GENERATING AND MAINTAINING HIGHLY EFFICIENT ADJOINT AND HESSIAN CODE FOR OPTIMISATION AND UNCERTAINTY ANALYSIS BY AUTOMATIC DIFFERENTIATION

Ralf Giering, Thomas Kaminski, Michael Voßbeck

FastOpt
Schanzenstr. 36, 20357 Hamburg, Germany
e-mail:info@FastOpt.com

Key words: Automatic differentiation, Design, Optimisation, Adjoint, Hessian, Gradient Methods, Computational Fluid Dynamics, Aerodynamics

Abstract. We introduce two tools for Automatic Differentiation (AD), TAF and TAC++. TAF transforms the Fortran 77-95 source code of an underlying function to highly efficient first (adjoint and tangent) or higher order (e.g. Hessian) derivative code. We highlight some TAF applications, e.g. for sensitivity analysis, optimisation, and uncertainty propagation, and describe TAF features that are essential for the efficiency of the respective derivative codes. TAC++ is becoming the C++ counterpart of TAF. We show an efficient adjoint of a CFD solver, which has been generated in a first TAC++ design study.

1 AUTOMATIC DIFFERENTIATION

In large scale optimisation, i.e. for problems that are formulated in terms of complex numerical codes and a large number of control variables, gradient algorithms constitute a most powerful, and often the only feasible approach. Their wider use has, however, been hampered by the difficulty of providing the necessary gradient (or even higher order derivative) information with sufficient accuracy.

Automatic Differentiation (AD [1]) 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 linear 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).

2 TAF and TAC++

Transformation of Algorithms in Fortran (TAF [2]) is an automatic differentiation tool (for other AD-tools see http://www.autodiff.org). TAF transforms the Fortran 77-95 source code of an underlying function to highly efficient first (adjoint and tangent) or higher order (e.g. Hessian) derivative code. For long time integrations TAF can generate memory/disk efficient adjoint code which uses flexible (linear) checkpointing schemes [3]. This adjoint code can even be furnished with a restart capability (divided adjoint [4]) such that the adjoint integration may be interrupted. For steady problems (iterative solvers) TAF can generate [5] an efficient alternative adjoint code variant as suggested by [6]. Parallelisation capabilities (OpenMP directives and MPI calls) of the function code can be transferred to the tangent linear and adjoint codes [4].

In the case of MPI, however, hand coded adjoint interface routines have to be provided. We give an overview on TAF applications such as sensitivity analysis, optimisation, and uncertainty propagation, to areas that include aerodynamics and mechanics.

Meanwhile TAF forms an integral part of a number of modelling systems, which allows to automate the maintenance of the derivative code and, hence, the update process of the entire system to new releases of the function code. This in turn reduces the gap between model development and (optimisation) applications, i.e. the frequency of the design cycle can be increased.

The design of Transformation of Algorithms in C++ (TAC++) is based on the same algorithms as TAF. We present an efficient adjoint code of a Roe flux solver (170 lines of C code) that has been generated automatically in a first design study.

REFERENCES


