Strategies for Computing Second-Order Derivatives in CFD Design Problems

Submitted to WEHSFF2007, Moscow, Topic: Multidisciplinary Optimization

Massimiliano Martinelli, Alain Dervieux, B.P. 93, 06902 Sophia-Antipolis, FRANCE
{Massimiliano.Martinelli, Alain.Dervieux}@sophia.inria.fr

Important advances in CFD design methods as been obtained using first-order derivatives and adjoints. However, in order to take into account uncertainties, reduced model are built with second-order derivatives ([9], [5], [1]). New functionals involving first-order derivatives are also used for robust design ([8], [5]) so their gradients will involve second-order derivatives. For these developments, building second-order derivatives is a mandatory task. We analyze strategies for computing second-order derivatives using Automatic Differentiation tools. The target is a solver for shape design with supersonic flow.

1 CFD Model

In our design problem the shape of an obstacle is defined from a shape parameter γ. For a γ we can define the external domain around the obstacle, Ω_γ, then the stationary Euler equations can be written as follows:

$$W_1 = \rho, W_2 = \rho u, W_3 = \rho v, W_4 = \rho w, W_5 = E; F(W)_x + G(W)_y + H(W)_z = 0 \tag{1}$$

$$\begin{aligned}(\Psi(\gamma, W), \phi) = &- \int_{\Omega_\gamma} (F(W) \cdot \frac{\partial \phi}{\partial x} + G(W) \cdot \frac{\partial \phi}{\partial y} + H(W) \cdot \frac{\partial \phi}{\partial z})\, d\Omega_\gamma \\ &+ \int_{\partial\Omega_B} (F_B n^x + G_B n^y + H_B n^z) \cdot \phi \, d\sigma \\ &+ \int_{\partial\Omega_\gamma} p(W)(n_x^\gamma \phi_2 + n_y^\gamma \phi_3 + n_z^\gamma \phi_4)\, d\sigma = 0.\end{aligned} \tag{2}$$

The notations F, G, H are the classical fluxes of the Euler equations and $p(W)$ is the pressure (that depends on W through the state law). The second integral keep in account the inflow/outflow conditions and the third one the (physical) boundary conditions. This is discretized with a second-order upwind scheme applying on tetrahedra. A differentiable functional $j(\gamma) = J(\gamma, W(\gamma))$ is introduced, strategies for gradient evaluation are given in ([4]), the subject of this work is strategies for computing second derivatives.

2 Second-order derivatives of a functional

We observe that $\gamma \to W(\gamma)$ and $\gamma \to j(\gamma)$ are functions *implicitly* defined through the state equation. Then we consider two different points of view:

- *Implicit differentiation:* it consists to differentiate directly the *implicit* function j as a function of the control variable γ.
- *Differentiation of explicit parts:* the second point of view is to apply differentiation only to routines which compute *explicit* functions, that is functions Ψ and J.

In the first approach, differentiating the entire routine j can be performed with either Tangent or Reverse mode (see [3] or [2] for definitions). It permits to obtain directly a differentiated program, in a black box manner, with the risk that this program has a rather good reliability but not sufficiently good performances.

Since the routine j contains the iterative solver method for the state equation, the differentiated routines will contain this solver in differentiated form. We assume that we need n_{iter} iterations in order to obtain the nonlinear solution, and that for each iteration we have an unitary cost. **Tangent mode** produces a program that we need to apply n time for computing the entire gradient. The cost is $n(n_{\text{iter}}\alpha_T)$ where α_T is usually $1 < \alpha_T < 4$, see for example [2]. Memory requirements is about the same. With **Reverse mode**, we are able to obtain the entire gradient with a single evaluation of the differentiated routine. But the usual Reverse mode produces a code which involves two successive parts: a *forward sweep* close to original code and a *backward sweep* needing data computed in the *forward sweep*, but in the opposite way. In one option to solve this, the Store-All (SA) strategy, these data are stored during the *forward sweep*. The CPU cost will be $(n_{\text{iter}}\alpha_R)$ with $1 < \alpha_R < 5$, 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 $(n^2_{\text{iter}}\alpha_R)$, i.e. $(n_{\text{iter}}\alpha_R)$ the nonlinear solution, but the memory will be the same of the undifferentiated routine. For real large programs trade-off between SA and RA are compulsory (see [3]).

When the iterative algorithm is a fixed point one, with enough convergence to the fixed point, only the final state variable is necessary for *backward sweep*. Lastly, the behaviour (efficiency, stability) of differentiated solution algorithms is questionable, while a natural and conservative tendancy is to re-use the algorithm used for state -preconditionner, iterator- for solving adjoint systems. For the previous arguments, in case of simple explicit solvers, the implicit differentiation is a good option, but for more sophisticated solvers (GMRES,...) we recommand the differentiation of explicit parts. In the sequel we analyze two strategies for second-order differencing using this last strategy.

2.1 Tangent-on-Tangent (ToT)

This methods was initially investigated by [7]. Here we present the mathematical background behind the idea and the efficient AD implementation of ([1]) but with a different analysis of the computational cost.

First of all, we need to compute the first-order derivative of the functional $j(\gamma)$ in Tangent mode, i.e. we compute the Gâteaux-derivatives with respect to each component direction ($e_i = (0, \dots 0, 1, 0, \dots, 0)^T$, where 1 is at the i-th component):

$$\frac{dj}{d\gamma_i} = \frac{dj}{d\gamma}e_i = \frac{\partial J}{\partial \gamma_i} + \frac{\partial J}{\partial W}\frac{dW}{d\gamma_i} \qquad \text{with} \qquad \frac{\partial \Psi}{\partial W}\frac{dW}{d\gamma_i} = -\frac{\partial \Psi}{\partial \gamma}e_i \ . \tag{3}$$

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 linear system $\frac{\partial \Psi}{\partial W}\frac{dW}{d\gamma_i} = -\frac{\partial \Psi}{\partial \gamma}e_i$ with an iterative matrix-free method (like GMRES, [6]), and the solution is obtained after $n_{\text{iter},T}$ 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). Therefore, the cost of the full gradient will be $n\alpha_T n_{\text{iter},T}$.

Starting from the derivative (3), we perform another differentiation with respect to the variable γ_k obtaining

$$\frac{d^2 j}{d\gamma_i d\gamma_k} = D^2_{i,k}J + \frac{\partial J}{\partial W}\frac{d^2 W}{d\gamma_i d\gamma_k} \tag{4}$$

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 $\frac{\partial\Psi}{\partial W}\frac{dW}{d\gamma_i} = -\frac{\partial\Psi}{\partial\gamma}e_i$ we get

$$D^2_{i,k}\Psi + \frac{\partial\Psi}{\partial W}\frac{d^2W}{d\gamma_i d\gamma_k} = 0 \tag{5}$$

where the equation for the term $D^2_{i,k}\Psi$ is analogous to the equation for $D^2_{i,k}J$ above. Substituting the second derivatives of the state with respect to the control variables $\frac{d^2W}{d\gamma_i d\gamma_k}$ in equation (4) from equation (5) we get

$$\frac{d^2 j}{d\gamma_i d\gamma_k} = D^2_{i,k}J - \frac{\partial J}{\partial W}\left(\frac{\partial\Psi}{\partial W}\right)^{-1}D^2_{i,k}\Psi = D^2_{i,k}J - \Pi_0^* D^2_{i,k}\Psi \tag{6}$$

where Π_0 is the solution of the adjoint system $\left(\frac{\partial\Psi}{\partial W}\right)^*\Pi = \left(\frac{\partial J}{\partial W}\right)^*$ evaluated at the point $(\gamma, W(\gamma))$ solution of the state equation $\Psi(\gamma, W) = 0$.

2.2 Tangent-on-Reverse (ToR)

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

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

where $W(\gamma)$ is the flow solution, and $\Pi(\gamma, W(\gamma))$ is the solution of the adjoint system $\left(\frac{\partial\Psi}{\partial W}\right)^*\Pi = \left(\frac{\partial J}{\partial W}\right)^*$. With some algebra we obtain

$$\boxed{\begin{aligned}\frac{\partial}{\partial\gamma_i}\left(\frac{\partial j}{\partial\gamma}\right)^* = \left(\frac{\partial^2 j}{\partial\gamma^2}\right)e_i = \frac{\partial}{\partial\gamma}\left(\frac{\partial J}{\partial\gamma}\right)^* e_i + \frac{\partial}{\partial W}\left(\frac{\partial J}{\partial\gamma}\right)^*\frac{dW}{d\gamma_i} + \\ -\frac{\partial}{\partial\gamma}\left[\left(\frac{\partial\Psi}{\partial\gamma}\right)^*\Pi_0\right]e_i - \frac{\partial}{\partial W}\left[\left(\frac{\partial\Psi}{\partial\gamma}\right)^*\Pi_0\right]\frac{dW}{d\gamma_i} - \left(\frac{\partial\Psi}{\partial\gamma}\right)^*\lambda_i\end{aligned}} \tag{7}$$

For each $i = 1, \ldots, n$, Equation (7) needs Π_0, the solution of the adjoint system

$$\left(\frac{\partial\Psi}{\partial W}\right)^*\Pi = \left(\frac{\partial J}{\partial W}\right)^* \tag{8}$$

and two perturbed N-dimensional linear systems:

$$\left\{\begin{aligned}&\frac{\partial\Psi}{\partial W}\frac{dW}{d\gamma_i} = -\frac{\partial\Psi}{\partial\gamma}e_i \\ &\left(\frac{\partial\Psi}{\partial W}\right)^*\lambda_i = \frac{\partial}{\partial\gamma}\left(\frac{\partial J}{\partial W}\right)^* e_i + \frac{\partial}{\partial W}\left(\frac{\partial J}{\partial W}\right)^*\frac{dW}{d\gamma_i} + \\ &\qquad -\frac{\partial}{\partial\gamma}\left[\left(\frac{\partial\Psi}{\partial W}\right)^*\Pi_0\right]e_i - \frac{\partial}{\partial W}\left[\left(\frac{\partial\Psi}{\partial W}\right)^*\Pi_0\right]\frac{dW}{d\gamma_i}\end{aligned}\right. \tag{9}$$

where all the functions in the equations (7)–(9) are evaluated at the final state (in order to verify $\Psi(\gamma, W(\gamma)) = 0$). In addition, the second linear system in (9) is of the same type of the adjoint system (8) but with a different right-hand side, so we can use the same algorithm (but with different initial data) for both equations. It is useful to note that Equation (7) gives us an entire column (or row, by symmetry) of the Hessian matrix, where the Tangent-on-Tangent approach (6) gives us a single element. The cost associated to the full Hessian can be evaluated as $n[\alpha_T n_{\text{iter},T} + \alpha_R\alpha_T + \alpha_R n_{\text{iter},R}]$.

3 Main conclusions

The algorithms for ToT and ToR approaches have a different part that characterizes the two approaches. Its cost is:

$$\frac{n(n+1)}{2}\alpha_T^2 \quad \text{for ToT} \qquad ; \qquad n\alpha_R[\alpha_T + n_{\text{iter},R}] \quad \text{for ToR.}$$

The crucial factor is the number of iterations needed to solve the adjoint system in (9). From a theoretical point of view, if we assume that $n_{\text{iter},R}$ is bounded (i.e. we have convergence in a finite number of iterations) we can say that ToR is less expensive then ToT for high values of n, but it is difficult to *a priori* say what is the value of n for which the ToR approach is preferable than the ToT. Theoretical and numerical performance will be compared for supresonic flows on various sizes of meshes.

References

[1] D. Ghate and M.B. Giles. *Inexpensive Monte Carlo uncertainty analysis*, pages 203–210. Recent Trends in Aerospace Design and Optimization. Tata McGraw-Hill, New Delhi, 2006.

[2] A. Griewank. *Evaluating Derivatives. Principles and Techniques of Algorithmic Differentiation*, volume 19 of *Frontiers in Applied Mathematics.* SIAM Philadelphia, 2000.

[3] L. Hascoët and V. Pascual. TAPENADE 2.1 user's guide. Technical Report 0300, INRIA, Sep 2004.

[4] L. Hascoet, M. Vazquez, and A. Dervieux. Automatic differentiation for optimum design, applied to sonic boom reduction. In *Proc. of International Conference on Computational Science and its Applications, ICCSA '03*, pages 85–94. Springer, 2003.

[5] M.M. Putko, P.A. Newman, A.C. Taylor III, and L.L. Green. Approach for uncertainty propagation and robust design in CFD using sensitivity derivatives. Technical Report 2528, AIAA, 2001.

[6] Y. Saad. *Iterative Methods for Sparse Linear Systems.* PWS Publishing Company, 1996.

[7] L.L. Sherman, A.C. Taylor III, L.L. Green, and P.A. Newman. First and second-order aerodynamic sensitivity derivatives via automatic differentiation with incremental iterative methods. *Journal of Computational Physics*, 129:307–331, 1996.

[8] L. Su and J.E. Renaud. Automatic Differentiation in Robust Optimization. *AIAA Journal*, 35(6), 1997.

[9] R.W. Walters and L. Huyse. Uncertainty analysis for fluid mechanics with applications. Technical Report 2002-211449, NASA, Feb 2002. ICASE Report No. 2002-1.