DOI: 10.13009/EUCASS2019-831

8TH EUROPEAN CONFERENCE FOR AERONAUTICS AND AEROSPACE SCIENCES (EUCASS)

DOI: ADD DOINUMBER HERE

Sensitivity computations by automatic differentiation of a CFD code based on spectral differences.

José I. Cardesa[⋆†] *and Christophe Airiau*[⋆]
[⋆]*Institut de Mécanique des Fluides de Toulouse (IMFT)*
2 Allée du Professeur Camille Soula, 31400 Toulouse, France
jcardesa@imft.fr · christophe.airiau@imft.fr
[†]Corresponding author

3rd July 2019

Abstract

We compute flow sensitivities by modifying a CFD code which is spatially-discretized with spectral differences. Our discrete approach relies on algorithmic differentiation. We obtain two transformed codes, one for each differentiation mode : tangent and adjoint. Both codes compute sensitivities in an unsteady test case of two-dimensional incompressible flow inside a periodic cube with an initial double-shear profile. The sensitivities from both codes agree to within machine accuracy, and compare well with those approximated by finite difference computations. We discuss execution times and describe our strategy to automatically differentiate in adjoint mode a parallel code containing MPI instructions.

Introduction

The computation of sensitivities is a prerequisite to the implementation of optimization or flow control tools in simulation codes for aerodynamics. One route to estimate these sensitivities is to derive a new set of continuous equations, which need to be discretized in some way and added to the original simulation code.[13] An alternative path is to derive the discrete sensitivity equations by differentiating the discrete equations in the original simulation code. Two main advantages are often attributed to the second approach, which are the knowledge of the exact gradient of the discrete objective function[2] and the easier generalization to higher-order derivatives.[11] Its main disadvantage, however, is that differentiating spatially and temporally discretized equations by hand quickly becomes impractical beyond trivial schemes.

High-order methods adapted to compressible/incompressible fluid flow computations on complex geometries have been recently developed, making them suitable candidates to become industrial tools in the near future.[16] One such code is JAGUAR,[1] developed jointly between CERFACS and ONERA. Its excellent scalability, its ability to handle structured or unstructured grids, its optimized 6-step time integration scheme as well as its high-order spatial discretization based on spectral differences[7,8] are all positive features which inevitably come at a price: the code is long and complex. Its differentiation for sensitivity computations is impractical. This is why we resorted to algorithmic differentiation software in order to automate the process.

Test case

Two-dimensional double shear layer in a periodic square

We consider a two-dimensional incompressible flow in a square periodic domain spanning $L = 1$ in the streamwise (x) and vertical (y) directions. The velocity field at the initial instant t_0 is given by

$$u = U \tanh\left[r\left(y - 1/4\right)\right], \quad y \leq 1/2 \tag{1}$$

$$u = U \tanh\left[r\left(3/4 - y\right)\right], \quad y > 1/2 \tag{2}$$

$$v = U\delta \sin\left[2\pi\left(x + 1/4\right)\right], \tag{3}$$

DOI: 10.13009/EUCASS2019-831

case name	*'r40'*	*'r80'*	*'r160'*
r	40	80	160
U	1	0.7072	0.5001

Table 1: The 3 shear parameters r considered in this study and their corresponding perturbation amplitudes U found from Equation (7). In all cases, the initial enstrophy is $\Omega_{ref} = 53.36$.

where all quantities are made non-dimensional with L and the streamwise reference velocity $U_0 = 1$ as follows:

$$t = \tilde{t}\, U_0/L, \quad y = \tilde{y}/L, \quad x = \tilde{x}/L, \quad U = \tilde{U}/U_0, \quad r = \tilde{r}\, L. \tag{4}$$

The parameters of the problem are U, r and δ. These are the streamwise velocity amplitude, the shear parameter and the ratio of vertical to streamwise velocity amplitudes, respectively. We set $\delta = 0.05$ for the remainder of this study so that it is no longer a free parameter. We analyze the evolution of the overall enstrophy Ω, defined as

$$\Omega = \int_0^1 \int_0^1 \frac{1}{2}\omega_z^2 \, dx \, dy, \tag{5}$$

where $\omega_z = \partial_x v - \partial_y u$ is the vorticity. It can be readily shown from Equations (1)-(3) and (5) that at $t = t_0$,

$$\Omega = \frac{U^2}{3}\left[6r\tanh(r/4) - 2r\tanh^3(r/4) + 3\delta^2\pi^2\right], \tag{6}$$

and we choose $r = 40$ with $U = 1$ to yield the initial enstrophy level $\Omega_{ref} = 53.36$ for all our considered test cases. This implies $U = U(r)$, which we can compute by re-arranging Equation (6) as follows:

$$U(r) = \left[3\Omega_{ref} \,/\, \left(6r\tanh(r/4) - 2r\tanh^3(r/4) + 3\delta^2\pi^2\right)\right]^{1/2} \tag{7}$$

The initial Reynolds number $Re_0 = U_0 L/\nu = 1.176 \times 10^4$ was identical for all our test cases outlined on Table 1. We show snapshots of ω_z for the case $r160$ at four different instants on Figure 1.

Simulations

The Navier-Stokes equations are solved for the flow described in the previous section using JAGUAR, with the Mach number set to zero and using 360^2 degrees of freedom based on a regular square grid. The flux point locations followed Legendre collocation in transformed space, the CFL was kept constant at 0.5 and the Riemann solver was based on the Roe scheme. We compared the output from JAGUAR against a fully spectral in-house code for periodic incompressible flows (MatSPE), which allowed us to do grid resolution studies and to validate our simulations. We compare $\Omega(t)$ between the fully spectral code and JAGUAR on Figure 2 (*left*). The agreement between the 2 codes is extremely good. We note that the JAGUAR computations were carried out with the serial and parallel versions of the code. The latter is based on message passing interface (MPI) instructions and ran on 4 processors.

Algorithmic differentiation

Sometimes referred to as automatic differentiation (AD), it is a tool that takes computer code as input. Some AD tools rely on operator overloading, others on source code transformation. In the latter case - the only one that will concern us, the output of the AD tool is a new code in the same programming language as the original code. If the original code computed a set of dependent variables Y_i based on independent variables X_j, then the new code will compute those derivatives dY_i/dX_j selected by the user. Since JAGUAR is written in modern FORTRAN, we chose an AD tool compatible with key features from recent FORTRAN standards and which is currently still supported by a team of developers. This tool is called TAPENADE,[3] developed by INRIA. It can differentiate code in one of two modes: tangent or adjoint. The tangent mode is most suitable when one seeks derivatives of many Y_i with respect to few X_j, while the adjoint mode is mandatory whenever few Y_i are to be differentiated with respect to a large number of variables X_j. The transformations inflicted by the AD tool on the original code will be radically different between the two differentiation modes, with the adjoint mode producing a new code which is the most challenging to both generate and recognise.

While AD tools can differentiate serial code on their own and are considered a fairly mature technology for this task, differentiating code with MPI directives in adjoint mode is still challenging. Few examples are available in

DOI: 10.13009/EUCASS2019-831

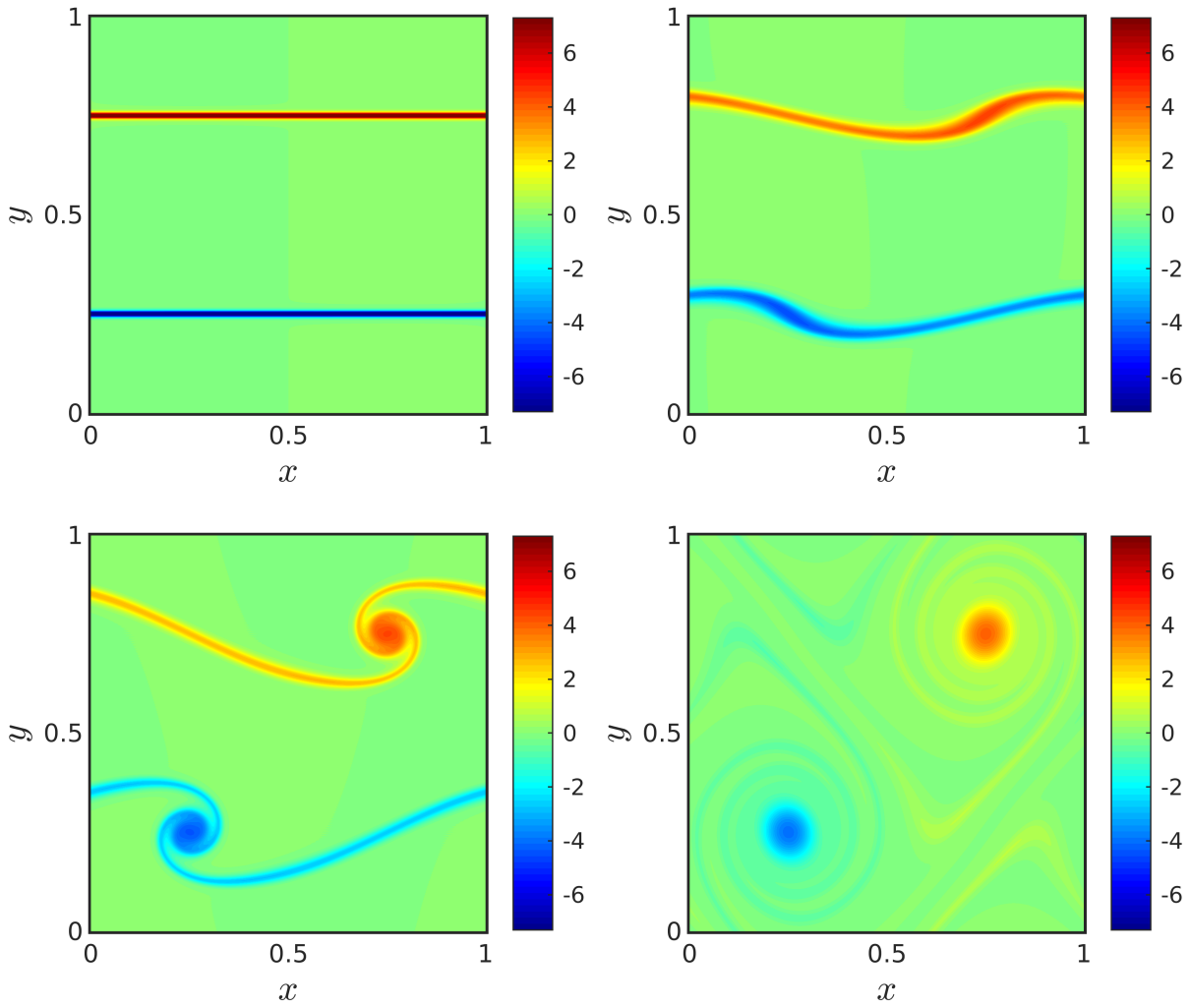

Figure 1: Spatial distribution of $\omega_z \left(\Omega_{ref}\right)^{-1/2}$ for case $r160$ at 4 instants $t \left(\Omega_{ref}\right)^{1/2} = \{0, 7, 10, 23\}$ in the following respective order: top left, top right, bottom left and bottom right.

the literature where AD tools had to analyze code lines containing MPI directives and generate their adjoint.[9,14] We emphasize the difference between the differentiation of code below the parallelisation layer[4,5] and differentiating code with MPI instructions. At present, the least *experimental* route is to re-write the MPI calls using the adjoinable MPI (AMPI) library[12] before differentiation with TAPENADE. At the time of writing, some MPI functions used in JAGUAR are not supported by the AMPI library, and hence the structure of the MPI communications has to be modified in the parallel version of JAGUAR in order to include exclusively AMPI-supported calls. Only then can the new code be migrated to AMPI and later differentiated by TAPENADE. It is worth underlining that the additional work necessary to obtain a parallel adjoint code with respect to a serial adjoint code is considerable. However, JAGUAR is designed to perform well on parallel supercomputers which are mandatory to tackle industrial problems, so that there would be little use of an adjoint serial code other than solving toy problems.

Sensitivities

We define the following two quantities

$$J_1 = \Omega(t) \tag{8}$$

$$J_2 = \int_0^T \Omega(t)\, dt \tag{9}$$

DOI: 10.13009/EUCASS2019-831

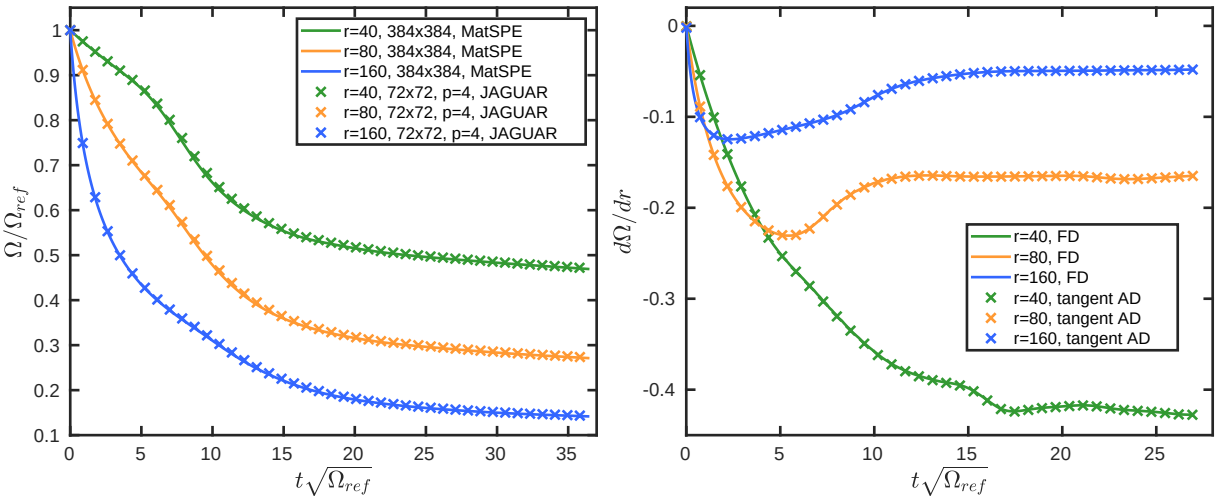

Figure 2: (***left***) Evolution of $\Omega(t)$ at 3 different values of r, JAGUAR *vs.* fully spectral and incompressible code MatSPE. The legend includes the number of Fourier modes used in each direction for the MatSPE simulation (of which 1/3 are zero-padded for dealiasing), while the JAGUAR data includes parameter p which is the selected order of the spatial discretization of the spectral difference scheme. The grid used in the JAGUAR simulation was a 72×72 structured mesh which, together with the setting $p = 4$, yields 360 degrees of freedom (DoF) per spatial direction. So we are effectively comparing 256^2 DoF with MatSPE against 360^2 DoF with JAGUAR. (***right***) Sensitivities dJ_1/dr, computed with both finite difference (FD) and the tangent-differentiated codes.

which, when differentiated with respect to r, yield the two sensitivities we compute by means of algorithic differentiation. dJ_1/dr is a time-dependent sensitivity, while dJ_2/dr is not. This implies dJ_2/dr can be computed through AD in both tangent and adjoint modes, while dJ_1/dr can only be computed in tangent mode. Both sensitivities will be compared against the finite difference (FD) estimates, computed with 2 independent realisations at r and $r + dr$, where $dr/r = 10^{-5}$.

Results and discussion

We modify JAGUAR in order to wrap the part of the code which takes r as input and computes $J_1(t)$ as output. In this form, TAPENADE differentiates the wrapped portion of the code into one which computes both J_1 and dJ_1/dr (in tangent mode). We compare this output to the estimate of dJ_1/dr based on FDs in Figure 2 (*right*). The agreement is excellent. The serial differentiated code is 1.9 times slower than the (serial) non-differentiated code. Since the FD computation takes exactly twice the execution time of the non-differentiated code, it appears that the higher accuracy of sensitivity computations from tangent AD comes with the added benefit of a slightly faster computation.

Similarly, we wrap the part of the code which takes r as input and computes $J_2(t)$ as output. We use TAPENADE to differentiate this version of the code in tangent and adjoint modes. The temporal integration is carried out from $t = 0$ to the $n - th$ iteration of the time integration loop in JAGUAR, with $n = 10^4$ and $n = 2 \times 10^4$. We show the output of both code sensitivities used with the 2 different time integration bounds on Table 2, together with the corresponding estimate computed with FDs. The results for dJ_2/dr computed with tangent and adjoint derivation modes agree to within numerical round-off error, while the FD estimate agrees less with the previous two due to the lesser accuracy of FDs. The agreement between the serial differentiated codes and the parallel counterparts validates the MPI transformations to JAGUAR, the migration to AMPI and the differentiation by TAPENADE of the AMPI code. We cannot stress enough how different the serial and parallel codes are from one another once differentiated in adjoint mode. To illustrate this further, both serial and parallel adjoint codes need seed values to compute the target gradient. The latter is recovered in a new, automatically generated differential variable. While in the serial code the differential variable contains *all* of the target gradient, in the parallel code the target gradient is *split* amongst the different processes. The way in which it is split will depend on the type of reduction operation applied to compute it (a sum over processes in our case) and on the type of partitionning applied by the CFD code.[9] JAGUAR uses a zero-halo partitioning scheme, and in order to recover the gradient with the parallel adjoint code we need to sum the value of the gradient obtained by each process. The agreement between this sum and the result from the serial adjoint code is evident on Table 2.

DOI: 10.13009/EUCASS2019-831

	$T = 10^4$ steps	$T = 2 \times 10^4$ steps
F. Diffs.	-1.41688**730**E-03	-4.5418**736**E-03
Tangent	-1.416883548249**268**E-03	-4.54186815284**618**E-03
Adjoint	-1.416883548249**464**E-03	-4.54186815284**581**E-03
F. Diffs. MPI	-1.41688**737**E-03	-4.5418**739**E-03
Tangent AMPI	-1.416883548249**40**E-03	-4.54186815284**779**E-03
Adjoint AMPI	-1.416883548247**57**E-03	-4.54186815284**222**E-03

Table 2: Estimates of dJ_2/dr according to 3 different computation methods, for serial code on one processor and parallel (MPI or AMPI) codes with 4 processors. Only computed for the case with $r = 160$, and for 2 integration intervals.

	$T = 10^4$ steps	$T = 2 \cdot 10^4$ steps
F. Diffs.	1.8 $hrs(\times 2)$	3.8 $hrs(\times 2)$
Tangent	3.1 hrs	6.2 hrs
Adjoint	58.5 hrs	123.2 hrs
F. Diffs. MPI	0.48 $hrs(\times 2)$	1.08 $hrs(\times 2)$
Tangent AMPI	0.81 hrs	1.62 hrs
Adjoint AMPI	9.53 hrs	18.1 hrs

Table 3: Same as Table 2 but showing computation times.

Table 3 shows the time taken by the computations. Both the tangent and the adjoint codes output by TAPENADE are free from further optimization. It is clear that the adjoint mode is very greedy on time. Initially we ran out of memory since in adjoint mode, a forward code execution is first carried out where snapshots of the data buffers are stored at every iteration in the time integration loop. In order to remedy this problem, we use an option of TAPENADE (binomial checkpointing) to specify the number of snapshots one is willing to store during a specific loop, at the expense of additional recomputation time.[15] Eventually, we chose to store 20 snapshots in the time stepping loop. This compromise reflects the challenge of using the adjoint mode for unsteady simulations as we do. We underline that most adjoint CFD computations - for instance using commercial software[10] - are done in stationary situations through iterative solution procedures with a set number of iterations (of the order of a few hundred). In these cases, one can do away in adjoint mode by storing only the final fixed-point solution - see the F.A.Q. section in TAPENADE's website. This is in stark contrast with our approach, where nonstationarity is intrinsic to our problem and the backwards time integration must be carried out based on storing and/or recomputing the intermediate results. Developing strategies to overcome this *computational barrier* is the object of current research.[6] We report an execution time in adjoint mode which is longer than in tangent mode by a factor of 19-20 for the serial codes and 11-12 for the parallel codes. The difference is due to the fact that the parallel code was stripped of many code lines which were not absolutely necessary to the present computation. In the serial code that was handed to TAPENADE, the source files and subroutine calls that allow JAGUAR to tackle various other problems (3D, compressible, different Riemann solvers, boundary condition types, *etc*...) were kept, inducing more differentiated code which did not modify the final answer yet required additional storage/recoveries from the backup buffers.

We emphasize that in adjoint mode the computation time would remain approximately constant as we increase the number of variables J_2 is differentiated with respect to. In tangent mode, it would grow linearly with the number of additional variables J_2 is differentiated with respect to. Hence the adjoint mode could still be preferable in those cases where the sensitivity of J_2 is needed with respect to a large number of parameters.

Conclusions and future work

The maturity of TAPENADE to cope with a modern FORTRAN code for computational fluid dynamics has allowed us to differentiate modified code in order to obtain sensitivity estimates in a simple test case. Many lessons have been learnt on the way the source code should be modified for a correct differentiation by TAPENADE. The differentiation of the parallel code, in particular, requires considerable code modifications. Even though the differentiation process is automatic, it should not be regarded as a black-box procedure. The output code requires careful analysis and checking, by means of comparison with FD or by using tools supplied with TAPENADE.

The simplicity of the test case, from a fluid mechanics point of view, does not preclude more complicated

DOI: 10.13009/EUCASS2019-831

simulations from being amenable to the same type of code differentiation, since it is the complexity of the code rather than that of the flow which modulates the performance/ability of TAPENADE to generate correctly differentiated code. Our target is set on the implementation of an optimal control loop in which the output sensitivities are iteratively used to reach a given goal.

Acknowledgments

This work has been financed by a grant from the STAE Foundation for the 3C2T project, managed by the IRT Saint-Exupery. J.I.C. acknowledges funding from the People Programme (Marie Curie Actions) of the European Union's Seventh Framework Programme (FP7/2007-2013) under REA grant agreement n. PCOFUND-GA-2013-609102, through the PRESTIGE programme coordinated by Campus France. We are very grateful for the help provided by Laurent Hascoët and Valérie Pascual on how to use TAPENADE.

References

[1] A. Cassagne, J.F. Boussuge, N. Villedieu, G. Puigt, I. D'ast, and A. Genot. Jaguar: a new cfd code dedicated to massively parallel high-order les computations on complex geometry. In *The 50th 3AF International Conference on Applied Aerodynamics (AERO 2015)*, 2015.

[2] M. Giles and N. Pierce. An introduction to the adjoint approach to design. *Flow, turbulence and combustion*, 65(3-4):393–415, 2000.

[3] L. Hascoet and V. Pascual. The tapenade automatic differentiation tool: Principles, model, and specification. *ACM Transactions on Mathematical Software (TOMS)*, 39(3):20, 2013.

[4] P. Heimbach, C. Hill, and R. Giering. Automatic generation of efficient adjoint code for a parallel navier-stokes solver. In *Computational Science — ICCS 2002*, pages 1019–1028, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[5] P. Hovland and C. Bischof. Automatic differentiation for message-passing parallel programs. In *Proceedings of the First Merged International Parallel Processing Symposium and Symposium on Parallel and Distributed Processing*, pages 98–104, March 1998.

[6] J.C. Hückelheim and J.D. Müller. Checkpointing with time gaps for unsteady adjoint cfd. In *Advances in Evolutionary and Deterministic Methods for Design, Optimization and Control in Engineering and Sciences*, pages 117–130. Springer, 2019.

[7] D.A. Kopriva. A staggered-grid multidomain spectral method for the compressible navier–stokes equations. *Journal of computational physics*, 143(1):125–158, 1998.

[8] Yen Liu, Marcel Vinokur, and Zhi Jian Wang. Spectral difference method for unstructured grids i: basic formulation. *Journal of Computational Physics*, 216(2):780–801, 2006.

[9] P. Mohanamuraly, L. Hascoët, and J.D. Müller. Seeding and adjoining zero-halo partitioned parallel scientific codes. *Optimization Methods and Software*, pages 1–20, 2019.

[10] J. Munoz-Paniagua, J. García, A. Crespo, and F. Laspougeas. Aerodynamic optimization of the nose shape of a train using the adjoint method. *Journal of Applied Fluid Mechanics*, 8(3):601–612, 2015.

[11] J.E. Peter and R.P. Dwight. Numerical sensitivity analysis for aerodynamic optimization: A survey of approaches. *Computers & Fluids*, 39(3):373–391, 2010.

[12] M. Schanen, U. Naumann, L. Hascoët, and J. Utke. Interpretative adjoints for numerical simulation codes using mpi. *Procedia Computer Science*, 1(1):1825–1833, 2010.

[13] B. Spagnoli and C. Airiau. Adjoint analysis for noise control in a two-dimensional compressible mixing layer. *Computers & Fluids*, 37(4):475–486, 2008.

[14] J. Utke, L. Hascoet, P. Heimbach, C. Hill, P. Hovland, and U. Naumann. Toward adjoinable mpi. In *2009 IEEE International Symposium on Parallel Distributed Processing*, pages 1–8, May 2009.

DOI: 10.13009/EUCASS2019-831

[15] A. Walther and A. Griewank. Advantages of binomial checkpointing for memory-reduced adjoint calculations. In *Numerical mathematics and advanced applications*, pages 834–843. Springer, 2004.

[16] F.D. Witherden and A. Jameson. *Future Directions in Computational Fluid Dynamics*, page 3791. 2017.