Automatic Differentiation of FLOWer and MUGRIDO

Ralf Giering[1], Thomas Kaminski[1], Bernhard Eisfeld[2], Nicolas Gauger[2], Jochen Raddatz[2], and Lars Reimer[3]

[1] FastOpt, Hamburg (`http://FastOpt.com`)
[2] DLR, Inst. for Aerodynamics and Flow Technology, Braunschweig (`http://DLR.de`)
[3] Mechanics Department (LFM), RWTH Aachen (`http://www.lufmech.rwth-aachen.de`)

1 Introduction

This chapter addresses the efficient computation of accurate sensitivity information in the aerodynamic design process. Mathematically, this sensitivity information is expressed by a derivative of a function that is defined via the numerical model of the aerodynamic system. This function links a number of independent variables to relevant target quantities such as lift, drag, or pitching moment. Figure 1 illustrates such a function definition via the process chain for a wing simulation. Independent parameters are the coordinates of the wing contours (see, e.g., [56]). In an automated design procedure, the space of wing coordinates can then be searched for a point that yields the optimum of the target quantity. The sensitivity of the target quantity with respect to the wing coordinates enables the use of powerful gradient algorithms for optimisation.

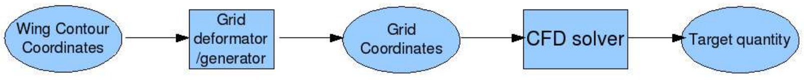

Fig. 1. Aerodynamic design chain. Oval boxes denote data and rectangular boxes numerical operations

For extracting such sensitivity information there are two major approaches. The first approach, often referred to as continuous approach, applies perturbation theory [38] to the equations underlying the numerical model. This results in linearised equations, which are discretised and then numerically integrated by the so-called tangent code. The computational cost of running the tangent code is roughly proportional to the number of independent variables. For most problems this number is much larger than that of the target quantities. For computationally demanding numerical models, sensitivity computation is then only feasible by means of adjoint code. In the continuous approach, adjoint code numerically integrates the discretisation of the adjoint version of the linearised model equations [39]. Lacking adjoint code, the inverse problem can only be tackled by reducing the complexity of the numerical model or the number of independent variables. The reduction of the design space is typically achieved via a parametrisation of the contour with only a few parameters. This would extend the chain of figure 1 by prepending the parametrisation, and the parameters would take the role of the independent variables and span the reduced design space. This regularisation of the inverse problem will typically yield a suboptimal solution, as the solution is to some degree prescribed by the parametrisation [55, 37].

The alternative route to sensitivity information applies Automatic Differentiation (AD, [27]). directly to the code of the original model: To generate the derivative code (tangent or adjoint code), the model code is decomposed into elementary functions (such as $+, -, \sin(\cdot)$), which (more or less) correspond to the individual statements in the code. These elementary functions are differentiated; this derivative is called local Jacobian. According to the chain rule, the product of the local Jacobians yields the derivative of the composite function. As opposed to derivative approximation by finite differences (also known as numerical differentiation), AD provides sensitivity information that is accurate within round-off error.

Like the continuous approach, AD can construct both tangent and adjoint codes. The tangent code uses the order in which the original model code evaluates the statements to evaluate the product of their local Jacobians. The adjoint code performs this evaluation in reverse order. In AD terminology, the tangent code operates in forward mode, and the adjoint code operates in reverse mode. Similar to the finite difference approximation, the computational resources needed in forward mode increase with the number of independent variables. In reverse mode, they are roughly proportional to the number of target quantities, but virtually independent of the number of independent variables.

The continuous approach involves the choice of a discretisation scheme for the adjoint equations. Typically it is not trivial to identify the scheme that yields, on the discretised level, the adjoint of the tangent. Any other scheme risks to produce sensitivity information that is inconsistent with the original code. This is of particular concern [50], when the adjoint sensitivity is used by an optimisation algorithm together with the target quantities provided by the original code.

AD can be carried out by an AD tool (for an overview see http://www.autodiff.org), but it is also common to apply the basic principles of derivative code construction [52, 19] by hand (see, e.g., [61, 58, 26]). The present paper describes AD of two Fortran codes that cover the design chain in figure 1: the DLR's RANS solver FLOWer [44], including a number of turbulence models, and the RWTH's flow grid deformator MUGRIDO [7, 33, 45]. For a number of CFD codes in Fortran, tangent (e.g. [9, 35, 16, 8, 5]) adjoint (e.g. [43]) and Hessian (e.g. [53]) codes have been generated by the AD tool ADIFOR [6], and adjoint codes (e.g. [41, 31]) by the AD tool Odyssée [46] and its successor TAPENADE [30]. Cusdin and Müller [13] compare the performance of tangent and adjoint versions simple CFD codes that can be handled by three AD tools.

For the present application we use the commercial AD tool Transformation of Algorithms in Fortran (TAF [19]). TAF and its predecessor TAMC generated tangent, adjoint, and Hessian codes, for a long list of applications (over 150 papers), primarily for large codes from Earth Sciences (up to 300,000 lines of Fortran excluding comments). For the feasibility of most of these applications, flexibility and computational efficiency of the derivative code are crucial and were, thus, in the focus of TAF development. Applications to CFD codes started this decade and include design of aircraft [54], turbo machinery [29], or cabin ventilation [42] as well as aeroacoustics [12]. Industrial CFD applications encompass design of race cars and aircraft. Generation of efficient second derivative code [20] for an airfoil configuration has been demonstrated by [24].

Use of an AD tool such as TAF is also favourable regarding the maintenance of the derivative code for models that are under development. Once the model code is TAF-compliant, the maintenance of the derivative code can be automated: At least after small changes of the model, the corresponding adjoint, tangent, and Hessian code can generally be updated automatically. TAF-compliance means that the derivative code is both correct and efficient as generated by TAF without any user intervention after the generation process. The effort of achieving this compliance typically pays off rapidly via the automated derivative code maintenance. For examples of this new concept that uses TAF as integrated component of the modelling system see [1, 28, 36, 42].

The remainder of this chapter is arranged as follows. Section 2 introduces the AD tool TAF and is followed by the descriptions of the AD process for FLOWer in section 3 and for MUGRIDO in section 4. Finally, section 5 draws conclusions.

2 TAF

Transformation of Algorithms in Fortran (TAF, [19]) is an AD tool for programmes written in Fortran 77-95. TAF operates as a source-to-source transformation tool. That is, from a given Fortran programme that evaluates a function, TAF generates a second Fortran programme that evaluates the function's derivative (gradient or Jacobian). TAF generates both forward and reverse mode derivative codes, i.e. tangent and adjoint models. In each mode, TAF can generate code to evaluate Jacobian times vector products or the full Jacobian. Second order derivative (Hessian) code is generated by invoking TAF twice. Typically, the most efficient strategy of obtaining second derivative information for a scalar-valued target quantity is the so-called forward over reverse mode of AD: TAF is invoked to generate the adjoint code, which afterwards is resubmitted to TAF to be differentiated in forward mode [20]. TAF is accessed via a secure shell connection to the FastOpt servers.

Another TAF feature [23] is Automatic Sparsity Detection (ASD), i.e. efficient determination of the sparsity structure of the Jacobian. This sparsity information can be important, because the Jacobian's sparsity pattern can be exploited to render the evaluation of the Jacobian more efficient (see, e.g., [36] for an application). In a CFD context ASD is of particular interest for evaluation of the sparse Jacobian representing the linearisation of a single solver iteration [14].

Recent TAF enhancements include basic support of parallel programming, namely the Message Passing Interface (MPI) and OpenMP (see [25, 32] for large-scale applications) as well as a mode for generation of a divided adjoint, which allows interruption and restart of the adjoint model run (see [32] for details).

TAF performs an analysis of the data flow in the code to be differentiated, which determines the active/passive variables. Active variables [4, 19] are all variables that depend on the independent variables and have influence on the target quantities. All non-active variables are called passive variables. Derivative information needs only be propagated and stored for active variables.

Required variables are all variables whose values are needed to evaluate the local Jacobian. For example, in integrations of non-linear systems the trajectory is part of the required variables. Their values can be provided by recomputation or by storing them on disk or in memory in an initial integration and reading them in the course of the adjoint integration. Most efficient adjoint code uses a combination of both [19, 21]. By default TAF inserts recomputations; automatic generation of a store/read scheme is triggered by TAF store directives. TAF can also generate a so-called checkpointing scheme [27] for particularly efficient use of disk/memory at the cost of an additional model integration.

For converging iterations, Christianson [10, 11] suggests an efficient alternative adjoint (based on the implicit function theorem), which only uses the required values from the last iteration and, thus, compared to the general adjoint considerably reduces the resources required for storing/recomputing. TAF implements automatic generation of the Christianson scheme, triggered by a TAF loop directive [24]. This is another feature of high interest in aerodynamic simulations, as these often address steady problems.

In case there are pieces of source code missing (black box routines), e.g. library routines, the user can provide the relevant data flow information via TAF flow directives [22, 24]. TAF flow directives are also applied to include available derivative code into TAF-generated code, which is useful, e.g., in case of self-adjoint routines, as demonstrated by [34, 25].

3 Automatic Differentiation of FLOWer

FLOWer is a Reynolds-averaged Navier-Stokes (RANS) solver [44] developed and maintained by DLR and used by its scientific and industrial partners. Excluding comments, the FLOWer code comprises over 100,000 lines of Fortran 77 (see Tab. 1) and can be run in a large variety of configurations [2], with a suite of algebraic, one-, and two-equation turbulence models [15]. For an Euler configuration, an adjoint version derived via the continuous approach [17] was available to the project. The initial strategy was to couple TAF-generated adjoint turbulence code with the continuous adjoint of the FLOWer core. In the course of the project, however, it turned out to be favourable to apply TAF to the entire FLOWer code.

Table 1. Performance of FLOWer's derivative code

Component	# of code lines	memory	CPU	rel. accuracy
Primal	166,000	1	1	
TLM	268,000	≈ 2	≈ 3	$\approx 10^{-8}$
ADM steady	310,000	2–3	6–10	$\approx 10^{-5}$
ADM general	310,000	variable	<10	$\approx 10^{-8}$

When rendering the FLOWer code TAF-compliant, we met a number of challenges. One is the use of large super-arrays for an implementation of (pseudo) dynamic memory management, a typical feature of legacy codes designed in the pre-Fortran-90 period. Another challenge is the implementation of an error-exit procedure via goto statements in every routine, which considerably complicates the programme's control flow structure. For the adjoint, an efficient store/read scheme has been devised. From the TAF-compliant solver code, a tangent and two adjoint versions were generated. The tangent code is mainly an intermediate result and is used for verification of the two adjoints. The first adjoint version (general adjoint) uses a flexible checkpointing scheme (see section 2) that stores required values on disk and in memory. It provides the exact gradient for steady and unsteady computations. The second version of the adjoint (steady adjoint) assumes convergence of the solver to a steady flow (see section 2) and stores this flow in memory. As an example of TAF-generated code, the Appendix shows the adjoint of FLOWer's LEA k-ω routine [47].

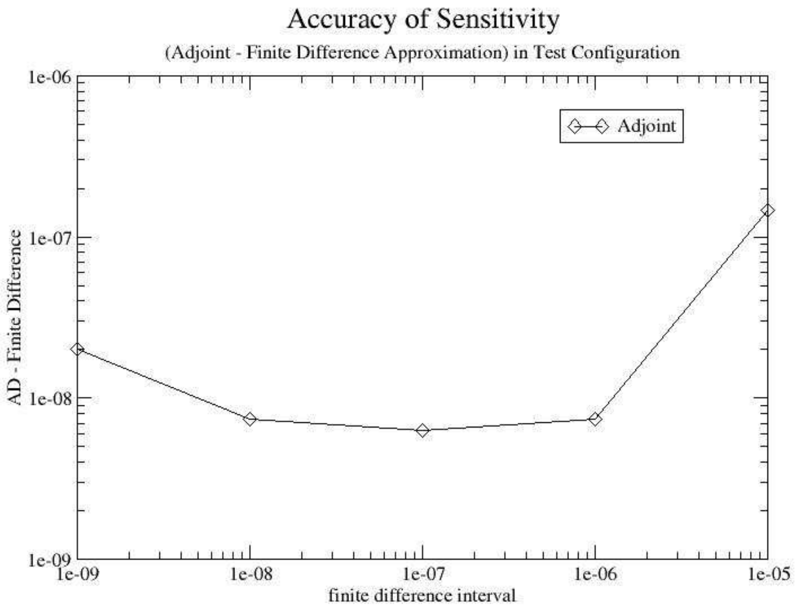

Fig. 2. Comparison of the derivative evaluated by the adjoint to finite difference approximations for a range of finite difference intervals.

The generated code has been verified for a 2d test configuration simulating the turbulent flow around a NACA 12 airfoil with 2000 iterations on a single fine grid. We evaluate the derivative of lift with respect to angle of attack. Figure 2 shows the relative difference of the general adjoint to the finite difference approximations for a range of finite difference intervals. Since we are running the evaluation in double precision with about 16 significant digits, a relative accuracy of the best finite difference approximation in the order of 10^{-8} is all we can expect. Lower accuracies usually indicate errors in the derivative code. The inaccuracy of the steady adjoint (see Tab. 1) is probably due to insufficient convergence of the primal integration. The relative difference between tangent and standard adjoint is in the order of 10^{-12}.

Tab. 1 lists the performances of the tangent and both adjoint versions for the test configuration with k-ω turbulence scheme [59, 60]. Owing to the flexibility of the checkpointing scheme (see section 2) the memory requirement for the general adjoint is variable. The CPU time is listed in multiples of primal solver integrations, and refers to the evaluation of the target function plus its derivative. For the adjoints this derivative refers to the full gradient, and for the tangent this refers to a directional derivative. CPU times vary with platform, compiler, and compiler options.

In addition to the Euler configuration, We have verified the derivative for the following five turbulence models:

- Baldwin and Lomax [3]
- Wilcox k-ω [59, 60]
- LLR k-ω [48]
- SST k-ω [40]
- LEA k-ω [47]

Curiously, for the one equation model of Spallart and Allmaras [51] the generated adjoint code produced a wrong gradient. We did not look into details but expect the problem is not too hard to identify and correct.

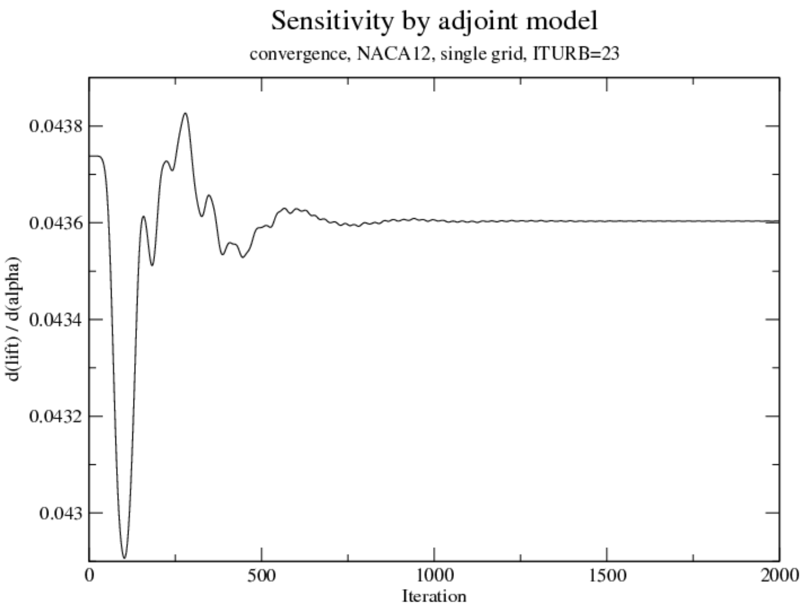

Fig. 3. Sensitivities of lift with respect to angle of attack computed by the adjoint of FLOWer with Wilcox [59, 60] turbulence formulation.

As an example of an adjoint sensitivity evaluation, figure 3 displays the sensitivity of lift with respect to angle of attack over the number of iterations in the adjoint solver. The computation uses the turbulence formulation according to Wilcox [59, 60].

4 Automatic Differentiation of MUGRIDO

Table 2. Performance of FLOWer's derivative code

Component	# of code lines	memory	CPU	rel. accuracy
Primal	25,000	1.0	1.0	
TLM scalar	42,800	2.1	1.0	$\approx 10^{-9}$
TLM 5 columns	43,400	3.4	1.2	$\approx 10^{-9}$
TLM 10 columns	43,400	5.8	1.5	$\approx 10^{-9}$
TLM 15 columns	43,400	8.3	2.2	$\approx 10^{-9}$

The Multiblock Grid Deformation Tool (MUGRIDO, [7, 33, 45]), was developed at RWTH Aachen. It can handle block-structured grid topologies, especially those used by FLOWer, and thus is well-suited for the design chain depicted in figure 1. MUGRIDO generates a fictitious beam framework by modelling block boundaries of the flow grid and a given percentage of additional grid lines as massless Timoshenko beams. After applying the Finite Element method the resulting linear system of equations is solved using the SPARSKIT utility for sparse matrices [49]. The right hand side to this system is supplied by deflections of the wetted surface relative to the undeformed grid. A well-shaped flow grid is finally reconstructed from the deformed beam framework by transfinite interpolation. MUGRIDO is written entirely in Fortran 77, and its coding concept is similar to FLOWer's, with the same super-array stucture for pseudo dynamic memory managment.

To demonstrate the applicability of TAF to a grid deformation tool, we rendered MUGRIDO TAF-compliant and generated its tangent in a fully automated procedure. For more efficient use of the tangent in the design chain, in addition to the standard (scalar) tangent, we also provided a vector mode version of the tangent, which simultaneously evaluates multiple directional derivatives. Tab. 2 lists the performance for the scalar tangent and the vector tangent for different numbers of directional derivatives. The CPU time refers to evaluation of the function plus the derivative. Increasing the number of directional derivatives does only marginally increase the CPU time. The agreement with the best finite difference approximation (last column) is excellent.

5 Conclusions

We demonstrated the feasibility of adjoint code generation for the CFD code FLOWer including a number of advanced turbulence models. The adjoint has been generated in two forms, one for steady simulations and a general one, which is suitable, e.g., for time-dependent simulations. The generated code is ready for applications, without any posterior modifications. The TAF-compliant FLOWer version is an excellent basis for further development of FLOWer, minimising the effort for updating the adjoint. The generated adjoint is efficient both in terms of memory usage and CPU time.

We also generated tangent code of the grid deformation tool MUGRIDO. The tangent is available in two forms, a scalar version for evaluation of a single directional derivate and a vector version for evaluation of multiple directional derivatives. The derivative code is highly efficient.

Many of the TAF algorithms can be ported without or with little modification to other programming languages. TAC++ [57] is the equivalent to TAF for differentiation of codes written in C(++). For a routine in the simplified Euler version of the DLR's RANS solver TAU [18], the tool generates highly efficient adjoint code in a fully automated procedure [57].

Appendix: Adjoint Code Example

Below we show the adjoint of LEA k-ω model [47], as an example of adjoint code generated by TAF. The declaration block, comment and blank lines are removed, to save space. Adjoint variables are denoted by the suffix `_ad`. The adjoint subroutine takes the sensitivity of `fmuet` (held in `fmuet_ad`) with respect to the target variable (e.g. lift) as input and propagates it back to the sensitivities of `r` (held in `r_ad`) and `shearvar`) (held in `shearvar_ad` with respect to the target variable. Note the recomputations before the nested loop and at the beginning of its kernel. For details in the generated code consult [19].

```
      subroutine turb26_ad( r, r_ad, swshear, shearvar, shearvar_ad,
     $fmuet_ad )
... (declarations, comments removed)
      help_h = epsma*1.e+8
      if (help_h .lt. 9.9999999999999e-31) then
        tolepsma = 9.9999999999999e-31
      else
        tolepsma = help_h
      endif
      twothird = 2./3.
      fothird = 4./3.
      cmu = 0.09
      c3 = 1.25
      c4 = 0.45
      do k = k2, 2, -1
        do j = j2, 2, -1
          do i = i2, 2, -1
            rho = r(i,j,k,1)
            help_j = r(i,j,k,it1)/rho
            if (help_j .ge. 0) then
              help_i = help_j
            else
              help_i = -help_j
            endif
            ka = help_i+tolepsma
            help_l = r(i,j,k,it2)/rho
            if (help_l .ge. 0) then
              help_k = help_l
            else
              help_k = -help_l
            endif
            om = help_k+tolepsma
            s = shearvar(i,j,k,1)/cmu/om
            st = shearvar(i,j,k,2)/cmu/om
            help_m = 1.5*st**1.7/(17.1+1.875*st**1.7)
            if (0.4 .lt. help_m) then
              c2 = help_m
            else
              c2 = 0.4
            endif
            arg1 = sqrt(0.8*s*s+0.2*st*st)
            arg2 = st*st/4.6225
            if (arg2 .lt. 1000.) then
              fact1 = 1.+0.95*(1.-tanh(arg2))
            else
              fact1 = 1.+0.95
            endif
            gr = 1./(1.6*fact1+st*st/(4.+1.83*arg1))
            beta1 = (fothird-c2)*gr*0.5
            beta2 = (2.-c4)*gr*0.5
            beta3 = (2.-c3)*gr
            xi2 = 0.5*beta2*beta2*s*s
            eta2 = 0.125*beta3*beta3*st*st
            cmust = beta1/(1.-twothird*eta2+2.*xi2)
            if (0.12 .gt. cmust) then
              help_n = cmust
            else
              help_n = 0.12
            endif
            if (0.04 .lt. help_n) then
              cmust = help_n
            else
              cmust = 0.04
            endif
            cmust_ad = cmust_ad+fmuet_ad(i,j,k)*(rho*ka/om/cmu)
            ka_ad = ka_ad+fmuet_ad(i,j,k)*(rho*cmust/om/cmu)
            om_ad = om_ad-fmuet_ad(i,j,k)*(rho*ka*cmust/(om*om)/cmu)
            rho_ad = rho_ad+fmuet_ad(i,j,k)*(ka*cmust/om/cmu)
            fmuet_ad(i,j,k) = 0.
            if (0.04 .lt. help_n) then
              help_n_ad = help_n_ad+cmust_ad
              cmust_ad = 0.
            else
              cmust_ad = 0.
            endif
            cmust = beta1/(1.-twothird*eta2+2.*xi2)
            if (0.12 .gt. cmust) then
              cmust_ad = cmust_ad+help_n_ad
              help_n_ad = 0.
            else
              help_n_ad = 0.
            endif
            beta1_ad = beta1_ad+cmust_ad/(1.-twothird*eta2+2.*xi2)
            eta2_ad = eta2_ad+cmust_ad*(beta1*twothird/((1.-twothird*
     $eta2+2.*xi2)*(1.-twothird*eta2+2.*xi2)))
            xi2_ad = xi2_ad-cmust_ad*(2*beta1/((1.-twothird*eta2+2.*xi2)
     $*(1.-twothird*eta2+2.*xi2)))
            cmust_ad = 0.
            beta3_ad = beta3_ad+0.25*eta2_ad*beta3*st*st
            st_ad = st_ad+0.25*eta2_ad*beta3*beta3*st
            eta2_ad = 0.
            beta2_ad = beta2_ad+xi2_ad*beta2*s*s
            s_ad = s_ad+xi2_ad*beta2*beta2*s
            xi2_ad = 0.
            gr_ad = gr_ad+beta3_ad*(2.-c3)
            beta3_ad = 0.
            gr_ad = gr_ad+0.5*beta2_ad*(2.-c4)
            beta2_ad = 0.
            c2_ad = c2_ad-0.5*beta1_ad*gr
            gr_ad = gr_ad+0.5*beta1_ad*(fothird-c2)
            beta1_ad = 0.
            arg1_ad = arg1_ad+gr_ad*(1.*(1.83*st*st/((4.+1.83*arg1)*(4.+
     $1.83*arg1)))/((1.6*fact1+st*st/(4.+1.83*arg1))*(1.6*fact1+st*st/(
     $4.+1.83*arg1))))
            fact1_ad = fact1_ad-gr_ad*(1.6/((1.6*fact1+st*st/(4.+1.83*
     $arg1))*(1.6*fact1+st*st/(4.+1.83*arg1))))
            st_ad = st_ad-gr_ad*(1.*(2*st/(4.+1.83*arg1))/((1.6*fact1+
     $st*st/(4.+1.83*arg1))*(1.6*fact1+st*st/(4.+1.83*arg1))))
            gr_ad = 0.
            if (arg2 .lt. 1000.) then
              arg2_ad = arg2_ad-0.95*fact1_ad*(1./cosh(arg2)**2)
              fact1_ad = 0.
            else
              fact1_ad = 0.
            endif
            st_ad = st_ad+arg2_ad*(2*st/4.6225)
            arg2_ad = 0.
            s_ad = s_ad+1.6*arg1_ad*1./(2.*sqrt(0.8*s*s+0.2*st*st))*s
            st_ad = st_ad+0.4*arg1_ad*1./(2.*sqrt(0.8*s*s+0.2*st*st))*st
            arg1_ad = 0.
            if (0.4 .lt. help_m) then
              help_m_ad = help_m_ad+c2_ad
              c2_ad = 0.
            else
              c2_ad = 0.
            endif
            st_ad = st_ad+help_m_ad*(2.55*st**0.7/(17.1+1.875*st**1.7)-
     $3.1875*1.5*st**1.7*st**0.7/((17.1+1.875*st**1.7)*(17.1+1.875*st**
     $1.7)))
            help_m_ad = 0.
            om_ad = om_ad-st_ad*(shearvar(i,j,k,2)/cmu/(om*om))
            shearvar_ad(i,j,k,2) = shearvar_ad(i,j,k,2)+st_ad*(1/cmu/om)
            st_ad = 0.
            om_ad = om_ad-s_ad*(shearvar(i,j,k,1)/cmu/(om*om))
            shearvar_ad(i,j,k,1) = shearvar_ad(i,j,k,1)+s_ad*(1/cmu/om)
            s_ad = 0.
            help_k_ad = help_k_ad+om_ad
            om_ad = 0.
            if (help_l .ge. 0) then
              help_l_ad = help_l_ad+help_k_ad
              help_k_ad = 0.
            else
              help_l_ad = help_l_ad-help_k_ad
              help_k_ad = 0.
            endif
            r_ad(i,j,k,it2) = r_ad(i,j,k,it2)+help_l_ad/rho
            rho_ad = rho_ad-help_l_ad*(r(i,j,k,it2)/(rho*rho))
            help_l_ad = 0.
            help_i_ad = help_i_ad+ka_ad
            ka_ad = 0.
            if (help_j .ge. 0) then
              help_j_ad = help_j_ad+help_i_ad
              help_i_ad = 0.
            else
              help_j_ad = help_j_ad-help_i_ad
              help_i_ad = 0.
            endif
            r_ad(i,j,k,it1) = r_ad(i,j,k,it1)+help_j_ad/rho
            rho_ad = rho_ad-help_j_ad*(r(i,j,k,it1)/(rho*rho))
            help_j_ad = 0.
            r_ad(i,j,k,1) = r_ad(i,j,k,1)+rho_ad
            rho_ad = 0.
          end do
```

```
  end do
end do
end subroutine turb26_ad
```

References

1. Adcroft, A., Campin, J.M., Heimbach, P., Hill, C., Marshall, J.: The MITgcm. Online documentation, Massachusetts Institute of Technology, USA (2002)
2. Aumann, P., Bartelheimer, W., Bleecke, H., Eisfeld, J., Lieser, J., Heinrich, R., Kroll, N., Kuntz, M., Monsen, E., Raddatz, J., Reisch, U., Roll, B.: FLOWer Installation and USER Handbook Release 116. Tech. Rep. MEGAFLOW-1001, DLR (2000)
3. Baldwin, B., Lomax, H.: Thin-layer approximation and algebraic model for separated turbulent flows. IAAA Paper 1978-0257, AIAA, Reston Va, USA (1978)
4. Bischof, C., Carle, A., Khademi, P., Mauer, A.: ADIFOR 2.0: Automatic differentiation of Fortran 77 programs. IEEE Computational Science & Engineering **3**(3), 18–32 (1996)
5. Bischof, C.H., Bücker, H.M., Lang, B., Rasch, A., Slusanschi, E.: Efficient and accurate derivatives for a software process chain in airfoil shape optimization. Tech. Rep. RWTH-CS-SC-02-06, Institute for Scientific Computing, Aachen University of Technology, Aachen (2002)
6. Bischof, C.H., Carle, A., Corliss, G.F., Griewank, A., Hovland, P.D.: ADIFOR: Generating derivative codes from Fortran programs. Scientific Programming **1**, 11–29 (1992)
7. Boucke, A.: Kopplungswerkzeuge für aeroelastische simulationen. Ph.D. thesis, RWTH Aachen (2003)
8. Bücker, H.M., Lang, B., Rasch, A., Bischof, C.H.: Computation of sensitivity information for aircraft design by automatic differentiation. In: P.M.A. Sloot, C.J.K. Tan, J.J. Dongarra, A.G. Hoekstra (eds.) Computational Science – ICCS 2002, Proceedings of the International Conference on Computational Science, Amsterdam, The Netherlands, April 21–24, 2002. Part II, *Lecture Notes in Computer Science*, vol. 2330, pp. 1069–1076. Springer, Berlin (2002)
9. Carle, A., Green, L., Bischof, C.H., Newman, P.: Applications of automatic differentiation in CFD. In: Proceedings of the 25th AIAA Fluid Dynamics Conference, AIAA Paper 94-2197. American Institute of Aeronautics and Astronautics (1994)
10. Christianson, B.: Reverse accumulation and attractive fixed points. Optimization Methods and Software **3**, 311–326 (1994)
11. Christianson, B.: Reverse accumulation and implicit functions. Optimization Methods and Software **9**(4), 307–322 (1998)
12. Collis, S.S., Ghayour, K., Heinkenschloss, M., Ulbrich, M., Ulbrich, S.: Towards Adjoint-Based Methods for Aeroacoustic Control. IAAA Paper 2001-0821, AIAA, Reston Va, USA (2001)
13. Cusdin, P., Müller, J.D.: Improving the performance of code generated by automatic differentiation. Tech. Rep. QUB-SAE-03-04, QUB School of Aeronautical Engineering (2003)
14. Dwight et al., R.: Development of Adjoint Methods for Hybrid RANS Solver TAU. In: this issue. Springer (2008)

15. Eisfeld, B.: Turbulence Models in FLOWer. In: J.K. Kroll Norbert; Fassbender (ed.) MEGAFLOW- Numerical Flow Simulation for Aircraft Design, *Notes on Numerical Fluid Mechanics and Multidisciplinary Design*, vol. 89, pp. 63–77. Springer Verlag (2005)
16. Forth, S.A., Evans, T.P.: Aerofoil Optimisation via AD of a Multigrid Cell-Vertex Euler Flow Solver. In: G. Corliss, C. Faure, A. Griewank, L. Hascoët, U. Naumann (eds.) Automatic Differentiation: From Simulation to Optimization, Computer and Information Science, chap. 17, pp. 153–160. Springer, New York (2001)
17. Gauger, N.: Das Adjungiertenverfahren in der aerodynamischen Formoptimierung. Ph.D. thesis, TU Braunschweig (2004)
18. Gerhold, T.: Overview of the hybrid rans code tau. In: J.K. Kroll Norbert; Fassbender (ed.) MEGAFLOW- Numerical Flow Simulation for Aircraft Design, *Notes on Numerical Fluid Mechanics and Multidisciplinary Design*, vol. 89, pp. 81 – 92. Springer Verlag (2005)
19. Giering, R., Kaminski, T.: Recipes for Adjoint Code Construction. ACM Trans. Math. Software **24**(4), 437–474 (1998)
20. Giering, R., Kaminski, T.: Using TAMC to generate efficient adjoint code: Comparison of automatically generated code for evaluation of first and second order derivatives to hand written code from the minpack-2 collection. In: C. Faure (ed.) Automatic Differentiation for Adjoint Code Generation, pp. 31–37. INRIA, Sophia Antipolis, France (1998)
21. Giering, R., Kaminski, T.: Recomputations in reverse mode AD. In: G. Corliss, A. Griewank, C. Fauré, L. Hascoet, U. Naumann (eds.) Automatic Differentiation of Algorithms: From Simulation to Optimization, chap. 33, pp. 283–291. Springer Verlag, Heidelberg (2002)
22. Giering, R., Kaminski, T.: Applying TAF to generate efficient derivative code of Fortran 77-95 programs. PAMM **2**(1), 54–57 (2003)
23. Giering, R., Kaminski, T.: Automatic sparsity detetection implemented as soruce-to-source transformation. In: V.N. Alexandrov, G.D. van Albada, P.M.A. Sloot, J. Dongarra (eds.) Computational Science – ICCS 2006, *Lecture Notes in Computer Science*, vol. 3394, pp. 591–598. Springer, Heidelberg (2006)
24. Giering, R., Kaminski, T., Slawig, T.: Generating Efficient Derivative Code with TAF: Adjoint and Tangent Linear Euler Flow Around an Airfoil. Future Generation Computer Systems **21**(8), 1345–1355 (2005)
25. Giering, R., Kaminski, T., Todling, R., Errico, R., Gelaro, R., Winslow, N.: Generating tangent linear and adjoint versions of NASA/GMAO's Fortran-90 global weather forecast model. In: H.M. Bücker, G. Corliss, P. Hovland, U. Naumann, B. Norris (eds.) Automatic Differentiation: Applications, Theory, and Tools, Lecture Notes in Computational Science and Engineering. Springer (2005)
26. Giles, M., Duta, M., Mueller, J., Pierce, N.: Algorithm developments for discrete adjoint methods. AIAA Journal **41**(2), 198–205 (2003)
27. Griewank, A.: Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation. Optimization Methods and Software **1**, 35–54 (1992)
28. Griffies, S.M., Harrison, M.J., Pacanowski, R.C., Rosati, A.: The FMS MOM4-beta User Guide. Tech. rep., NOAA/Geophysical Fluid Dynamics Laboratory (2002)
29. Hall, K.C., Thomas, J.P.: Sensitivity analysis of coupled aerodynamic/structural dynamic behavior of blade rows. Extended Abstract for the 7th National Turbine

Engine High Cycle Fatigue (HCF) Conference, Palm Beach Gardens, Florida, 14-17 May 2002 (2002)

30. Hascoët, L., Pascual, V.: TAPENADE 2.1 user's guide. Rapport technique 300, INRIA, Sophia Antipolis (2004)
31. Hascoët, L., Vázquez, M., Dervieux, A.: Automatic differentiation for optimum design, applied to sonic boom reduction. In: V. Kumar, M.L. Gavrilova, C.J.K. Tan, P. L'Ecuyer (eds.) Computational Science and Its Applications – ICCSA 2003, Proceedings of the International Conference on Computational Science and its Applications, Montreal, Canada, May 18–21, 2003. Part II, *Lecture Notes in Computer Science*, vol. 2668, pp. 85–94. Springer, Berlin (2003)
32. Heimbach, P., Hill, C., Giering, R.: An efficient exact adjoint of the parallel MIT general circulation model, generated via automatic differentiation. Future Generation Computer Systems **21**(8), 1356–1371 (2005)
33. Hesse, M.: Entwicklung eines automatischen gitterdeformationsalgorithmus zur stroemungsberechnung um komplexe konfiguration auf hexaeder-netzen. Ph.D. thesis, RWTH Aachen (2006)
34. Hinze, M., Slawig, T.: Adjoint gradients compared to gradients from algorithmic differentiation in instataneous control of the Navier-Stokes equations. Optimization Methods & Software **18**(3), 299–315 (2003)
35. Hovland, P.D., Mohammadi, B., Bischof, C.H.: Automatic differentiation of Navier-Stokes computations. Tech. Rep. MCS-P687-0997, Argonne National Laboratory (1997)
36. Kaminski, T., Giering, R., Scholze, M., Rayner, P., Knorr, W.: An example of an automatic differentiation-based modelling system. In: V. Kumar, L. Gavrilova, C.J.K. Tan, P. L'Ecuyer (eds.) Computational Science – ICCSA 2003, International Conference Montreal, Canada, May 2003, Proceedings, Part II, *Lecture Notes in Computer Science*, vol. 2668, pp. 95–104. Springer, Berlin (2003)
37. Kaminski, T., Heimann, M.: Inverse modeling of atmospheric carbon dioxide fluxes. Science **294**(5541), 259 (2001)
38. Kato, T.: Perturbation theory for linear operators. Springer, Berlin (1966)
39. Marchuk, G.I.: Adjoint Equations and Analysis of Complex Systems. Kluwer, Dordrecht (1995)
40. Menter, F.: Two-equation eddy-viscosity turbulence models for engineering applications. AIAA Journal **32**(8), 1598–1605 (1994)
41. Mohammadi, B., Malé, J.M., Rostaing-Schmidt, N.: Automatic differentiation in direct and reverse modes: Application to optimum shapes design in fluid mechanics. In: M. Berz, C.H. Bischof, G.F. Corliss, A. Griewank (eds.) Computational Differentiation: Techniques, Applications, and Tools, pp. 309–318. SIAM, Philadelphia, Penn. (1996)
42. Othmer, C., Kaminski, T., Giering, R.: Computation of topological sensitivities in fluid dynamics: Cost function versatility. In: P. Wesseling, E.O. nate, J. Périaux (eds.) ECCOMAS CFD 2006. TU Delft (2006)
43. Park, M.A., Green, L.L., Montgomery, R.C., Raney, D.L.: Determination of Stability and Control Derivatives Using Computational Fluid Dynamics and Automatic Differentiation. IAAA Paper 1999-3136, AIAA, Reston Va, USA (1999)
44. Raddatz, J., Fassbender, J.: Block Structured Navier-Stokes Solver FLOWer. In: J.K. Kroll Norbert; Fassbender (ed.) MEGAFLOW- Numerical Flow Simulation for Aircraft Design, *Notes on Numerical Fluid Mechanics and Multidisciplinary Design*, vol. 89, pp. 27–44. Springer Verlag (2005)

45. Reimer, L., Hesse, M.: Kurzdokumentation des Mehrblock-Gitterdeformationsverfahrens MUGRIDO. Tech. rep., RWTH Aachen (2006)
46. Rostaing, N., Dalmas, S., Galligo, A.: Automatic differentiation in Odyssée. Tellus **45A**, 558–568 (1993)
47. Rung, T., Luebcke, H., Franke, M., Xue, L., Thiele, F., Fu, S.: Assessment of explicit algebraic stress models in transonic flows. In: Proceedings of the 4th International Symposium on Engineering Turbulence Modelling and Measurements, Ajaccio, France; 24-26 May 1999, pp. 659–668 (1999)
48. Rung, T., Thiele, F.: Computational modelling of complex boundary-layer flows. In: Proceedings of the 9th Int. Symp. on Transport Phenomena in Thermal-Fluid Engineering, Singapore (1996)
49. Saad, Y.: Sparskit: A Basic Tool Kit for Sparse Matrix Computation (1994)
50. Shah, P.: Application of adjoint equations to estimation of parameters in distributed dynamic systems. In: A. Griewank, G.F. Corliss (eds.) Automatic Differentiation of Algorithms: Theory, Implementation, and Application, pp. 181–190. SIAM, Philadelphia, Penn. (1991)
51. Spallart, P., Allmaras, S.: A One-Equation Model for Aerodynamic Flows'. AIAA Journal **92**(439) (1992)
52. Talagrand, O.: The use of adjoint equations in numerical modelling of the atmospheric circulation. In: A. Griewank, G.F. Corliss (eds.) Automatic Differentiation of Algorithms: Theory, Implementation, and Application, pp. 169–180. SIAM, Philadelphia, Penn. (1991)
53. Taylor III, A.C., Green, L.L., Newman, P.A., Putko, M.M.: Some Advanced Concepts in Discrete Aerodynamic Sensitivity Analysis. IAAA Paper 2001-2529, AIAA, Reston Va, USA (2001)
54. Thomas, J.P., Hall, K.C., Dowell, E.H.: A discrete adjoint approach for modeling unsteady aerodynamic design sensitivities. AIAA Journal. **43**(9), 1931–1936 (2005)
55. Trampert, J., Snieder, R.: Model estimations biased by truncated expansions: Possible artifacts in seismic tomography. Science **271**, 1257–1260 (1996)
56. Ulbrich, S.: Optimal Control of Nonlinear Hyperbolic Conservation Laws with Source Terms , Habilitationsschrift. Fakultät für Mathematik, Technische Universität München, Germany (2002)
57. Voßbeck, M., Giering, R., Kaminski, T.: Development and First Applications of TAC++. In: C. Bischof, H.M. Bücker, P.D. Hovland, U. Naumann, J. Utke (eds.) to appear in Advances in Automatic Differentiation, Lecture Notes in Computational Science and Engineering. Springer, Berlin (2008)
58. Weaver, A., Vialard, J., Anderson, D.: Three-and Four-Dimensional Variational Assimilation with a General Circulation Model of the Tropical Pacific Ocean. Part I: Formulation, Internal Diagnostics, and Consistency Checks. Monthly Weather Review **131**(7), 1360–1378 (2003)
59. Wilcox, D.: Reassessment of the scale-determining equation for advanced turbulence models. AIAA, Aerospace Sciences Meeting **26**, 1299–1310 (1988)
60. Wilcox, D.: Turbulence Modeling for CFD, DCW Industries. Inc., La Canada, California (1993)
61. Zhu, J., Kamachi, M.: The Role of Time Step Size in Numerical Stability of Tangent Linear Models. Monthly Weather Review **128**(5), 1562–1572 (2000)