Extension of fixed point PDE solvers for optimal design by one-shot method

With first applications to aerodynamic shape optimization

Nicolas R. Gauger*, — Andreas Griewank* — Jan Riehme*****

** Humboldt University Berlin, Department of Mathematics*
Unter den Linden 6, 10099 Berlin, Germany
nicolas.gauger@dlr.de, griewank@mathematik.hu-berlin.de
*** German Aerospace Center (DLR), IAS, 38108 Braunschweig, Germany*
**** RWTH Aachen, STCE, 52056 Aachen, Germany*
riehme@stce.rwth-aachen.de

ABSTRACT. *This paper concerns mathematical methods, algorithmic techniques and software tools for the transition from simulation to optimization. We focus in particular on applications in aerodynamics. The methodology is applicable to all areas of scientific computing, where large scale governing equations involving discretized PDEs are treated by custom made fixed point solvers. To exploit the domain specific experience and expertise invested in these simulation tools we propose to extend them in a semi-automated fashion. First they are augmented with an adjoint solvers to obtain (reduced) derivatives and then this sensitivity information is immediately used to determine optimization corrections.*

RÉSUMÉ. *Cet article concerne des méthodes mathématiques, de techniques algorithmiques et d'outils de programmation pour le passage de la simulation à l'optimisation. On s'intéresse plus particulièrement aux applications en aérodynamique. La méthodologie est applicable à tous les secteurs du calcul scientifique, pour lesquels des EDPs discrétisées sont traitées en utilisant des solveurs basés sur des méthodes usuelles de point fixe. En vue d'exploiter l'expérience et l'expertise acquises dans le domaine spécifique de ces outils de simulation, on propose leurs extensions de façon semi-automatique. Tout d'abord, on rajoute la résolution de l'adjoint afin d'obtenir les dérivées réduites. Cette information sur les sensibilités est immediatement utilisée pour déterminer les corrections d'optimisation.*

KEYWORDS: *automated optimal design, one-shot.*

MOTS-CLÉS : *conception optimale de forme, différentiation automatique, one-shot.*

DOI:10.3166/REMN.17.87-102

1. Common obstacles to optimization with PDEs

For many actual or potential optimization users the transition from existing simulation models and tools to optimal design is anything but simple. Often the prospect of redesigning and reimplementing their models to interface with a classical NLP (non-linear programming) package is so daunting that the idea of employing calculus based optimization methods is abandoned all together. Especially for discretizations of PDEs the specifications of Jacobian sparsity patterns and the provision of partial derivative values can be extremely laborious. Moreover due to sheer size and lack of structure that effort may not even lead to an efficient solver for the purpose of *system simulation, i.e.* the resolution of a nonlinear *state equation*

$$c(y,u) \;=\; 0 \quad \text{with} \quad c : Y \times U \to Y.$$

Here $u \in U$ is a design vector, which may be kept fixed as $c(y,u) = 0$ is solved for the corresponding *state vector* $y = y_*(u) \in Y$. In aerodynamics u may represent a parameterization of a wing shape, which determines together with appropriate free stream boundary conditions the flow field y around the wing. In climatological studies u is a vector of model parameters and y is a vector of prognostic variables, *i.e.* ocean and atmosphere flow velocities and temperature. In the right function space setting one may assume that the linearized operator $c_y \equiv \nabla_y c$ has a bounded inverse, but often the Jacobian obtained for a suitable discretization is so unwieldy that no Newton-like solver can be realized. In (Heinkenschloss *et al.*, 2001; Hazra *et al.*, 2005; Biros *et al.*, 2002) one finds frameworks for constrained optimization in cases where the linearized state equations can be solved reasonably rapidly.

Instead we address the situation where one has to make do with a fixed point iteration for solving $c(y,u) = 0$

$$y_{k+1} \;=\; G(y_k, u) \quad \text{for} \quad k = 0, 1, \ldots$$

which frequently may be interpreted as pseudo-time stepping on an underlying instationary version of the state equation.

For example, in aerodynamics, one can use quasi-unsteady formulations which are solved by explicit central finite volume schemes stabilized by artificial dissipation and Runge-Kutta time integration (Jameson *et al.*, 1981). These schemes are most efficient in combination with geometric multigrid (Jameson, 1986; Swanson *et al.*, 1997).

There is some steady progress in simulation models and of course computing power. Nevertheless we have to assume that in many application areas a single state equation solve to full accuracy takes several hours or even days on a single machine. Compared to the effort of gaining feasibility for given u in this way, the evaluation of an objective

$$f(y,u) : Y \times U \to \mathbb{R}^m,$$

which may represent a fitting functional or other performance indices is usually a cheap byproduct. Hence the transition from simulation to optimization may appear at

first quite simple. Consequently there are many software tools that implement assorted direct search strategies based on computing solutions y_k with $c(y_k, u_k) \approx 0$ and then $f(y_k, u_k)$ at a cloud of sampling points u_k in the design domain U.

Disregarding frequent claims of global convergence to global minima on nondifferentiable problems, one can expect that local minima will be approximately located by Nelder Mead type algorithms (Nelder *et al.*, 1964) if $c(y, u)$ and $f(y, u)$ are at least once continuously differentiable. Instead of the linear models on which Nelder Mead is based, one may of course fit other *surrogate objective* functions through the points evaluated at any stage. To construct a reasonable *response surface* of that kind or to approximate a single gradient by differences one needs at the very least $\dim(u)$ evaluations. Hence there is no hope to achieve what might be called the principle of bounded cost deterioration

$$\textbf{Cost Optimization} \sim \textbf{Cost Simulation}$$

for optimal design.

In the computational practice one tries very hard to reduce $\dim(u)$ by allowing only certain *principle mode* perturbations in the design domain. Nevertheless, completely derivative-free optimization calculations rarely involve fewer than a hundred function evaluations and the use of evolutionary algorithms that aim at locating global minima can easy lead to thousands if not millions of sample points. At a recent engineering conference several speakers reported of structural and aerodynamical design optimization studies that tied up hundreds of computers for several weeks. Similar brute force calculations are also standard in the German automotive industry, for example to optimize the placement of welding point with regards to the resulting crash test properties. As far as practical optimization with PDE based models is concerned this must be considered the state of the art.

Apart from the very important ease-of use issue two reasons are often advanced for the preference of evolutionary algorithms. First, their ability to climb out of local minima and hopefully to reach nearly globally minimal values. Especially on adaptively discretized and iteratively solved PDEs many of the local minima may be generated by 'roughness' of the actually computed function, which is also correctly seen as an obstacle to the calculation of derivatives. In the context of fixed point solvers this difficulty can be partly overcome by using the sequence

$$\textbf{discretize} \rightarrow \textbf{differentiate} \rightarrow \textbf{iterate}$$

rather than

$$\textbf{discretize} \rightarrow \textbf{iterate} \rightarrow \textbf{differentiate}.$$

As a negative consequence the derivatives calculated will no longer be strictly consistent with the underlying function values. This suggests that one might as well pull the differentiation all the way in front and prefer the sequence

$$\textbf{differentiate} \rightarrow \textbf{discretize} \rightarrow \textbf{iterate}.$$

We note that the popular debate whether **discretize** should preceed **optimize** or not makes little sense in the context of one-shot approaches. In any case on non-linear problems one should always distinguish between the processes **optimize** and **differentiate**, which is often synonymous with **linearize**. The second reason often given by users of evolutionary algorithms is the desire to perform multi-objective optimization, for which calculus based methods are widely considered inadequate. We content that this is a misconception and will pursue the approach of (Das *et al.*, 1998) to trace the Pareto (hyper) surface.

2. AD tools

The forward and reverse mode of algorithmic or automatic differentiation (AD) were already proposed in the mid-fifties and mid-sixties, respectively. The development of corresponding software tools started in earnest only in the mid-eighties and has yet to reach the professional level that would be appropriate for the complexity of the task at hand. This caveat concerns in particular the reverse, or adjoint mode, while forward differentiation represents no difficulty even when applied to the adaptive and iterative solvers of interest here. For example Bischof has differentiated the commercial CFD code FLUENT using ADIFOR 2.0 (Bischof *et al.*, 2001), Giering and Kaminski have differentiated the CFD code NSC2KE using TAF (Giering *et al.*, 2005) and Hascoet has applied TAPENADE for sonic boom reduction (Hascoët *et al.*, 2003).

In order to achieve bounded cost deterioration irrespective of the dimension of the design domain the reverse mode must be employed. The DLR codes FLOWer (Fortran) (Kroll *et al.*, 1999) and TAUij (C-Code, 2D version of TAUijk) (Heinrich, 2006) have been correctly differentiated in the reverse mode using TAF and ADOL-C (Schlenkrich *et al.*, 2006), respectively. However, in both cases there was a considerable effort in terms of man power. To a large extent that is due to the complicated information flow and extensive memory traffic generated by the reverse mode. In particular in the context of iterative solvers the easy, safe, and efficient use of the reverse mode still requires some improvements of the AD tools.

3. One-step one-shot optimization on fixed point solvers

The problem of augmenting fixed point solvers for PDEs with sensitivity and optimization calculation has been considered by the authors during the last few years (Griewank *et al.*, 2002; Griewank, 2006; Griewank *et al.*, 2005).

For a given objective $f(y,u)$ we require to fulfill the state equation $c(y,u) = P(y-G)$, while $G(y,u) = y - P^{-1}c$ is an iteration function and P an appropriate

preconditioner. We assume a uniform contraction rate $\|G_y\| \leq \rho < 1$ and define the shiftet Lagrangian function

$$\begin{aligned} N(y, \bar{y}, u) &:= G(y, u)^\top \bar{y} + f(y, u) \\ &= f(y, u) - \left(P^{-1}c\right)^\top \bar{y} + y^\top \bar{y} \, . \end{aligned}$$

Rather than first fully converging the primal state using

$$y_{k+1} = G(y_k, u_k) \quad \rightarrow \quad \text{primal feasibility at } y_*$$

and then fully converging the dual state applying

$$\bar{y}_{k+1} = N_y(y_k, \bar{y}_k, u_k) \quad \rightarrow \quad \text{dual feasibility at } \bar{y}_*$$

before finally performing an "outer" optimization loop

$$u_{k+1} = u_k - H_k^{-1} N_u(y_k, \bar{y}_k, u_k) \quad \rightarrow \quad \text{optimality at } u_* \ ,$$

we suggest an extended single-step one-shot iteration of the form

$$\begin{bmatrix} y_{k+1} \\ \bar{y}_{k+1} \\ \bar{u}_{k+1} \\ u_{k+1} \end{bmatrix} = \begin{bmatrix} G(y_k, u_k) \\ G_y(y_k, u_k)^\top \bar{y}_k + f_y(y_k, u_k)^\top \\ G_u(y_k, u_k)^\top \bar{y}_k + f_u(y_k, u_k)^\top \\ u_k - H_k^{-1} \bar{u}_{k+1} \end{bmatrix} = \begin{bmatrix} N_{\bar{y}}(y_k, u_k) \\ N_y(y_k, \bar{y}_k, u_k) \\ N_u(y_k, \bar{y}_k, u_k) \\ u_k - H_k^{-1} N_u(y_k, \bar{y}_k, u_k) \end{bmatrix} .$$

For computing the optimization correction $u_{k+1} - u_k$ one has to choose the symmetric positive definite matrix $H_k \succ 0$, which we will refer to as the *preconditioner* of the approximate reduced gradient $\bar{u}_k$. As discussed in (Griewank *et al.*, 2002) the $\bar{y}_k$ and $\bar{u}_k$ are not the exact adjoints of y_k and u_k but represent approximations with the same limit.

For simplicity we assume that the H_k converge to an ideal limit H_*, which must reflect the parameterization of the design domain U. The construction of a suitable H_* may be interpreted as the design of a feed back state controller based on the 'output' function N_u to stabilize the KKT (Karush-Kuhn-Tucker) point. The ramifications of results from the theory of discrete dynamical systems (see e.g. (Hinrichsen *et al.*, 2005)) on our nonlinear optimization tasks have not yet been examined but will be explored in the proposed project.

The gradient (N_y, N_u) and thus the adjoints $(\bar{y}_{k+1}, \bar{u}_{k+1})$ can be computed by algorithmic differentiation provided G and f are available as a "grey box", *i.e.* the source code of the solver is available and suited for the application of AD techniques. Alternatively, of course a user supplied adjoint solver can be utilized. By differentiating the step function G in the reverse mode we implement the $\mathbf{differentiate} \rightarrow \mathbf{iterate}$ strategy mentioned above. To obtain good sensitivity information it is important that the size of the adjoint discrepancy $N_y - \bar{y}$ be included into the overall stopping criterion. Assuming only that the largest eigenvalues of G_y are nondefective, it was

shown in (Griewank *et al.*, 2005) that for fixed design $u_k = u$ the adjoints $\bar{y}_k$ converge with the same R-factor as the y_k, but that the error ratio $\|\bar{y}_k - \bar{y}_*\| / \|y_k - y_*\|$ grows proportionally to the iteration counter k. Second order adjoints were found to lag behind by a factor of k^2, which might delay the construction of the preconditioner H_* somewhat.

Provided we have convergence of the extended iteration, the Jacobian at a limit point $(y_*, \bar{y}_*, u_*)$ takes the form

$$\begin{aligned} \hat{J}_* &= \left. \frac{\partial\,(y_{k+1}, \bar{y}_{k+1}, u_{k+1})}{\partial\,(y_k, \bar{y}_k, u_k)} \right|_{(y_*, \bar{y}_*, u_*)} \\ &= \begin{bmatrix} G_y & 0 & G_u \\ N_{yy} & G_y^\top & N_{yu} \\ -H_*^{-1} N_{uy} & -H_*^{-1} G_u^\top & \mathbb{1} - H_*^{-1} N_{uu} \end{bmatrix}. \end{aligned}$$

We found that the imposition of contractivity with respect to a norm of the diagonal form $\left(\|.\|_Y^2 + \|.\|_{\bar{Y}}^2 + \|.\|_U^2\right)^{0.5}$ leads to very tight conditions on H_*. Hence we prefer a spectral analysis of the extended fixed point iteration which places less restrictions on the choice of H_*.

A sufficient condition for contractivity in some norm of the extended fixed point iteration is that the spectral radius $\hat{\rho}$ of $\hat{J}_*$ be smaller than 1. Whenever we can define H such that

$$\frac{1-\hat{\rho}}{1-\rho} \approx \frac{\log(\hat{\rho})}{\log(\rho)} < \text{const}$$

over a range of problems, the goal of **bounded retardation** may be considered attained.

It was shown in (Griewank, 2006) that the eigenvalues of $\hat{J}_*$ are the zeros of the equation

$$\det((\lambda - 1)H_* + H(\lambda)) = 0 \qquad ,$$

where

$$H(\lambda) := \left[-G_u^\top (G_y^\top - \lambda \mathbb{1})^{-1} \quad \mathbb{1}\right] N_{xx} \begin{bmatrix} -(G_y - \lambda \mathbb{1})^{-1} G_u \\ \mathbb{1} \end{bmatrix} \qquad [1]$$

and

$$N_{xx} := \nabla^2_{(y,u)} N = \begin{bmatrix} N_{yy} & N_{yu} \\ N_{uy} & N_{uu} \end{bmatrix}.$$

Here N_{xx} is the Hessian of the Lagrangian and $H(1)$ its projection onto the feasible tangent space, which must be at least semi-definite at local minimizers.

To exclude real eigenvalues $\lambda \geq 1$ it is necessary and sufficient that H_* is positive definite, which was already assumed anyway. To exclude real eigenvalues $\lambda \leq -1$ we derived in (Griewank, 2006) the more interesting necessary condition that

$$H_* \succ H(\lambda)/(1-\lambda) \quad \text{if} \quad \lambda \leq -1 \quad .$$

On a simple 2D test problem with positive definite N_{xx} it was found in (Griewank, 2006) that the choice $H_* = H(-1)$ worked quite well. However, on closer examination one finds that in general checking the above condition on any finite subinterval of $(-\infty, -1]$ is not enough to ensure that H_* satisfies the test for all λ. Since we have so far also disregarded the likely possibility of complex eigenvalues, it becomes clear that finding the H_* for which $\hat{\rho}$ is less than 1 (let alone minimal) requires further investigation.

There is some indication that it may sometimes be advantageous to alternate between various versions of H or even to modify the original solver, so that it has more desirable spectral properties. For example if all eigenvalues of G_y were real one could eliminate alternating modes in the convergence of G by considering $y_{k+1} = G^2(y_k, u_k) \equiv G(G(y_k, u_k), u_k))$ as basic step. A cheaper modification then also yielding a strictly positive spectrum would be to take the average $y_{k+1} = [y_k + G(y_k, u_k)]/2$. More general one might consider a nonlinear Chebychev iteration of the form

$$y_{k+1} \;=\; \sum_{j=0}^{q} \alpha_j G^j(y_k, u) \quad \text{with} \quad \sum_{j=0}^{q} \alpha_j = 1 \quad .$$

Possibly, once could then also apply cyclically $q > 0$ different preconditioners during the substeps, but that is pure speculation at this stage.

When H_* is defined as $H(1)$ or $H(-1)$ as defined in Equation [1] it can be computed as a family of $\dim(U)$ second order adjoint vectors by another level of algorithmic differentiation. As we will see this requires in particular the calculation of a $\dim(U)\dim(Y)$ matrix of $\dim(U)$ directions in the state space. For more general definitions of H_* one may have to estimate certain parameters from the current iteration behavior, so that many practical issues remain to be resolved.

Whereas near the optimal and feasible solution point the preconditioner ensures an asymptotic contraction rate, a line search has to be applied in the earlier stage of the iteration to enforce convergence. Preliminary examinations suggest the augmented Lagrangian

$$p(y, \bar{y}, u) := \frac{\alpha}{2}\|G(y,u) - y\|_2^2 + \frac{\beta}{2}\|N_y(y, \bar{y}, u) - \bar{y}\|_2^2 + N - \bar{y}^\top y \quad ,$$

which is a smooth, exact penalty function, (see (Pillo, 1994)). The gradient of the merit function p can be expressed in the factorized form

$$\begin{bmatrix} \nabla_y p \\ \nabla_{\bar{y}} p \\ \nabla_u p \end{bmatrix} \;=\; -\begin{bmatrix} \alpha(\mathbb{1} - G_y)^\top & -\beta N_{yy} - \mathbb{1} & 0 \\ -\mathbb{1} & \beta(\mathbb{1} - G_y) & 0 \\ -\alpha G_u^\top & -\beta N_{uy} & H \end{bmatrix} \begin{bmatrix} G - y \\ N_y - \bar{y} \\ -H^{-1} N_u \end{bmatrix} .$$

When the weights α and β are selected such that the block matrix on the right is nonsingular, it follows from the general contractivity assumption on G that the KKT points of the original problem correspond exactly to stationary points of the augmented Lagrangian $p(y, \bar{y}, u)$. Moreover we find that the step increment

$$s := s(y, \bar{y}, u) = \left[G - y, \quad N_y - \bar{y}, \quad -H^{-1}N_u\right]^\top$$

of the extended iteration yields descent on the merit function when α, β and H are chosen adequately. More specifically we obtain the directional derivative as the quadratic form

$$\begin{bmatrix}\nabla_y p\\ \nabla_{\bar{y}} p\\ \nabla_u p\end{bmatrix}^\top s = -s^\top \begin{bmatrix}\alpha(\mathbb{1} - \overline{G}_y) & -\frac{\beta}{2}N_{yy} - \mathbb{1} & -\frac{\alpha}{2}G_u\\ -\frac{\beta}{2}N_{yy} - \mathbb{1} & \beta(\mathbb{1} - \overline{G}_y) & -\frac{\beta}{2}N_{yu}\\ -\frac{\alpha}{2}G_u & -\frac{\beta}{2}N_{yu} & H\end{bmatrix} s \quad ,$$

where $\overline{G}_y = \frac{1}{2}(G_y + G_y^\top)$ denotes the symmetrization of the matrix G_y. For suitable choices of α, β and H the smallest eigenvalue of the symmetrized block matrix on the right can be bounded away from zero. Then one can design a line search procedure for determining γ_k such that the iterates

$$\begin{aligned}[y_{k+1}, \bar{y}_{k+1}, u_{k+1}] &= (1 - \gamma_k)\,[y_k, \bar{y}_k, u_k]\\ &+ \gamma_k \left[G\left(y_k, u_k\right), N_y\left(y_k, \bar{y}_k, u_k\right), -H^{-1}\bar{u}_{k+1}\right]\end{aligned}$$

must converge to a KKT point. Since N_y and thus p involves already first derivatives of G and f it will be a challenge to design an efficient line-search that does not require the evaluation of second derivatives, although that would also be possible in principle.

4. Implementation at the software level

In this section we discuss how the original 'primal state' iteration $y_{k+1} = G(y_k, u_k)$ can be augmented by the 'dual state' iteration $\bar{y}_{k+1} = N_y(y_k, \bar{y}_k, u_k)$. Mathematically this is easily done and the cheap gradient result of automatic differentiation guarantees that the operations count for evaluating the combined gradient $\nabla N = (N_y, N_u)$ is no more than five times that for evaluating the pair (G, f) and thus N itself. This favorable complexity result presupposes the ability to store all intermediate quantities generated during one evaluation of (G, f). It should be emphasized that this 'logging' of intermediates is normally only required for one evaluation of (G, f) at a given argument (y, u), rather than the sequence of iterations $y_{k+1} = G(y_k, u_k)$. Typically the resulting memory requirement is also a small multiple of that for evaluating $G(y, u)$ itself. Hence there are theoretically no serious obstacles to generating an efficient adjoint iteration. However, from a software point of view the situation is not quiet as straightforward.

First, let us consider the ideal situation, where one state iteration and one objective evaluation can be wrapped into a user supplied subroutine call of the form:

input:	where:
step $(\overset{\downarrow}{u}, \overset{\downarrow}{y}, \underset{\downarrow}{z}, \underset{\downarrow}{f})$	$z = G(u, y)$
output:	$f = f(u, y)$

In other words the call to the routine step must compute from the current design vector u and the current state vector y the next, improved state vector z and the objective function value f. The arrows indicate in an obvious way, which variables are input and which variables are output. Then the pseudo-time iteration can be performed by a simple, conceptional loop of the following form:

```
init(u, z);        y = 0
while(‖y − z‖ ≫ 0)
    y = z
    step(u, y, z, f)
use(z, f)
```

Of course the while condition is just conceptual and must be replaced by something more specif in a real implementation. Provided that step and all routines called by it are available as source code one may use a suitable AD tool to generate an adjoint call of the form

$$\mathsf{bstep}(\underset{\downarrow}{\overset{(0)}{bu}}, \overset{\downarrow}{u}, \underset{\downarrow}{\overset{(0)}{by}}, \overset{\downarrow}{y}, \overset{\downarrow}{bz}, \underset{\downarrow}{z}, \overset{\downarrow}{bf}, \underset{\downarrow}{f}) \quad ,$$

where mathematically

$$bu = G_u(y,u)^\top bz + f_u(y,u)^\top bf \quad \text{and} \quad by = G_y(y,u)^\top bz + f_y(y,u)^\top bf.$$

It should be noted that for the dual variables bu, by, by and bf (where the prefix b represents bar) the information flow is opposite to that for the underlying primal variables u, y, z, f. The dual variable bf is usually set to 1 in optimization. Other positive values may be used to effect a scaling of the dual variables. The zeros on top of bu and by indicate that these vectors must be initialized to zero because adjoint subroutines deal with dual variables in an incremental fashion. The adjoint code can now be employed in an augmented loop of the following form:

Coupled basic and adjoint iteration:

```
init(u, z, by);    y = 0;   bz = 0
while(‖z − y‖ + ‖by − bz‖ ≫ 0)
    y = z;  bz = by;  bu = 0;  by = 0;  bf = 1          [2]
    bstep(bu, u, by, y, bz, z, bf, f)
use(z, f, bu, by)
```

Again the stopping criterion is only conceptual. Before discussing the problems one may face in generating this first order loop let us note what happens if one differentiates the routine bstep once more in the forward mode. Then one obtains a second order adjoint routine dbstep of the following form:

$$\mathsf{dbstep}(\overset{(0)}{\underset{\downarrow}{bu}}, \underset{\downarrow}{u}, \underset{\downarrow}{du}, \overset{(0)}{\underset{\downarrow}{dbu}}, \overset{(0)}{\underset{\downarrow}{by}}, \underset{\downarrow}{y}, \underset{\downarrow}{dy}, \overset{(0)}{\underset{\downarrow}{dby}}, \underset{\downarrow}{bz}, \underset{\downarrow}{z}, \underset{\downarrow}{dz}, \underset{\downarrow}{dbz}, \underset{\downarrow}{bf}, \underset{\downarrow}{f}, \underset{\downarrow}{df})$$

Here the prefix d represents *dot*, *i.e.* differentiation in the direction defined by the input tangents du and dy. In contrast to 'barring' the 'dotting' of variables does not alter the information flow, including the initialization requirement compared to the original variables. Correspondingly the pseudo-time loop now takes the somewhat more complicated form:

$\mathsf{init}(u, z, by, dz, dby); \quad y = 0; \quad bz = 0; \quad dby = 0$
$\mathsf{while}(\|z - y\| + \|by - bz\| + \|dz - dy\| + \|dby - dbz\| \gg 0)$
$\quad y = z;\ bz = by;\ bu = 0;\ by = 0;\ dbu = 0;\ dby = 0$
$\quad dy = dz/\lambda;\ dbz = dby/\lambda$
$\quad \mathsf{dbstep}(bu, u, du, dbu, by, y, dy, dby, bz, z, dz, dbz, bf, f, df)$
$\mathsf{use}(z, f, bu, by, dbu, dby)$

When du is set to a Cartesian basis vector in the design space U the resulting second order adjoint vector dbu can be shown to converge to the corresponding column of the projected Hessian $H(\lambda)$ provided $|\lambda| = 1$. If one generates dbstep in the so-called vector mode such that 'dotting' means generating $n = dim(u)$ directional derivatives then one obtains the whole projected Hessian $H(\lambda)$ as defined in [1]. Naturally the spatial and temporal complexity then also grows by the factor n. Even though this second level of vector differentiation looks quite complicated it is comparatively easy to implement, once the first reverse differentiation process has been completed successful. That may prove quite difficult for the following diverse reasons.

First there is unfortunately in the AD community no agreement on how exactly the adjoint routine bstep should handle the primal variables, here in particular the outputs z and f. For our purposes it would be most convenient if z and f have on return from bstep exactly the same values as on return from step, so that the functionality of the latter is in fact incorporated into the former. In the overloading based tool ADOL-C the routines gradient and reverse have this combined functionality, albeit without generating anything like a problem specific adjoint bstep. The situation is rather different for source transformation tools like TAPENADE, which we have used here to extend a step represented by a collection of Fortran codes.

Irrespective of the software technology, adjoint routines like bstep always perform two successive sweeps, the first propagates primal values forward, just like step but with logging, and the second, reverse sweep, propagates dual values backward. Since the latter must restore or recompute the original primal values in opposite order it is in

some sense natural that the output variables z and f should have on exit from bstep again the same values as on entry. Generally, we may label an adjoint routine *primally consistent* if it transforms the primal variable exactly as the underlying undifferentiated routine and *primally constant* if on exit all primal variables have been restored to their values on entry. When the given adjoint bstep is primally constant we can follow it up by a call to step itself, such that the combination of both has the effect of a primally consistent adjoint. Obviously this simple trick to turn a primally constant adjoint into a primally consistent adjoint incurres a lot redundant calculations as the original code is effectively executed twice.

In our implementation we replaced in the coupled iteration [2] the call to the TAPENADE generated bstep by the pair of successive calls

$$\begin{array}{ll} \text{(ysave,usave)} \leftarrow \text{(y,u)} ; & \mathsf{bstep}(bu, u, by, y, bz, z, bf, f) \\ \text{(y,u)} \leftarrow \text{(ysave,usave)} ; & \mathsf{step}(u, y, z, f) \end{array} \qquad [3]$$

Unfortunately, we found no convenient way around this redundancy. Furthermore, we had to add the save and restore operations on (u, y) for the following reason. Some AD tools generate adjoints that are neither primally consistent nor primally constant, but leave the values of the primal variables (here u, y, z and f) on exit in an undefined state. This undesirable uncertainty is sometimes justified by potential gains in efficiency for calculations with significant linear contents. In our view the chance of significant reductions in complexity is rather slight and certainly does not outweigh the danger of wrong results and extra confusion in the mind of potential users. Naturally, we recommend that all AD tools should provide in future both, the primally consistent and the primally constant adjoint mode. While we need here the former the latter is also extremely useful, for example for reversing call trees in the so-called joined mode. The implementation of primal consistency in adjoints merely require a save of all primal variables at the end of the forward, logging sweep, and their restoration upon completion of the reverse sweep.

Unfortunately, we encountered another serious implementation problem. It has less to do with the modes provided by AD tools but arises from the nature of aerodynamics codes. The problem is that what we called here the state and represented by a single vector y is in reality a heterogeneuos collection of scalars, vectors, matrices and even tensors. The values of these quantities are often interrelated, as some may for example be fluxes representing products of velocities and densities. Then the state representation is to some extent redundant. Moreover some of these state components may reside in static local variables or common blocks rather than being explicit calling parameters. Finally, and most importantly some or all of these state components are immediately updated in place at each iteration. Thus the tidy separation of old state vector y and new state vector z as well as the corresponding adjoint vectors by and bz, which we assumed above may well be almost impossible to realize. We therefore look for an implementation that only needs to touch explicitily the primal variables (u, f) and their duals $(\bar{u}, \bar{f})$.

Suppose we represent the motley crew of state variables that occur as calling parameters by the list $y1, \ldots, yN$ so that the supplied subroutine takes the form:

input:	where:
$\mathsf{step}(\overset{\downarrow}{u}, \overset{\downarrow}{y1}, \ldots, \overset{\downarrow}{yN}, f)$	$y1, \ldots, yN = G(u, y1, \ldots, yN)$
output:	$f = f(u, y1, \ldots, yN)$,

and the basic iteration is given by:

init$(u, y1, \ldots, yN)$
while(y1,...,yN vary)
    step$(u, y1, \ldots, yN, f)$
use$(y1, \ldots, yN, f)$

Correspondingly we obtain an adjoint of the form:

$$\mathsf{bstep}(\overset{(0)}{bu}, \overset{\downarrow}{u}, \overset{\downarrow}{by1}, \ldots, \overset{\downarrow}{byN}, \overset{\downarrow}{y1}, \ldots, \overset{\downarrow}{yN}, \overset{\downarrow}{bf}, f)$$

Provided bstep is primally consistent (which might be ensured again by the doubling up trick discussed above) we now obtain the coupled iteration:

init$(u, y1, \ldots, yN, by1, \ldots, byN)$;
while$(y1, \ldots, yN$ or $by1, ..., byN$ vary$)$ do
    $bu = 0; bf = 1$
    bstep$(bu, u, by1, \ldots, byN, y1, \ldots, yN, bf, f)$
use$(f, y1, \ldots, yN, bu)$

To obtain asymptotically the correct results one does not really need to initialize the adjoints $by1, \ldots, byN$, which may start from zero or some arbitrary value. Hence, there is no longer a need to even find out what the state components are, provided they are suitably initialized somewhere in the original code. All one has to do now is to reset bu and bf at each iteration to the zero vector and the scalar one, respectively. Again due to the uncertain state of primal variables in TAPENADE generated adjoints we had to save all of them before the call to bstep and then restore them before a subsequent call to step just as in [3].

5. Application to aerodynamic shape optimization

As a first test case we optimize the glide ratio of an inviscid NACA0012 airfoil under transonic flight conditions ($\alpha = 2°$, $Ma = 0.8$). The CFD solver taken for this

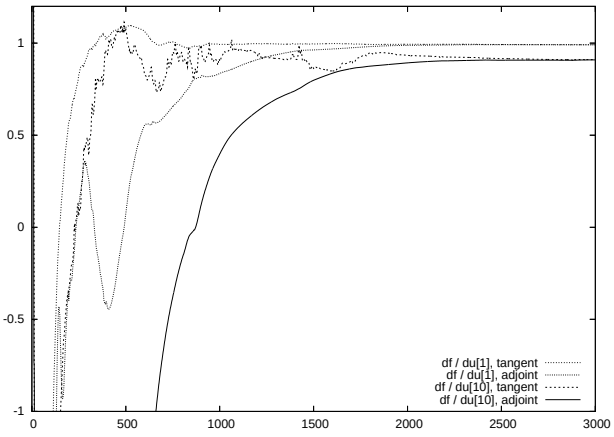

Figure 1. *Glide ratio simulation of the NACA0012 airfoil ($\alpha = 2^\circ$, $Ma = 0.8$). Design variables: 20 Hicks-Henne coefficients. The figure illustrates the sensitivities, evaluated by TAPENADE in forward as well as adjoint mode, with respect to two of the Hicks-Henne functions for the initial airfoil*

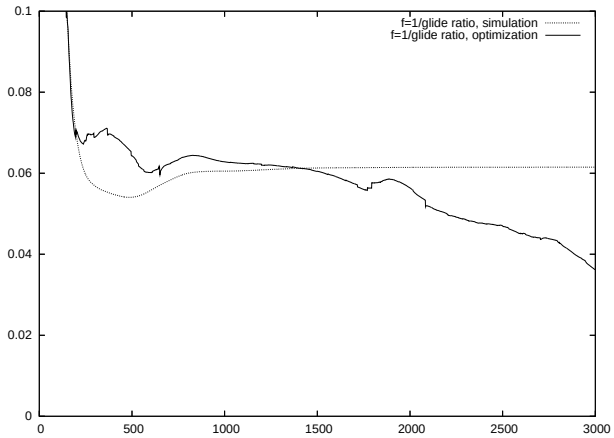

Figure 2. *Glide ratio optimization of the NACA0012 airfoil ($\alpha = 2^\circ$, $Ma = 0.8$). Design variables: 20 Hicks-Henne coefficients. The figure illustrates the convergence history of the single-step one-shot (optimization) procedure in comparison to a simulation run of the initial airfoil*

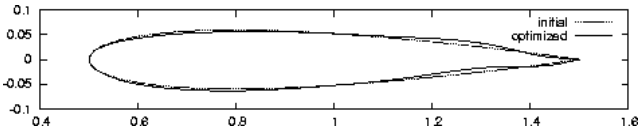

Figure 3. *Glide ratio optimization of the NACA0012 airfoil ($\alpha = 2°$, $Ma = 0.8$). Design variables: 20 Hicks-Henne coefficients. The figure illustrates the initial as well as the optimized airfoil*

case is a simplified 2D version of the FLOWer code (Kroll *et al.*, 1999), using an explicit central finite volume scheme stabilized by artificial dissipation and Runge-Kutta time integration (Jameson *et al.*, 1981) acting on structured meshes. The thickness of the airfoil is kept constant. This is ensured by a camberline-thickness decomposition of the airfoil, while the thickness distribution never changes. For the parameterization of the camberline we choose 20 coefficients of Hicks-Henne functions (Hicks *et al.*, 1978), which are just added to the camberline and therefore define their deformation. The changes in the surface mesh of the resulting airfoil geometry are propagated into the mesh by applying Reuther's mesh deformation technique (Jameson *et al.*, 1994).

All the above described routines are coded in Fortran and finally put together into one compilation unit. Each pseudo-time step is effected by a call of the form $\mathsf{step}(u, y1...yN, f)$ as described in Section 4. All the derivatives needed and explained in Section 4 are generated by the AD tool TAPENADE.

For validation Figure 1 illustrates the sensitivities with respect to two of the Hicks-Henne functions, evaluated for the initial airfoil. As expected, the derivatives evaluated in forward as well as adjoint mode are asymptotically consistent. We prefer the reverse mode because it yields all gradient components simultaneously. The second derivatives are provided by the forward on reverse approach. Figure 2 illustrates the convergence history of the single-step one-shot procedure in comparison to a simulation run of the initial airfoil. Finally, Figure 3 shows the initial as well as the optimized airfoil.

6. References

Biros G., Ghattas O., " Inexactness Issues in the Lagrange-Newton-Krylov-Schur Method for PDE-Constrained optimization", *Springer's Lecture Notes in Computational Science and Engineering*, 2002.

Bischof C. H., Bücker H. M., Lang B., Rasch A., " Automatic Differentiation of the FLUENT CFD program: procedure and first results", *Working Note of the Institute for Scientific Computing RWTH-CS-SC-01-22, Aachen University of Technology, Aachen*, 2001.

Das I., Dennis J., " Normal boundary intersection: a new method for generating Pareto optimal points in multicriteria optimization problems", *SIAM J. on Optimization*, vol. 8, n° 3, p. 631-657, 1998.

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*, vol. 21, n° 8, p. 1345-1355, 2005.

Griewank A., " Projected Hessians for Preconditioning in One-Step One-Shot Design Optimization", *Large Scale Nonlinear Optimization*, vol. 83, p. 151-171, 2006.

Griewank A., Faure C., " Reduced Functions, Gradients and Hessians from Fixed Point Iteration for State Equations", *Numerical Algorithms*, vol. 30, n° 2, p. 113-139, 2002.

Griewank A., Kressner D., " Time-lag in Derivative Convergence for Fixed Point Iterations", *Revue ARIMA*, vol. Numéro spécial CARI'04, p. 87-102, 2005.

Hascoët L., Vázquez M., Dervieux A., " Automatic Differentiation for Optimum Design, Applied to Sonic Boom Reduction", *in* V.Kumar *et al.* (ed.), *Proceedings of the International Conference on Computational Science and its Applications, ICCSA'03, Montreal, Canada*, LNCS 2668, Springer, p. 85-94, 2003.

Hazra S., Schulz V., " Simultaneous pseudo-timestepping for state constrained optimization problems in aerodynamics", *in* L. Biegler, O. Ghattas, M. Heinkenschloss, D. Keyes, B. van Bloemen Waanders (eds), *Large-scale PDE-Constrained Optimization: Towards Real-time and Online PDE-Constrained Optimization*, SIAM series on Computational Science and Engineering, 2005.

Heinkenschloss M., Vicente L. N., " Analysis of Inexact Trust-Region Interior-Point Algorithms", *SIAM J. Optim.*,p. 282-302, 2001.

Heinrich R., " Implementation and usage of structured algorithms within an unstructured CFD-code", *Notes on Numerical Fluid Mechanics and Multidisciplinary Design*, vol. 92, 2006.

Hicks R. M., Henne P. A., " Wing design by numerical optimization", *Journal of Aircraft*, vol. 15, p. 407-412, 1978.

Hinrichsen D., Pritchard A. J., *Mathematical Systems Theory I*, vol. 48 of *Texts in Applied Mathematics*, Springer-Verlag, 2005.

Jameson A., " Multigrid algorithms for compressible flow calculations", *in* W. Hackbusch, U. Trottenberg (eds), *Multigrid Methods II*, vol. 1228 of *Lecture Notes in Mathematics*, Springer, p. 166-201, 1986.

Jameson A., Reuther J., " Control theory based airfoil design using the Euler equations", *AIAA Proceedings 94-4272-CP*, 1994.

Jameson A., Schmidt W., Turkel E., " Numerical Solutions of the Euler Equation by Finite Volume Methods Using Runge-Kutta Time-Stepping Schemes", *AIAA 81-1259*, 1981.

Kroll N., Eisfeld B., Bleecke H. M., " FLOWer", *Notes on Numerical Fluid Mechanics*, vol. 71, p. 58-68, 1999.

Nelder J. A., Mead R. A., " A simplex method for function minimisation", *Comput. J.*, vol. 7, p. 308-313, 1964.

Pillo G. D., " Exact penalty methods", *in* E. Spedicato (ed.), *Algorithms for continuous optimization: the state of the art*, Kluwer Academic Publishers, p. 209-253, 1994.

Schlenkrich S., Walther A., Gauger N., Heinrich R., " Differentiating fixed point iterations with ADOL-C: Gradient calculation for fluid dynamics", *Proceedings of HPSC*, Hanoi, March 6-10, 2006.

Swanson R. C., Turkel E., Multistage Scheme with Multigrid for Euler and Navier-Stokes Equations (Components and Analysis), Technical Report n° 3631, NASA, 1997.