint, **d)** Coupled discrete adjoint with LST

dco::ga1s<double>::global_tape->register_variable(alpha[i]);

5. Obtain the primal solution by solving the coupled system

6. Evaluate the cost function from Equation (5)

This black-box discrete adjoint approach based on *puCoupledFoam* is applied to a small scale test case, *backwardFacingStepBend* .

The case is steady, laminar with a Reynolds number of 100. The The sensitivities obtained using the various approached are shown in Fig.2. The computational domain comprises of roughly 5000 grid points.

3 Performance improvement

3.1 Checkpointing

As often is the case, a black box approach is inadequate even for small size test cases. The amount of memory required for a black box approach is usually not available even on clusters. Performance improvement strategies are especially needed for computing adjoints of coupled solvers since the size of the system matrix is usually bigger compared to a decoupled solver. Also the number of iterations required for convergence of the block-aware linear solver such as the algebraic multigrid (AMG) or BiCGStab (Bi-conjugate gradient stabilized) in Foam-extend is typically large. The alternate option would be to recompute everything where only a single time-step (pseudo, as in this case) resides in the memory at any given instant of time. This approach pertaining to the tedious number of recomputations is often not efficient in terms of runtime. A useful tradeoff between these two approaches or rather between memory and runtime is achieved via checkpointing. The former approach can be perceived as one where the distance between two checkpoints is the entire pseudo-time iteration process, whereas in the case of the later, each pseudo-time step is checkpointed. The useful tradeoff lies between these two. A naive equidistant checkpointing scheme, where based on factors like the number of iterations and amount of memory available, checkpoints are placed at a fixed distance from one another. This method though useful, can be improved upon by employing Binomial Checkpointing using Revolve [4], where checkpoints are placed based on mathematical co-relations of the constituent factors like time steps and number of affordable checkpoints. Also, the Binomial checkpointing scheme re-uses checkpoints once they are freed which contributes to the efficiency in computation of adjoints during the reverse mode. Fig. 1 is a schematic diagram showing the process of Binomial checkpointing scheme applied to *puCoupledFoam*.

3.2 Analytic treatment of block matrix linear solver

Though the checkpointing scheme brings us to a decent compromise between memory and runtime, still the amount of memory required is rather high, especially if this framework is to be adaptable for medium to large scale industrial process. It has been noted that almost 90 % of the amount of memory required is consumed in recording the linear solver iterations, and this very expensive especially in the case of coupled solvers. For the *backwardFacingstepbend* test case, Fig. 3 denotes the number of linear solver iterations required in each pseudo-time iteration step for the problem to converge. This is particularly expensive for taping while using coupled linear solvers like the Algebraic Multigrid (AMG) since the number of linear solver iterations is usually large. There are basically two strategies that we can employ here. One is checkpointing of the linear solver and the second is analytic treatment of the linear solver [3]. The former usually overcompensates in runtime and hence may not be a good choice. In case of the later, we do not tape through the linear solver anymore. The tape is switched off during the forward run after the variables are registered and then we symbolically solve a different set of linear equations during the reverse mode usually by using the same linear solver.

The linear equation system solved during the forward run is:

Solver	Primal RAM	Primal Time	Tape Mem.	Adjoint Time
Decoupled	820 MB	80min(3.6T1)	10.3Gb	4.9h(4T2)
Coupled	1.3 GB	22min(T1)	10.1GB	1.2h(T2)

Table 2: Memory and Runtime for the S-bend test case

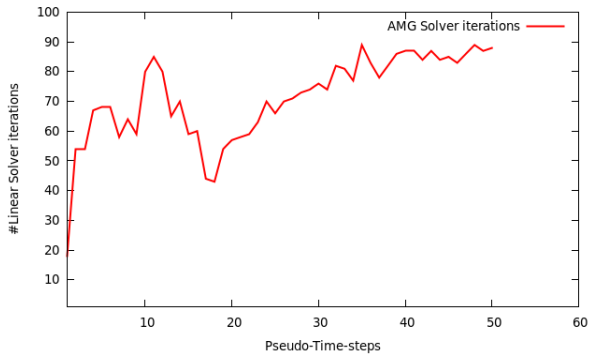

Figure 3: No. of linear solver iterations for the AMG solver

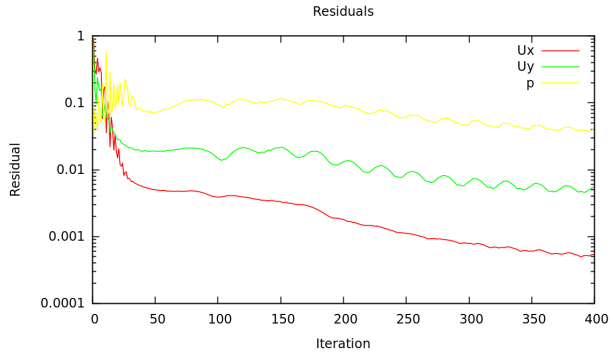

Figure 4: Convergence of the BiCG (bi-conjugate gradient) linear solver that poses problems for blackbox adjoint

$$A \cdot x = b \tag{6}$$

In this approach we do not tape the iterations of this linear solve during the forward run. We register the Block Matrix A and right hand side, b as inputs and x as output and perform the linear solve passively. During the reverse run, we receive incoming adjoints, $x_{(1)}$ from the tape. We then solve the following system of linear equations to fill the missing gap during the reverse run:

$$A^T \cdot b_{(1)} = x_{(1)} \tag{7}$$

$$A_{(1)} = -b_{(1)}^T \cdot x \tag{8}$$

As shown in Equation 7, the transpose of matrix A is required to solve the adjoint linear system. This might not be trivial in some cases but since the BlockIduMatrix format uses ldu addressing scheme in Foam, this can be performed with relative ease. Note that in Fig. 5 each block in Matrix A has 16 coefficients. During the registering of inputs during the forward run, each of the 16 coefficients have to be individually registered and correspondingly during the reverse run, adjoints of each coefficient is obtained by multiplying every coefficient in $b_{(1)}^T$ of order n*1 with each coefficent of x which is of order 1*n resulting in $A_{(1)}$ of order n*n.

Also, analytical treatment of the linear solver can be indispensable for obtaining correct adjoints when dealing with certain kinds of linear solver. It is now well known that a black-box approach for obtaining adjoints using Conjugate Gradient (CG) linear solver algorithm and it's variants may result in the divergence of the respective adjoints. The convergence of the primal for the test case under consideration is shown in Fig 4. The residuals seem to enter into a kind of limit cycle oscillations. Doing a black-box approach over this results in incorrect adjoints. The mathematical foundation for this is still the subject of ongoing investigation.

Table 1 shows the relative runtime and memory requirements with and without the linear solver treatment.

4 Case study: Verification and Validation

Three dimensional (3D) S-bend airduct for cabin ventilation: The flow regime here is laminar with a Reynolds number (Re) of 150. The computational domain is an unstructured grid of roughly 0.23 million cells. The boundary condition at the inlet and outlet is Drichlet, whereas the walls are no-slip no-penetration. The objective function in this case is the power loss between the inlet and the outlet as in Equation 3. Sensitivities, i.e. derivative of the objective function with respect to the porosities $alpha$, are computed using the discrete adjoint of implicit solver *puCoupledFoam*. These derivatives serve as the building blocks of the design optimization process. The desired sensitivity map is shown in Fig.6 . For comparison purpose, the sensitivities obtained by the continuous adjoint implementation in OpenFOAM, namely *adjointShapeOptimizationFoam* and discrete adjoint based on *simpleFoam* is also shown. Table 2 shows the relative performance of the two methods in terms of memory and runtime.

The negative sensitivities depict the region where optimization could place material. There seems to be a qualitative agreement between different approaches in terms of where optimization should place material. However exact quantitative consistency is not possible since the algorithms for obtaining the adjoints are different.

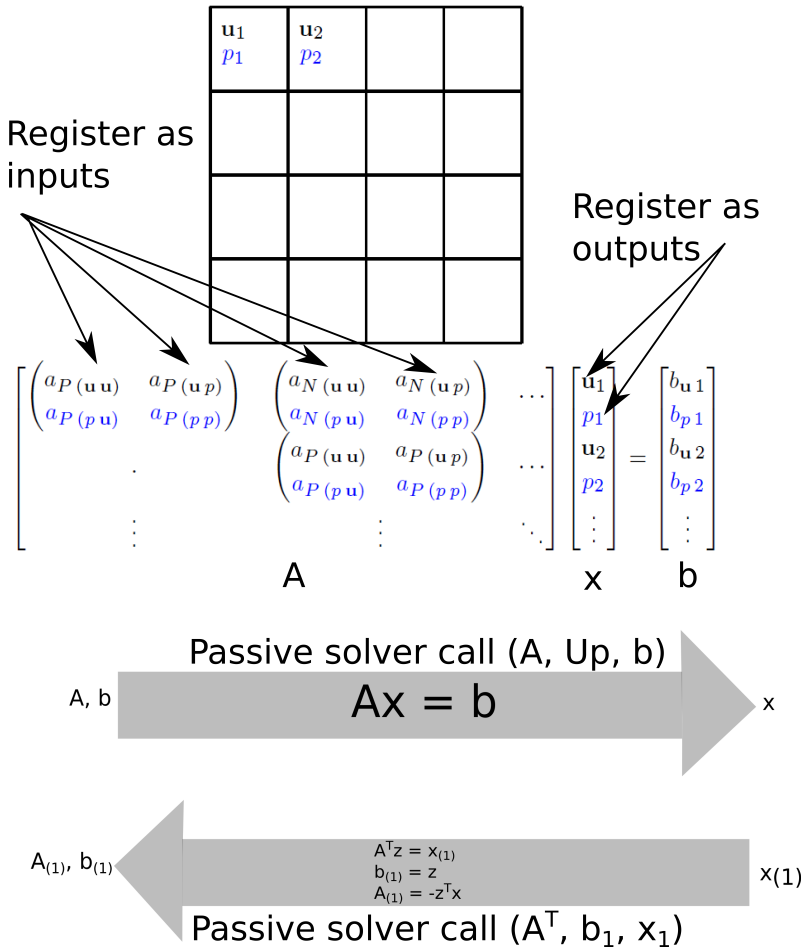

Figure 5: Schematic of the symbolic treatment of Block-Matrix linear solver

5 Conclusion

The speed-up in the computation of sensitivities using adjoint of coupled implicit solvers come from two sources. First is due to the faster convergence of the primal in terms of runtime, compared to a SIMPLE based decoupled solver. The later and the more significant contribution comes from the fact that the primal potentially converges in far fewer iterations of the non-linear solver. This is especially advantageous since we employ checkpointing on the pseudo-time steps and by reducing these pseudo-time steps, the recomputation cost is significantly reduced, thus assisting in the accelaration of the optimization process.

6 Acknowledgement

This work was funded by the EU through the FP7-PEOPLE-2012-ITN "AboutFlow" Grant agreement number:317006

References

[1] A.Sen, M.Towara, and U.Naumann. A discrete adjoint version of an unsteady incompressible solver for openfoam using algorithmic differentiation. *WCCM XI – ECCM V – ECFD VI, Barcelona, 2014*, 2014.

[2] Dirk Büche. Automated design optimization of compressor blades for stationary, large-scale turbomachinery. In *in Proceedings of the ASME/IGTI Turbo Expo 2003. ASME*, 2003.

[3] M. B. Giles. Collected matrix derivative results for forward and reverse mode algorithmic differentiation. In *Advances in Automatic Differentiation*, pages 35–44. Springer, 2008.

[4] A. Griewank and A. Walther. Algorithm 799: Revolve: An implementation of checkpointing for the reverse or adjoint mode of computational differentiation. *ACM Transactions on Mathematical Software*, 26(1):19–45, March 2000.

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

[6] Antony Jameson. Aerodynamic shape optimization using the adjoint method. In *VKI Lecture Series on Aerodynamic Drag Prediction and Reduction,*

von Karman Institute of Fluid Dynamics, Rhode St Genese, pages 3–7, 2003.

[7] K. Jareteg, V. Vukčević, and H. Jasak. pucoupledfoam- an open source coupled incompressible pressure-velocity solver based on foam-extend, 2014.

[8] J. Lotz, K. Leppkes, and U. Naumann. dco/c++ - derivative code by overloading in C++. *Aachener Informatik Berichte (AIB-2011-06)*, 2011.

[9] U. Naumann. *The Art of Differentiating Computer Programs. An Introduction to Algorithmic Differentiation.*, chapter 1&2. SIAM, 2012.

[10] OpenFOAM Ltd. OpenFOAM - The Open Source Computational Fluid Dynamics (CFD) Toolbox. `http://openfoam.com/`.

[11] C. Othmer. A continuous adjoint formulation for the computation of topological and surface sensitivities of ducted flows. *International Journal for Numerical Methods in Fluids*, 58(8):861–877, 2008.

[12] C. Othmer. Adjoint methods for car aerodynamics. *Journal of Mathematics in Industry*, 4:4–6, 2014.

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

[14] The OpenFOAM® Extend Project. Foam-extend-3.1. `http://sourceforge.net/p/openfoam-extend/foam-extend-3.1/ci/master/tree/ReleaseNotes.txt`.

[15] M. Towara and U. Naumann. A discrete adjoint model for OpenFOAM. *Procedia Computer Science*, 18(0):429 – 438, 2013. 2013 International Conference on Computational Science.

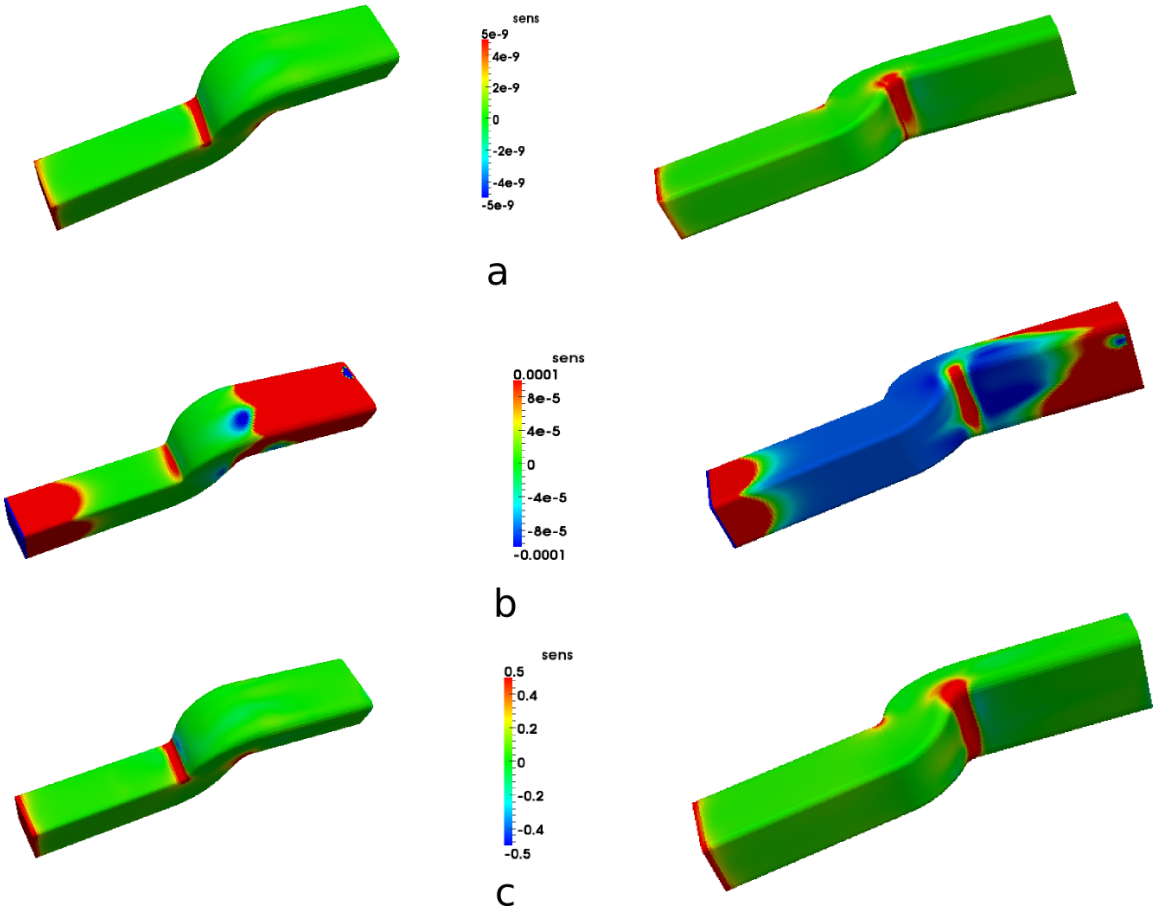

Figure 6: Sensitivities of the S-bend shaped duct, Adjoints obtained by **a, b)** continuous method, **c, d)** Discrete adjoint of coupled solver, **e, f)** Discrete adjoint of decoupled solver