AD-based perturbation methods for uncertainties and errors

A. Belme, M. Martinelli, L. Hascoët, V. Pascual, Alain Dervieux

AD-based perturbation methods for uncertainties and errors

A. Belme, M. Martinelli, L. Hascoët, V. Pascual, A. Dervieux
Tropics Project, INRIA, 2004 Route des Lucioles,
06902 Sophia-Antipolis FR.
{Alain.Dervieux, Laurent.Hascoet}@sophia.inria.fr

March 19, 2009

Key words: Automatic Differentiation, second-order derivatives, uncertainty, error correction, design, CFD.[1]

Abstract. The progress of Automatic Differentiation (**AD**) and its impact on perturbation methods is the object of this paper. AD studies show an important activity for developing methods addressing the management of modern CFD kernels, taking into account the language evolution, and intensive parallel computing. The evaluation of a posteriori error analysis and of resulting correctors will be addressed. Recents works in the AD-based contruction of second-derivatives for building reduced-order models based on a Taylor formula will be presented on the test case of a steady compressible flow around an aircraft.

1 Introduction

While high fidelity models are mainly used for deterministic design, which assumes a perfect knowledge of the environmental and operational parameters, uncertainty can arise in many aspects of the entire design-production-operational process: from the assumptions done in the mathematical model describing the underlying physical process, to the manufacturing tolerances, and to the operational parameters and conditions that could be affected by unpredictable factors (e.g. atmospheric conditions). Exact and approximate techniques for propagating these uncertainties require additional computational effort but are progressively well established. The proposed study takes place in NODESIM-CFD FP6 project [10, 15]. The AD tool TAPENADE [13] has been developed for a large range of applications where the code to code direct and reverse differentiation is needed. Direct and reverse Automatic Differentiation are used for addressing numerical error reduction since they help building correctors. This discussed in Section 3. Uncertainty propagation is addressed by a perturbation technique using the first terms of Taylor series of the high-fidelity model (Method of Moments). This is built as an application of Automatic Differentiation (AD). Efficient first and second derivatives softwares are produced by TAPENADE, thanks to recent improvements that we shortly present in first section. Then the development of a Method of Moments using second derivatives obtained with TAPENADE is presented in Section 2.

[1] 44th AAAF, Nantes (F) 2009

2 Automated Differentiation improvements

Our AD tool TAPENADE has been extended to deal with Fortran95 and with ANSI C ([1,4]). Figure 1 shows the architecture of TAPENADE. It is implemented mostly in JAVA (115 000 lines) except for the separate front-ends which can be written in their own languages. Front- and

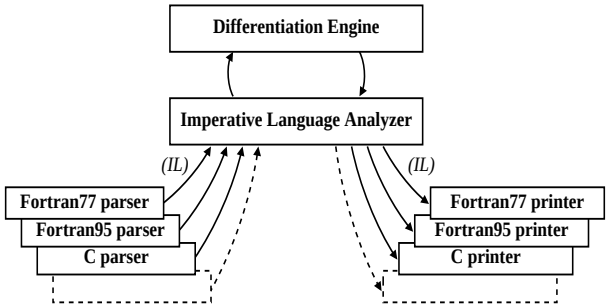

Figure 1: Overall Architecture of TAPENADE

back-ends communicate with the kernel via an intermediate abstract language ("IL") that makes the union of the constructs of individual imperative languages. Notice also the clear separation between the general-purpose program analysis and the differentiation engine itself.

Thanks to the language-independent internal representation of programs, this still makes a single and only tool, and every development benefits to differentiation of each input language. One of these developments concerned the pointer analysis. The reverse mode now accepts most uses of pointers and allocation. Another development concerned declarations. The differentiated program respects the order of declarations, uses the include files and keeps the comments from the original program. Generated codes are more readable and often smaller. We also investigated extensions to TAPENADE to successive differentiations, in particular to efficiently handle tangent differentiation of the stack primitives present in the reverse differentiated codes. We implemented user directives for the reverse differentiation of a frequent class of parallel loops (directive `II-LOOP`) and for opti-

mal checkpointing in reverse differentiation ([5],[6],[7]). TAPENADE lets the user specify finely which procedure calls must be checkpointed or not with the directive `NOCHECKPOINT`.

3 Numerical errors reduction

3.1 Error estimates and corectors

Let us recall first how linearised -direct or adjoint- states can be useful for improving numerical accuracy issues.

Numerical error involves the deviation between the solution $W = W(x, y, z)$ of mathematical model i.e. of the nonlinear PDE symbolized by:

$$\Psi(W) = 0 \ . \tag{1}$$

and the output data produced by the computations, i.e. the more or less perfect numerical solution of the discrete system:

$$\mathbf{\Psi}_h(\mathbf{W}_h) = \mathbf{0} \ \in \ \mathbb{R}^N \tag{2}$$

The discrete unknown $\mathbf{W}_h$ is the N-dimensional array of degrees of freedom:

$$\mathbf{W}_h \ \in \ \mathbb{R}^N \ , \ \mathbf{W}_h = [(\mathbf{W}_h)_i] \ .$$

The output data produced by the computation do not involve a function W_h but, instead the array $\mathbf{W}_h$ which needs to be transformed via an interpolation: Let $V \ \in \ L^2(\Omega)$ a space of rather smooth function. In practice, $V \ \subset \ \mathcal{C}^0(\bar{\Omega})$. Let R_h be a linear interpolation operator transforming an array of N degrees of freedom into a continuous function:

$$R_h : \mathbb{R}^N \ \rightarrow \ V \quad \mathbf{v}_h \ \mapsto \ R_h \mathbf{v}_h. \tag{3}$$

Let:

$$W_h(x, y, z) = (R_h \mathbf{W}_h)(x, y, z).$$

Similarly, we need an operator from continuous functions to arrays. Let T_h be an operator transforming a continuous function into an array of N degrees of freedom:

$$T_h : V \ \rightarrow \ \mathbb{R}^N \quad v \ \mapsto \ T_h v. \tag{4}$$

It is useful to take the adjoint of R_h:

$$T_h = R_h^*. \tag{5}$$

The deviation between the PDE solution and the numerical one can be defined as $W - W_h$. It consists mainly of approximation errors, of algorithmic errors, arising typically because iterative algorithmes are not iterated infinitely, and of round-off errors due to the fact that the programm is run in floating point arithmetics. We discuss here mainly of approximation errors, although the other ones may be also addressed in part by the method studied here.

Another way to postprocess a computation is to use it for evaluating a **scalar** functional:

Let j a smooth linear functional applying W into the scalar number:

$$j(u) = (g, W)_{L^2(\Omega)}$$

where g is a given $L^2(\Omega)$ function. This allows to define:

$$\begin{aligned} \mathbf{g}_h &= T_h g \\ g_h &= R_h \mathbf{g}_h = R_h T_h g \end{aligned} \tag{6}$$

The continuous adjoint writes:

$$(\frac{\partial \Psi}{\partial u})^* p = g \ .$$

The discrete adjoint equation is then defined by:

$$\left[\frac{\partial \mathbf{\Psi}_h}{\partial \mathbf{u}_h}\right]^T \mathbf{p}_h = T_h g \ . \tag{7}$$

And we can then consider:

$$p_h = R_h \mathbf{p}_h \ .$$

A fundamental assumption of the present analysis is that this discrete adjoint is a good enough approximation of continuous adjoint p for allowing to replace p by p_h the calculations which follow. In order to evaluate the approximation error, two kinds of estimates can be applied:

A posteriori estimate:

$$\Psi(W) - \Psi(W_h) = -\Psi(W_h) \tag{8}$$

where $\Psi(W_h)$ is the continuous residual applied to discrete solution. Then:

$$W - W_h \approx -\left[\frac{\partial \Psi}{\partial W}\right]^{-1} \Psi(W_h). \tag{9}$$

A priori estimate:

$$\mathbf{\Psi}_h(T_h W) - \mathbf{\Psi}_h(\mathbf{W}_h) = -\mathbf{\Psi}_h(T_h W) \tag{10}$$

where $\Psi_h(T_h W)$ is the discrete residual applied to discretised continuous solution.Then:

$$T_h W - \mathbf{W}_h \approx -\left[\frac{\partial \mathbf{\Psi}_h}{\partial \mathbf{W}_h}\right]^{-1} \Psi_h(T_h W). \tag{11}$$

We observe that these estimates involve unavailable continuous functions. In the *a posteriori* estimate, the solution of the continuous linearised system can be approximated thanks to the discrete Jacobian. For the *a priori* estimate, we can also solve this issue in some particular case see [14], by replacing $\mathbf{\Psi}_h(T_h W)$ by an expression $T_h \Theta_h(W_h)$ depending only of W_h. Corresponding to these estimates, we have the following **field correctors**:

$$\delta W_h = -R_h \left[\frac{\partial \mathbf{\Psi_h}}{\partial W_h}\right]^{-1} T_h \Psi(W_h) \tag{12}$$

$$\delta R_h \mathbf{W}_h = -R_h \left[\frac{\partial \mathbf{\Psi}_h}{\partial \mathbf{W}_h}\right]^{-1} T_h \Theta_h(W_h) \tag{13}$$

and the following **direct-linearized goal-oriented correctors**:

$$\delta_1 j = -\left(g \ , \ R_h \left[\frac{\partial \mathbf{\Psi_h}}{\partial W_h}\right]^{-1} T_h \Psi(W_h)\right)_{L^2(\Omega)} \tag{14}$$

$$\delta_2 j = -\left(g \ , \ R_h \left[\frac{\partial \mathbf{\Psi}_h}{\partial \mathbf{W}_h}\right]^{-1} T_h \Theta_h(W_h)\right)_{L^2(\Omega)} \ . \tag{15}$$

Also follows the **adjoint-based goal-oriented correctors**:

$$\delta_1 j = -\left(p_h \ , \ T_h \Psi(W_h)\right)_{L^2(\Omega)} \tag{16}$$

$$\delta_2 j = -\left(p_h \ , \ T_h \Theta_h(W_h)\right)_{L^2(\Omega)} \ . \tag{17}$$

We recognize in ([?]) the superconvergent corrector of [?]. We observe that, thanks to the choice of T_h as the adjoint operator of R_h, the linearised-based and adjoint-based formulation are perfectly equivalent. At the contrary, the effort to compute them are very different, particularly in the case of unsteady PDE, since the adjoint system has to be solved reverse in time, while using the state solution at all time levels. This remark leads to the following recommendations:

- use the direct linearised formulation in any case you only need a corrector for the field as well as a corrector for one or several output functionals.

- the adjoint formulation is compulsory when you wish to derive an goal-oriented optimal mesh.

The second recommendation is motivated by the fact that an optimal mesh will be derived from minimisation of the error term in which we need to put in evidence the dependance of error with respect to mesh. Since the adjoint is an approximation of a continuous function, it does not much depend of mesh. At the contrary, the continuous residual $T_h\Psi(W_h)$ or the truncation error $T_h\Theta_h(W_h)$ are proportional to a power of the mesh size. In [14], the truncation error is expressed in terms of second derivatives of solution field and allows the derivation of an optimal mesh.

3.2 An example

To end this discussion, we give an numerical example of corrector evaluation built on a finite-element approximation. we can write Euler equations under the form:

$$\begin{aligned} W \in V = H^1(\Omega)^5, \quad \forall \phi \in V, \\ \int_\Omega \mathcal{F}(W)\nabla\phi \, d\Omega - \int_{\partial\Omega} \phi \hat{\mathcal{F}}(W).n \, d\Gamma \ = \ 0 \, . \end{aligned} \tag{18}$$

where $\hat{\mathcal{F}}(W)$ accounts for the different boundary conditions. Let us introduce a discretization of the previous EDP. Let τ_h a tetrahedrization of Ω with N vertices. It will rely on a discrete space of functions:

$$V_h = \{\phi_h \in H^1(\Omega)^5, \ \forall T \in \tau_h, \phi_h|T \in \mathcal{P}^1\}$$

the canonical basis of which is denoted:

$$V_h = span[N_i], \ N_i(x_j) = \delta_{ij} \forall i,j, \text{vertices of } \tau_h,$$

and on the interpolation operator:

$$\Pi_h : \mathcal{V} \to V_h, \ \Pi_h\phi(x_i) = \phi(x_i), \forall i, \text{vertex of } \tau_h.$$

Comparing with the previous abstract theory, we get:

$$\begin{aligned} R_h : \quad & \mathbb{R}^{5N} \to V_h, \quad & \mathbf{f}_h \mapsto R_h\mathbf{f}_h = \Sigma_i [\mathbf{f}_h]_i N_i \ , \\ T_h : \quad & \mathcal{V} \to \mathbb{R}^{5N}, \quad & \phi \mapsto T_h\phi = [\phi(x_i)] \ . \end{aligned}$$

The discretization is set into the discrete space, but also it differs from the continuous statement in two features, a discrete flux $\mathcal{F}_h$ instead of $\mathcal{F}$:

$$\mathcal{F}_h : \mathcal{V} \to V'$$

and an extra term of artificial diffusion D_h:

$$\begin{aligned} & W_h \in V_h, \quad \forall \phi_h \in V_h, \\ & (\Psi_h(W_h), \phi_h)_{V' \times V} = 0, \\ & \qquad \text{with} \\ & (\Psi_h(W_h), \phi_h)_{V' \times V} = \\ & \int_\Omega \phi_h \nabla.\mathcal{F}_h(W_h) \, d\Omega - \int_\Gamma \phi_h \bar{\mathcal{F}}_h(W_h).n \, d\Gamma \\ & + \int_\Omega \phi_h \, D_h(W_h) d\Omega \, . \end{aligned} \tag{19}$$

The discrete fluxes are chosen as follows:

$$\begin{aligned} \mathcal{F}_h(W) = \mathcal{F}_h(\Pi_h W) = \Pi_h \mathcal{F}(\Pi_h W) \, . \\ \bar{\mathcal{F}}_h(W) = \bar{\mathcal{F}}_h(\Pi_h W) = \Pi_h \bar{\mathcal{F}}(\Pi_h W). \end{aligned} \tag{20}$$

After some calculations and simplifications, the main error term appears as follows:

$$\begin{aligned} & (\frac{\partial \Psi_h}{\partial W_h}\delta W_h, \phi_h) = \\ & - \int_\Omega \nabla\phi_h (\mathcal{F}(W) - \Pi_h \mathcal{F}(W)) d\Omega \\ & + \int_\Gamma \phi_h (\hat{\mathcal{F}}^{out}(W) - \Pi_h \hat{\mathcal{F}}^{out}(W)).n \, d\Gamma \end{aligned} \tag{21}$$

with $\bar{\mathcal{F}}(W).n = \mathcal{F}(W).n - \hat{\mathcal{F}}(W).n$. A Gauss quadrature is applied for the evaluation of the right hand side. We have applied this to a steady subsonic flow and give some preliminary results. Figure 2 compares the entropy generation in the flow computed directly and the same flow corrected by formula (21). Entropy level is one order of magnitude smaller.

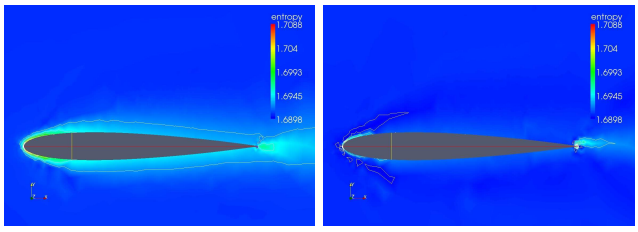

Figure 2: Entropy spurious generation for a direct computation of a steady flow and for a corrected one.

4 Uncertainty propagation techniques

In optimization problems, uncertainty propagation analysis may concern the study of the *cost functional*

$$j : \gamma \mapsto j(\gamma) := J(\gamma, W) \in \mathbb{R} \tag{22}$$

where all varying parameters are represented by the *uncertain (i.e. not-deterministic) control variables* $\gamma \in \mathbb{R}^n$, and where the state variables $W = W(\gamma) \in \mathbb{R}^N$ are solution of the (nonlinear) *state equation*

$$\Psi(\gamma, W) = 0. \tag{23}$$

It is important to note that the state equation (23) contains the governing PDE of the mathematical model of the physical system of interest (for example the stationary part of the Euler or Navier-Stokes equations) and it can be viewed as an *equality constraint* for the functional (22). The basic *probabilistic* approaches for analyzing the propagation of uncertainties are Monte-Carlo methods. A full nonlinear Monte-Carlo method gives us a complete and exact information about uncertainty propagation in the form of its PDF, but with a prohibitively expensive cost in terms of CPU time. In NODESIM-CFD, several other *probabilistic* approaches for analyzing the propagation of uncertainties are considered such as Latine Hypercubes and Polynomial Chaos. We contribute on perturbative methods based on the Taylor expansion.

4.1 Perturbation methods

To reduce the computational cost, we may think to use only some (derivate) quantities characterizing the distribution of the input variables instead of an entire sample drawn from a population with a given PDF. Therefore, the idea behind the Method of Moments is based on the Taylor series expansion of the original nonlinear functional (22) around the *mean value* of the input control ($\mu_\gamma = E[\gamma]$), and then computing some statistical moments of the output (usually mean and variance). In this way, we are assuming that the input control γ can be decomposed as sum of a fully deterministic quantity μ_γ with a stochastic perturbation $\delta\gamma_u$ with the property $E[\delta\gamma_u] = 0$. With these definitions, the Taylor series expansion of the functional $j(\gamma)$ around the mean value μ_γ is:

$$j(\gamma) = j(\mu_\gamma + \delta\gamma_u) = j(\mu_\gamma) + G\delta\gamma_u + \\ + \frac{1}{2}\delta\gamma_u^* H \delta\gamma_u + O(||\delta\gamma_u||^3) \tag{24}$$

where $G = \left.\frac{\partial j}{\partial \gamma_u}\right|_{\mu_\gamma}$ is the gradient of the functional respect to the uncertain variables and $H = \left.\frac{\partial^2 j}{\partial {\gamma_u}^2}\right|_{\mu_\gamma}$ is the Hessian matrix, both evaluated at the mean of the input variables μ_γ.

By considering various orders of the Taylor expansion (24) and taking the first and the second statistical moment, we can approximate the mean μ_j and the variance σ_j^2 of the functional $j(\gamma)$ in terms of its derivatives evaluated at μ_γ and in terms of statistical moments of the control γ.

First order moment methods:

$$\begin{cases} \mu_j = j(\mu_\gamma) + O\Big(E\big[\delta\gamma_u^2\big]\Big) \\ \sigma_j^2 = E\Big[\big(G\delta\gamma_u\big)^2\Big] + O\Big(E\big[\delta\gamma_u^3\big]\Big) \end{cases} \tag{25}$$

Second order moment methods:

$$\mu_j = j(\mu_\gamma) + \frac{1}{2}E\big[\delta\gamma_u^* H \delta\gamma_u\big] \\ + O\Big(E\big[\delta\gamma_u^3\big]\Big) \\ \sigma_j^2 = \\ E\Big[\big(G\delta\gamma_u\big)^2\Big] + E\big[\big(G\delta\gamma_u\big)\big(\delta\gamma_u^* H\delta\gamma_u\big)\big] \\ -\frac{1}{4}E\big[\delta\gamma_u^* H\delta\gamma_u\big]^2 \quad + \frac{1}{4}E\Big[\big(\delta\gamma_u^* H\delta\gamma_u\big)^2\Big] \\ + O\Big(E\big[\delta\gamma_u^4\big]\Big)$$

With this method it is clear that we are using only some partial informations about the input uncertainties, in fact we are using only some statistical moments of the control variable instead of full information available with its PDF, and we will not have anymore the PDF of the propagated uncertainty, but only its approximate mean and variance. Another important point is that the Method of Moments is applicable only for *small* uncertainties, due to the local nature of Taylor expansion approximation.
Two things should be noted here: the first one is that for the Method of Moments *we need the derivatives* of the functional respect to the control variables affected by uncertainties: in particular we need the gradient for the first order method, and gradient and Hessian for the second order method. Due to the fact that $j(\gamma) = J(\gamma, W)$, where $W = W(\gamma)$ is solution of the state equation (23) we have for the derivative:

$$\frac{\partial j}{\partial \gamma_u} = \frac{\partial J}{\partial \gamma_u} + \frac{\partial J}{\partial W}\frac{\partial W}{\partial \gamma_u}$$

Since we know the solution $W(\gamma)$ by its numerical values as result of a program (implementing an appropriate method, e.g. fixed point method), it is intersting to use of Automatic Differentiation tools (like TAPENADE) in order to obtain the needed derivatives. The same remarks apply to the computation of the Hessian matrix. In particular we note that the derivatives are computed at the mean value of the control μ_γ, so they are fully deterministic and can be picked out from the expectations in the equations (25) or (26). In other words we can write

$$E\Big[\big(G\delta\gamma_u\big)^2\Big] = \\ \sum_{i,k} G_i G_k E\big[\delta\gamma_u^{(i)}\delta\gamma_u^{(k)}\big] = \sum_{i,k} G_i G_k C_{ik} \\ E\big[\delta\gamma_u^* H\delta\gamma_u\big] = \\ \sum_{i,k} H_{ik} E\big[\delta\gamma_u^{(i)}\delta\gamma_u^{(k)}\big] = \sum_{i,k} H_{ik} C_{ik} \\ E\big[\big(G\delta\gamma_u\big)\big(\delta\gamma_u^* H\delta\gamma_u\big)\big] = \\ \sum_{i,k,l} G_l H_{ik} E\big[\delta\gamma_u^{(i)}\delta\gamma_u^{(k)}\delta\gamma_u^{(l)}\big] \\ E\Big[\big(\delta\gamma_u^* H\delta\gamma_u\big)^2\Big] = \\ \sum_{i,k,l,m} H_{ik} H_{lm} E\big[\delta\gamma_u^{(i)}\delta\gamma_u^{(k)}\delta\gamma_u^{(l)}\delta\gamma_u^{(m)}\big] \tag{26}$$

where $G_i = \left.\frac{\partial j}{\partial \gamma_u^{(i)}}\right|_{\mu_\gamma}$ are the elements of the gradient,

$H_{ik} = \dfrac{\partial^2 j}{\partial\gamma_u^{(i)}\partial\gamma_u^{(k)}}\Big|_{\mu_\gamma}$ are the elements of the Hessian matrix and $C_{ik} = E\big[\delta\gamma_u^{(i)}\delta\gamma_u^{(k)}\big] = \mathrm{cov}(\gamma_u^{(i)}, \gamma_u^{(k)})$ are the elements of the *covariance matrix*. Every expectaction term $E[\dots]$ in the equations (4.1), is defined by the statistical model of the uncertainties and could be computed in a preprocessing phase.

For example, for the important case where the uncertainties are random and normally distribuited, we have:

$$\begin{aligned} &E\big[\delta\gamma_u^{(i)}\delta\gamma_u^{(k)}\delta\gamma_u^{(l)}\big] = 0 \\ &E\big[\delta\gamma_u^{(i)}\delta\gamma_u^{(k)}\delta\gamma_u^{(l)}\delta\gamma_u^{(m)}\big] = \\ &C_{ik}C_{lm} + C_{il}C_{km} + C_{im}C_{kl} \end{aligned}$$

and if these (normal) uncertainties are idependents, then holds the relation $C_{ik} = \sigma_i^2\delta_{ij}$ where $\sigma_i^2 = E\big[\delta\gamma_u^{(i)}\delta\gamma_u^{(i)}\big]$ and the equations (4.1) become

$$\begin{aligned} &E\Big[\big(G\delta\gamma_u\big)^2\Big] = \sum_i G_i^2\sigma_i^2 \\ &E\big[\delta\gamma_u^* H\delta\gamma_u\big] = \sum_i H_{ii}\sigma_i^2 \\ &E\big[\big(G\delta\gamma_u\big)\big(\delta\gamma_u^* H\delta\gamma_u\big)\big] = 0 \\ &E\Big[\big(\delta\gamma_u^* H\delta\gamma_u\big)^2\Big] = \sum_{i,k}\big(H_{ii}H_{kk} + 2H_{ik}^2\big)\sigma_i^2\sigma_k^2 \end{aligned} \tag{27}$$

Since we have the term $E\big[\big(\delta\gamma_u^* H\delta\gamma_u\big)^2\big]/4$, the error is still of the order of $E\big[\delta\gamma_u^4\big]$. Computing the other terms of same order require the knowledge of order of derivatives higher than the second. From the previous discussion, it is clear that in order to apply the Method of Moments we need to solve only one (expensive) nonlinear system with derivatives (at the mean μ_γ) and then apply the (inexpensive) equations (25) or (26) where, for the fully nonlinear Monte-Carlo approach of the previous section, we need to solve $N \gg 1$ nonlinear systems (23).

4.2 First and second-order derivatives of a functional

We are interested by obtaining the first and second derivatives of a functional j depending of $\gamma \in \mathbb{R}^n$, and expressed in terms of a state $W \in \mathbb{R}^N$ as follows:

$$\begin{cases} \psi(\gamma) = \Psi(\gamma, W(\gamma)) = 0 \\ j(c) = J(\gamma, W(\gamma)) \end{cases} \tag{28}$$

Our problem can be viewed from two different point of view: the first one is consider the solution algorithm for state equation as part of j itself, i.e. considering j as a function of the control variables γ only. The second one is consider the system made by two different routines: one of them is the routine that solves the nonlinear system $\Psi(\gamma, W(\gamma)) = 0$ (and contains the evaluation the residual $\Psi(\gamma, W)$), and the other is the routine $J(\gamma, W)$ that computes the value of the functional from the state variables W and (eventually) the control variables γ.

The first approach lead to a straightforward algorithm for first order derivatives, in fact we just need to differentiate the entire routine j with tangent or reverse mode . In this context, the routine j contains the iterative solver method for the state equation, and the differentiated routines will also contains this loop in differentiated form. If we need n_{iter} loop iterations in order to obtain the nonlinear solution, and we assume for each iteration an unitary cost, we can analyze the cost for the gradient of the functional.

Using tangent mode, the cost for the entire gradient will be $n(n_{\mathrm{iter}}\alpha_T)$ where n is the number of components of the gradient and $1 < \alpha_T < 4$ is the overhead associated with the differentiated code respect to the original one. For this stategy, the memory requirements will be of the same order of the undifferentiated code.

With reverse mode we are able to obtain the entire gradient with a single evaluation of the differentiated routine, but the total cost (in terms of CPU time and memory) will depends on the strategy used by the AD tool to solve the problem of inverse order differentiation for the original routine. For the case of a Store-All (**SA**) strategy, the CPU cost will be $(n_{\mathrm{iter}}\alpha_R)$ with $1 < \alpha_R <$, i.e. α_R times the undifferentiated code, but the required memory will be n times greater. For a Recompute-All (**RA**) strategy the CPU cost will be $(n_{\mathrm{iter}}^2\alpha_R)$, i.e. $(n_{\mathrm{iter}}\alpha_R)$ the nonlinear solution, but the memory will be the same of the undifferentiated routine. For real large programs, neither SA or RA strategy can work, so we need a special storage/recomputation trade-off in order to be efficient using *checkpoints*. Obviously, with checkpointing the CPU cost will be greater than the cost of SA strategy and can be shown that the cost for the differentiated code will be of the order of $\sqrt[s]{n_{\mathrm{iter}}}$ (where s is the number of snapshots available).

It is clear that for gradient computation with $n \gg 1$, the reverse mode is faster than tangent mode, but for a program containing an iterative algorithm, the reverse mode is not always applicable. The problem relies on the fact that the Reverse mode computation is performed in the opposite way of the original code (*backward sweep*) after a *forward sweep* needed to store the variable needed in the successive phases.

For the previous arguments, we prefer differentiate not the entire program (solution of the state equation + functional evaluation), but the two main component in a separate way, using the fact that at the solution, the residuals will be zero (i.e. we don't differentiate the routine containing the main loop, but only the quantities involved after the last iteration). For this second approach, we have to analyse the influence of state equation and the functional evaluation in more details. This is the purpose of the next sections.

First derivative

Using the chain rule, the gradient of the functional $j(\gamma) = J(\gamma, W(\gamma))$ is given by

$$\frac{dj}{d\gamma} = \frac{\partial J}{\partial\gamma} + \frac{\partial J}{\partial W}\frac{dW}{d\gamma}$$

where the derivatives of the state variables $W(c)$ are obtained solving the linear system

$$\frac{d\psi}{d\gamma} = \frac{\partial\Psi}{\partial\gamma} + \frac{\partial\Psi}{\partial W}\frac{dW}{d\gamma} = 0$$

Therefore, two strategies can be applied:

Direct differentiation :

It consists in computing the Gateaux-derivatives with respect to each component direction ($e_i = (0, \dots 0, 1, 0, \dots, 0)^T$, where 1 is at the i-esim component):

$$\frac{dj}{d\gamma_i} = \frac{dj}{d\gamma} e_i = \frac{\partial J}{\partial \gamma_i} + \frac{\partial J}{\partial W}\frac{dW}{d\gamma_i} \tag{29}$$

with:

$$\frac{\partial \Psi}{\partial W}\frac{dW}{d\gamma_i} = -\frac{\partial \Psi}{\partial \gamma} e_i \tag{30}$$

This has to be applied to each component of γ, i.e. n times and the cost is n linearised N-dimensional systems to solve. If we choose to solve the single system (30) with an iterative matrix-free method, and the solution is obtained after n_{iter} step, the total cost will be of the order of $\alpha_T n_{\text{iter},T}$, i.e. $n_{\text{iter},T}$ evaluation of the matrix-by-vector operation $\left(\frac{\partial \Psi}{\partial W}\right)x$, where each evaluation costs α_T times the evaluation of the state residual $\Psi(\gamma, W)$ (and the cost of the state residual is taken as reference equal to 1). Therfore, the cost of the full gradient will be $n\alpha_T n_{\text{iter},T}$.

Inverse differentiation (Reverse mode)

The complete gradient is given by the equation

$$\left(\frac{dj}{d\gamma}\right)^* = \left(\frac{\partial J}{\partial \gamma}\right)^* - \left(\frac{\partial \Psi}{\partial \gamma}\right)^* \Pi_0 \tag{31}$$

where Π_0 is the solution of the linear system

$$\left(\frac{\partial \Psi}{\partial W}\right)^* \Pi = \left(\frac{\partial J}{\partial W}\right)^* \tag{32}$$

This computation needs only one extra linearised N-dimensional system, the adjoint system (some methods for calculation of the adjoint solutions are described in). If we choose to solve the adjoint system (32) with an iterative matrix-free method, we can apply the same estimate done as in the case of the Tangent mode differentiation, but this time the overhead associated with the evaluation of the matrix-by-vector operation $\left(\frac{\partial \Psi}{\partial W}\right)^* x$ respect to the state residual evaluation will be α_R (and usually $\alpha_R > \alpha_T$, and the number of iteration $n_{\text{iter},R}$ for the convercence of the solution could be different from $n_{\text{iter},T}$ of the previous case (but the asymptotical rate of convergence will be the same of the original linear system $\left(\frac{\partial \Psi}{\partial W}\right)x = b$, see). Therefore the cost for the gradient will be $\alpha_R n_{\text{iter},R}$, and the Reverse mode differentiation for the gradient computation is cheaper than the Tangent mode if $n \gg 1$.

Second derivative

For second derivatives we have different possibilities:

Direct-direct option

This methods was initially investigated by along with various other algorithms, but the publication does not go into the implementation details for a generic fluid dynamic code. Here we present the mathematical background behind the idea and the efficient AD implementation of Ghate and Giles but with a different analysis of the computational cost.

Starting from the derivative (29), we perform another differentiation respect to the variable γ_k obtaining

$$\frac{d^2 j}{d\gamma_i d\gamma_k} = D^2_{i,k} J + \frac{\partial J}{\partial W}\frac{d^2 W}{d\gamma_i d\gamma_k} \tag{33}$$

where

$$D^2_{i,k} J = \frac{\partial}{\partial \gamma}\left(\frac{\partial J}{\partial \gamma} e_i\right) e_k + \frac{\partial}{\partial W}\left(\frac{\partial J}{\partial \gamma} e_i\right)\frac{dW}{d\gamma_k} + \frac{\partial}{\partial W}\left(\frac{\partial J}{\partial \gamma} e_k\right)\frac{dW}{d\gamma_i} + \frac{\partial}{\partial W}\left(\frac{\partial J}{\partial W}\frac{dW}{d\gamma_i}\right)\frac{dW}{d\gamma_k}$$

Differentiating the equation (30) we get

$$D^2_{i,k}\Psi + \frac{\partial \Psi}{\partial W}\frac{d^2 W}{d\gamma_i d\gamma_k} = 0 \tag{34}$$

where

$$\begin{aligned} D^2_{i,k}\Psi = &\frac{\partial}{\partial \gamma}\left(\frac{\partial \Psi}{\partial \gamma} e_i\right) e_k \\ &+ \frac{\partial}{\partial W}\left(\frac{\partial \Psi}{\partial \gamma} e_i\right)\frac{dW}{d\gamma_k} \\ &+ \frac{\partial}{\partial W}\left(\frac{\partial \Psi}{\partial \gamma} e_k\right)\frac{dW}{d\gamma_i} + \\ &\frac{\partial}{\partial W}\left(\frac{\partial \Psi}{\partial W}\frac{dW}{d\gamma_i}\right)\frac{dW}{d\gamma_k} \end{aligned}$$

Substituting the second derivatives of the state respect to the control variables $\frac{d^2 W}{d\gamma_i d\gamma_k}$ in equation (33) from equation (34) we get

$$\begin{aligned} \frac{d^2 j}{d\gamma_i d\gamma_k} &= D^2_{i,k} J - \frac{\partial J}{\partial W}\left(\frac{\partial \Psi}{\partial W}\right)^{-1} D^2_{i,k}\Psi \\ &= D^2_{i,k} J - \Pi_0^* D^2_{i,k}\Psi \end{aligned} \tag{35}$$

where Π_0 is the solution of the adjoint system (32) evaluated at the point $(\gamma, W(\gamma))$ solution of the state equation $\Psi(\gamma, W) = 0$. The n derivatives $\dfrac{dW}{d\gamma_i}$ should be computed (and stored) using tangent mode differentiation of the nonlinear solver algorithm, and each derivatives costs $n_{\text{iter}}\alpha_T$. If we need the full Hessian matrix we have to evaluate the quantity (35) $n(n+1)/2$ times, i.e. we have to evaluate the terms $D^2_{i,k}\Psi$ and $D^2_{i,k}J$ for $i = 1, \dots, n$ and $j = i, \dots, n$ due to the simmetry of the Hessian, and each evaluation of $(D^2_{i,k}\Psi$ costs α^2_T (the evaluation of $D^2_{i,k}J)$ is negligible respect to $(D^2_{i,k}\Psi)$. Therfore the full Hessian costs $n\alpha_T[n_{\text{iter},T} + (n+1)\alpha_T/2]$. With similar arguments, if we want only the diagonal part of the Hessian, the cost is $n\alpha_T[n_{\text{iter},T} + \alpha_T]$.

Inverse-direct

This consists in the direct derivation in any direction $e_i, i = 1, n$ of the (non-scalar) function:

$$\left(\frac{\partial j}{\partial c}\right)^* (c, W(c)) = \left(\frac{\partial J}{\partial c}\right)^* (c, W(c)) - \left(\frac{\partial \Psi}{\partial c}\right)^* \Pi(c, W(c))$$

where $W(c)$ and $\Pi(c, W(c))$ are solutions of the above two state systems. With some algebra we obtain

$$\frac{\partial}{\partial c_i}\left(\frac{\partial j}{\partial c}\right)^* = \left(\frac{\partial^2 j}{\partial c^2}\right) e_i = \frac{\partial}{\partial c}\left(\frac{\partial J}{\partial c}\right)^* e_i + \frac{\partial}{\partial W}\left(\frac{\partial J}{\partial c}\right)^* \theta_i - \frac{\partial}{\partial c}\left[\left(\frac{\partial \Psi}{\partial c}\right)^* \Pi_0\right] e_i - \frac{\partial}{\partial W}\left[\left(\frac{\partial \Psi}{\partial c}\right)^* \Pi_0\right] \theta_i - \left(\frac{\partial \Psi}{\partial c}\right)^* \lambda_i$$

The derivation needs the solution of the adjoint systems

$$\left(\frac{\partial \Psi}{\partial W}\right)^* \Pi_0 = \left(\frac{\partial J}{\partial W}\right)^* \tag{36}$$

and $2n$ perturbed N-dimensional linear systems (for the full Hessian):

$$\begin{cases} \dfrac{\partial \Psi}{\partial W}\theta_i = -\dfrac{\partial \Psi}{\partial c} e_i \\ \left(\dfrac{\partial \Psi}{\partial W}\right)^* \lambda_i = \dfrac{\partial}{\partial c}\left(\dfrac{\partial J}{\partial W}\right)^* e_i \\ + \dfrac{\partial}{\partial W}\left(\dfrac{\partial J}{\partial W}\right)^* \theta_i - \dfrac{\partial}{\partial c}\left[\left(\dfrac{\partial \Psi}{\partial W}\right)^* \Pi_0\right] e_i \\ - \dfrac{\partial}{\partial W}\left[\left(\dfrac{\partial \Psi}{\partial W}\right)^* \Pi_0\right] \theta_i \end{cases}$$

where all the functions in the equations (36)–(37) are evalued at the final state (in order to verify $\Psi(c, W(c)) = 0$).

Inverse-inverse

If we are interested in a (scalar!) functional depending on the gradient, then it can be interesting to apply a second inverse differentiation. We do not focus on this direction at the moment.

5 Numerical experiments

The testcase considered here corresponds to the optimization of the wing shape of a business aircraft (courtesy of Piaggio Aero Ind.), for a transonic regime (see the shape and the mesh in Fig. 3). The nominal operational conditions are defined by the free-stream Mach number $M_\infty = 0.83$ and the incidence $\alpha = 2°$. We suppose that only these two quantities are subject to random fluctuations. For simplicity, we assume that their PDF are Gaussian with given mean and variance. The mean values correspond to the nominal values. The section of the initial wing shape corresponds to the NACA 0012 airfoil.

For the present work, due to the fact that we consider only two uncertain variables, we used a ToT approach for the Hessian evaluation. A comparision of the resulting approximate drag coefficient with respect to the real value is given in Fig. 5. We want to point out the fact that the second-order approximation requires to solve only one nonlinear state equation $\Psi = 0$ plus 4 linear systems using ToT. The real values of the functional plotted in Fig 4 require 21×21 nonlinear simulations.

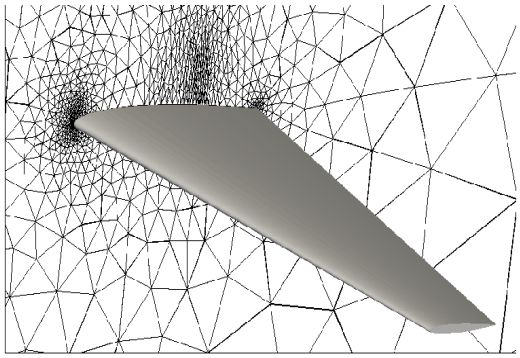

Figure 3: Wing shape and mesh in the symmetry plane.

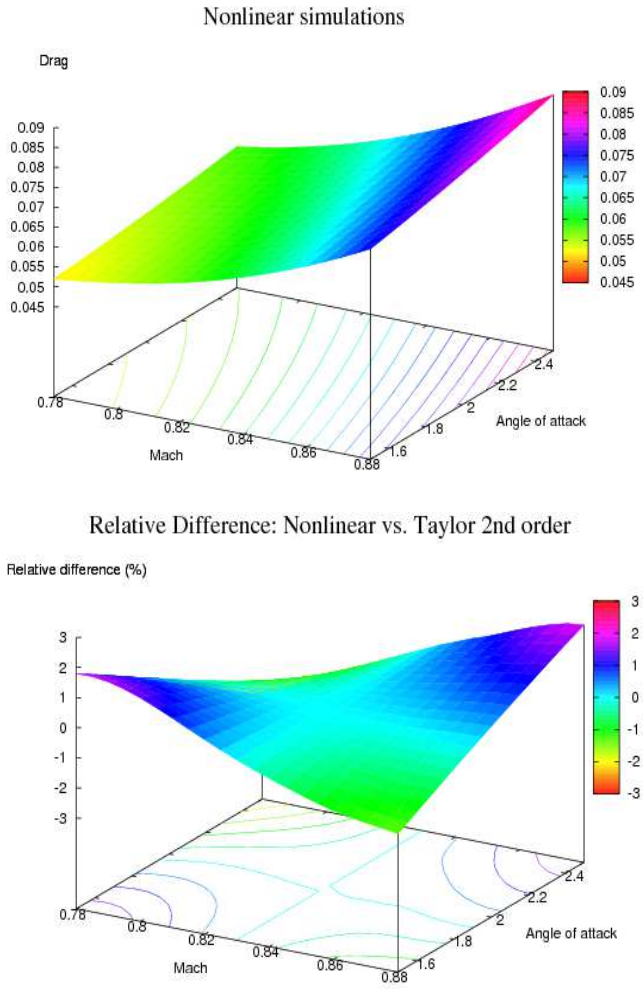

Figure 4: Drag coefficient vs. Mach number and angle of attack (first-order spatial accuracy) for the transonic wing: nonlinear simulations (first image); percentage relative difference between the nonlinear simulations and the second-order Taylor approximation (second image). For the top plot we have solved 21×21 nonlinear systems $\Psi = 0$.

Acknowlegments: This studies are supported in part by NODESIM-CFP FP-6 European project.

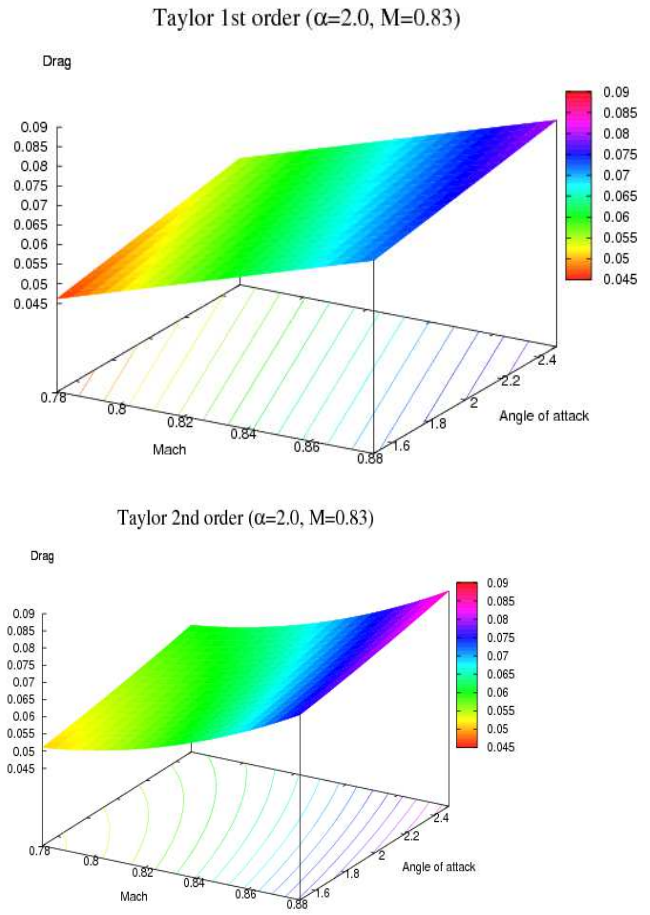

Figure 5: First-order (first image) and second-order (second image) Taylor approximation around $\alpha = 2°$ and $M = 0.83$; For the 1st and 2nd-order Taylor approximation, we have solved only one nonlinear system.

References

[1] Pascual V., Hascoët, L.,"Extension of TAPENADE towards Fortran 95", *in* Automatic Differentiation: Applications, Theory, and Tools, Lecture Notes in Computational Science and Engineering, Selected papers from AD2004 Chicago, July 2005, Springer, 2005.

[2] Hascoët L. and Dauvergne B., "Adjoints of large simulation codes through Automatic Differentiation", *European Journal of Computational Mechanics*, 17:63-86, 2008.

[3] Hascoët L., Naumann U., Pascual V., ""To be recorded" analysis in reverse-mode automatic differentiation", *Future Generation Comp. Syst.* 21(8): 1401-1417 (2005)

[4] Pascual V., Hascoët L., "TAPENADE for C", Advances in Automatic Differentiation, Springer, 2008

[5] Naumann U., Hascoët L., Hill C., Hovland P.D., Riehme J., Utke J., " A Framework for Proving Correctness of Adjoint Message-Passing Programs", PVM/MPI Dublin, Ireland 2008: 316-321

[6] Hascoët L., Utke J., Naumann U.,"Cheaper Adjoints by Reversing Address Computations", *Scientific Programming*, 16:1, 2008

[7] Tber M.-H., Hascoët L., Vidard A., Dauvergne B., "Building the Tangent and Adjoint codes of the Ocean General Circulation Model OPA with the Automatic Differentiation tool TAPENADE", Research Report, INRIA, 2007, no 6372, http://hal.inria.fr/inria-00192415. Also in The Computing Research Repository (CoRR)abs/0711.4444: (2007)

[8] Martinelli M.,"Sensitivity Evaluation in Aerodynamics Optimal Design",Ph. Dissertation, Scuola Normale di Pisa (Italy), 2007

[9] Martinelli M., Dervieux A., Hascoët L., "Strategies for computing second-order derivatives in CFD design problems", *in*: Proc. of West-East High Speed Flow Field Conference, Moscou, Russia, nov. 19-22 2007.

[10] Martinelli M., Hascoët L., "Tangent-on-Tangent vs. Tangent-on-Reverse for Second Differentiation of Constrained Functionals", Proceedings of the AD2008 Conference, Bonn, Germany, august 2008, to appear in Lecture Notes in Computational Science and Engineering, Springer.

[11] Martinelli M., Duvigneau, R., "Comparison of second-order derivatives and metamodel-based Monte-Carlo approaches to estimate statistics for robust design of a transonic wing", *AIAA* 2008-2071, Proceedings of the 10th AIAA Non-Deterministic Approaches Conference, April 7-10, 2008, Schaumburg (IL), USA.

[12] Martinelli M., Praveen C., Duvigneau R., "On the estimation of drag uncertainty", Proceedings of the 43rd AAAF Congress on Applied Aerodynamics, March 10-12, 2008, Poitiers, France.

[13] Hascoët L. and Pascual V., "TAPENADE 2.1 user's guide", INRIA Techincal Report, http://www.inria.fr/rrrt/rt-0300.html,year 2004

[14] A. Loseille, "Adaptation de maillage anosotrope 3D multi-echelles et ciblée à une fonctionnelle pour la Mécanique des Fluides, application à la prédiction du bang sonique" (in french), Ph. Dissertation at University P.M. Curie, 2008. , http://tel.archives-ouvertes.fr/tel-00361961/fr/

[15] NODESIM-CFD, Non-deterministic simulation for CFD-based design technologies, FP6 program, http://www.nodesim.eu/conference.html

[16] Michael B. Giles and Niles A.Pierce, "An introduction to the Adjoint Approach to Design" 2000, Flow, Turbulence and Combustion 65:393-415

[17] Pierce N. A. and Giles M. B., "Adjoint recovery of superconvergent functionals from approximate solutions of partial differential equations", *Oxford University Computing Laboratory Report*, AMS(MOS): 65G99,76N15, December 1998