Design Optimization International Conference,
April 5 - April 7 2005, Las Palmas, Spain,

TOPOLOGICAL DESIGN BASED ON HIGHLY EFFICIENT ADJOINTS GENERATED BY AUTOMATIC DIFFERENTIATION

Thomas Kaminski[1], **Ralf Giering**[1], **Carsten Othmer**[2]

[1] FastOpt, Hamburg, Germany, `http://www.FastOpt.com`
[2] Numerical Analysis, Product Line E1, Volkswagen AG, Wolfsburg

Key words: Automatic Differentiation, Optimisation, Adjoint Methods, Computational Fluid Dynamics, Automotive Design

Abstract. *We report on a proof-of-concept study for the application of an automatically generated adjoint Navier-Stokes code in the automotive design process. The design application consists in the optimisation of the topology of a duct for the cabin ventilation. The numerical model is a three-dimensional Navier-Stokes solver based on Griebel's NaSt2D. The design approach is based on a descent algorithm, which relies on the gradient of dissipated energy with respect to the duct topology. This gradient is provided by the adjoint of the solver, which is generated from the solver's source code by the automatic differentiation tool TAF. To render the solver TAF-compliant, a few initial modifications were necessary. Subsequent modifications of the TAF compliant solver such as a change of the objective function have been transferred to an updated adjoint in an automated process chain. The computational cost of a combined nominal and adjoint solve, i.e. of a function and gradient evaluation, is less than twice the cost of a nominal solve.*

1 DESIGN PROBLEM

The design problem consists in constructing a duct for the cabin ventilation, which has to fit within a prescribed three-dimensional installation space with fixed in- and outflows. The optimal topology of the duct minimises the flow's dissipated energy. The design problem is actually discrete: a grid cell in the installation space either belongs to the duct or not. To render the problem continuous, a three-dimensional porosity field is introduced, i.e. the dimension of the design space equals the number of grid points. A grid cell with porosity of 0 is part of the duct, whereas a cell with porosity of 1 is not. For details of this method, please refer to [1].

For simulation of the flow, a three-dimensional solver based on the Fortran 90 version of NaSt2D [2] has been prepared. It uses a finite difference method and an explicit time stepping scheme to solve the incompressible, transient Navier-Stokes equations.

The dissipative energy is minimised by a descent algorithm that relies on the availability of the gradient of the dissipative energy with respect to all design variables, i.e. with respect to the porosity in each grid cell of the mesh. For this proof-of-concept study computational domains of up to 100 grid cells in each spatial dimension have been used. This corresponds to a design space of around 10^6 dimensions, which renders a gradient approximation by finite differences of nominal solves prohibitively expensive. This leaves the designer with two choices. Firstly, they could reduce their design space (via splines etc) by ad hoc assumptions on the topology. This option is, however, at high risk of missing the optimum. The alternative is the efficient computation of the gradient by the adjoint of the solver.

2 AUTOMATIC DIFFERENTIATION

Automatic Differentiation (AD [3]) provides derivative code that is accurate up to numerical rounding error. The principle of AD is simple: The underlying function code is decomposed into elementary functions that correspond to individual operations in the code such as "+", " /", or intrinsics like "`EXP`". On the level of these elementary functions the derivative (local Jacobian matrix) is relatively easy to generate. According to the chain rule, the derivative of the underlying (composite) function then equals the product of all local Jacobian matrices. This product can be evaluated in any order. The forward mode (tangent code) preserves the order defined by the function code, while the reverse mode (adjoint code) reverses that order. The computational burden of a forward mode computation is proportional to the number of independent variables (inputs to the function), while in reverse mode it is proportional to the number of dependent (output) variables but independent of the number of independent variables. Hence, the reverse mode is ideally suited for optimisation problems as they can usually be formulated in terms of a scalar-valued objective function which may also account for constraints (in weak formulation).

3 TAF

Transformation of Algorithms in Fortran (TAF [4]) is an AD tool (for other AD-tools see `http://www.autodiff.org`). TAF transforms the Fortran 77-95 source code of an underlying function (nominal solver) to highly efficient first (adjoint and tangent) or higher order (e.g. Hessian) derivative code. Adjoint code requires intermediate values from the function evaluation (required values). By default TAF generates recomputations of these values. Alternatively TAF directives can be inserted into the function code to trigger generation of a flexible store/read scheme. For long time integrations TAF can generate memory/disk efficient adjoint code which uses flexible (linear) checkpointing schemes [5]. This adjoint code can even be furnished with a restart capability (divided adjoint [6]) such that the adjoint integration may be interrupted. For steady problems (iterative solvers) TAF can generate [7] an efficient alternative adjoint code variant as suggested by [8]. Parallelisation capabilities (OpenMP directives and MPI calls) of the function code can be transferred to the tangent and adjoint codes [6, 9]. In the case of MPI, however, hand coded adjoint interface routines have to be provided.

TAF's concepts and algorithms are being transferred to a new AD tool for handling of C(++) codes, Transformation of Algorithms in C++ (TAC++). The tool has been applied to five small codes (the largest of which comprises some 300 code lines, excluding comments), and generates corresponding tangent, adjoint, and Hessian codes in a fully automated procedure.

4 AD of SOLVER and DESIGN

Excluding comments, the solver contains about 3500 lines of Fortran 90 source code. For its use in an optimisation loop and compliance with TAF, the solver code was modified at a few places. The most important changes were the following: Pre- and post-processing were split off the main solver. The computation of the maximum velocities (for the step size control) has been slightly recoded: A `DO` loop with two exits was replaced by a `DO-WHILE` loop. 14 TAF directives were included to arrange storing/reading of required values or to support TAF's code analyses.

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 adjoints. The first adjoint version (standard adjoint) stores required values in each solver iteration on disk. It provides the exact gradient for steady and unsteady computations. The second version of the adjoint (iteration adjoint) expects convergence of the solver to a steady flow and stores this flow in memory.

The derivative codes have been verified by comparing derivative values against finite differences of nominal runs. Tangent and standard adjoint correspond to each other, and the finite differences have a relative error of less than 10^{-5}. For the iteration adjoint this is only the case for a sufficiently high number of iterations.

In an optimisation loop controlled by a simple descent algorithm, only five iterations are needed to find a topology for which the dissipated energy is reduced by about 20 %.

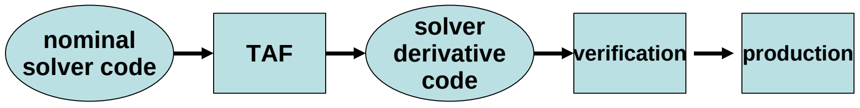

Figure 1: Process chain for update of derivative code after revision of nominal solver.

The performance of the derivative code has been tested on a Pentium 1.86GHz Linux-Notebook with 1GB of core memory. The Lahey-Fujitsu Fortran 95 compiler with options `-O3 --dbl`, i.e. full optimisation and with double precision, was used. For a test configuration of 15 × 15 × 15 grid cells, the computational cost of a nominal plus tangent solve is about 1.3 times the cost of a nominal solve. For both adjoints this ratio is about 1.8.

One big advantage of using AD for the generation of adjoint code is that extensions to the suite of objective functions can be made in a practically automatic fashion: in the present study, we added the flow uniformity as a second objective function (a subroutine of about 100 lines of Fortran). Within a turnaround time of less than 24 hours, the tangent and the two adjoint codes were updated by TAF and verified in an automated procedure.

5 CONCLUSIONS

The generation of efficient tangent and adjoint codes for topology optimisation and their successful application in the automotive design process were demonstrated. The usage of TAF allows to automate the maintenance of the derivative code and, hence, the update process of the entire system to new releases of the nominal solver. This in turn reduces the delay between model development and design applications, i.e. the overall efficiency of the design cycle can be increased.

REFERENCES

[1] Othmer, C., Grahs, T.: Approaches to fluid dynamic optimization in the car development process. In Schilling, R., Haase, W., Periaux, J., Baier, H., Bugeda, G., eds.: EUROGEN 2005, Evolutionary and Deterministic Methods for Design, Optimization and Control with Applications to Industrial and Societal Problems, Munich, Germany. (2005)

[2] Griebel, M., Dornseifer, T., Neunhoeffer, T.: Numerical Simulation in Fluid Dynamics, a Practical Introduction. SIAM, Philadelphia (1998)

[3] Griewank, A.: Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation. SIAM, Philadelphia (2000)

[4] Giering, R., Kaminski, T.: Recipes for Adjoint Code Construction. ACM Trans. Math. Software **24** (1998) 437–474

[5] Griewank, A.: Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation. Optimization Methods and Software **1** (1992) 35–54

[6] Heimbach, P., Hill, C., Giering, R.: Automatic generation of efficient adjoint code for a parallel Navier-Stokes solver. In Sloot, P.M.A., Tan, C.J.K., Dongarra, J.J., Hoekstra, A.G., eds.: Computational Science – ICCS 2002, Proceedings of the International Conference on Computational Science, Amsterdam, The Netherlands, April 21–24, 2002. Part II. Volume 2330 of Lecture Notes in Computer Science., Berlin, Springer (2002) 1019–1028

[7] Giering, R., Kaminski, T., Slawig, T.: Applying TAF to a Navier-Stokes solver that simulates an Euler flow around an airfoil. Future Generation Computer Systems **21** (2005) 1345–1355

[8] Christianson, B.: Reverse accumulation and implicit functions. Optimization Methods and Software **9** (1998) 307–322

[9] 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. Volume 50 of Lecture Notes in Computational Science and Engineering. Springer, New York, NY (2005) 273–282