CCES Seminar WS 12/13

The one-shot approach for optimization and control of steady and unsteady flows

Lisa Kusch

RWTH Aachen

Supervisor: Prof. Dr. Nicolas R. Gauger,
Computational Mathematics Group,
Department of Mathematics and
Center for Computational Engineering Science,
RWTH Aachen University

Contents

1 Introduction

The present paper was developed for the CES seminar in the Master program of Computational Engineering Science at RWTH Aachen University. It deals with the one-shot approach for shape optimization as presented in the main reference [7] by Özkaya and Gauger.

A lot of tasks in the field of engineering involve optimization problems or optimal control problems with underlying partial differential equations (PDEs). Usually, as in the case of computational fluid dynamics, these PDE constraints are computationally expensive. Therefore it is often made use of adjoint approaches that guarantee an evaluation of derivatives that is independent of the number of design variables. The one-shot approach, in comparison to classical optimization methods, is very efficient, as the PDEs are solved simultaneously with the optimization problem.

Özkaya and Gauger describe the one-shot method and its application to the shape optimization of an airfoil. It is made use of the reverse mode of algorithmic differentiation for calculating the discrete adjoints.

The aim of the present paper is to reuse the work on the one-shot method and to apply algorithmic differentiation to the recent version of the ELAN code.

The paper is structured as follows: In Section 2 the general shape optimization or optimal control problem is presented together with the idea of adjoint methods. Section 3 gives the central ideas of the steady and the unsteady one-shot approach. The solver ELAN that is used for the flow simulation and an idea for the parameterization of the shape are given in Section 4. The basics of algorithmic differentiation and the applied tool TAPENADE are presented in Section 5. Finally, Section 6 provides the results of applying the one-shot approach to the shape optimization of an airfoil that are presented in the main reference and own results of the algorithmic differentiation.

2 Fundamentals

2.1 Shape Optimization Problem

Shape optimization aims at finding a shape of the considered geometry that corresponds to the optimal solution of a specific task. The general shape optimization problem can be posed as

$$\min_{u \in D} f(y(u), u) \tag{1}$$

with the vector of design variables u and the vector of state variables $y \in Y$. The design space D is for example given as

$$D = \{u \in U : c(y(u), u) = 0\}, \tag{2}$$

where $c(y, u)$ is the underlying non-linear partial differential equation. In the present work it stands for the fact that y has to fulfill the steady, incompressible Reynolds-averaged Navier-Stokes (RANS) equation with the k-ω turbulence model for a two-dimensional flow field. In the following it is assumed that $\frac{\partial c(y,u)}{\partial y}$ is regular such that one can define a unique state variable y for each design variable u. The objective function f is identified with the drag coefficient C_d. The design variable u parametrizes the shape of the airfoil.

The Lagrange function L corresponding to the optimization problem (1) with the design space 2 is

$$L(\bar{y}, y, u) = f(y, u) + c(y, u)^T \bar{y} \tag{3}$$

with the the vector of Lagrange multipliers $\bar{y}$, that can also be identified as the vector of adjoint variables. A Karush-Kuhn-Tucker (KKT) optimality point $(y^*, \bar{y}^*, u^*)$ has to fulfill the conditions

$$\begin{aligned} c(y^*, u^*) &= 0 \\ \nabla_y f(y^*, u^*) + c_y(y^*, u^*)^T \bar{y}^* &= 0 \\ \nabla_u f(y^*, u^*) + c_u(y^*, u^*)^T \bar{y}^* &= 0. \end{aligned} \tag{4}$$

In the listed order it is referred to the equations as the state equation, the adjoint equation and the design equation. The gradient of $\tilde{f} := f(y(u), u)$ that is used in a gradient-based optimization algorithm is given by

$$\nabla_u \tilde{f} = \nabla_u f(y, u) + \left(\frac{\partial y}{\partial u}\right)^T \nabla_y f(y, u). \tag{5}$$

It contains the sensitivity gradient $\frac{\partial y}{\partial u}$ that cannot be computed directly. It could be approximated with the help of finite differences or the forward mode of algorithmic differentiation. Both approaches are computationally expensive as the evaluation of the sensitivity gradient depends on the number of design variables. Therefore, it is convenient to turn to adjoint approaches.

The gradient can be expressed with the help of the equations (4) as

$$\nabla \tilde{f} = \nabla_u f(y, u) + c_u(y, u)^T \bar{y}. \tag{6}$$

Now the task is to find the vector of adjoint variables $\bar{y}$.

2.2 Adjoint Approaches

There exist two different adjoint approaches. These are the continuous and the discrete adjoint approach. In the continuous adjoint approach the aim is to derive the continuous adjoint equation for the optimization problem and discretize it afterwards. The adjoint equation corresponding to the state equation is derived analytically from the given model. This is computationally efficient but it can be very complex. As a result, the implementation of the adjoint equation is error-prone. The continuous adjoint approach is limited to the frozen eddy viscosity assumption because of the fact that many turbulence models cannot be differentiated analytically.

The discrete adjoint is built by first discretizing the state equation and then finding the discrete adjoint variables, that are consistent to the primal variables. This can for example be done with the help of the reverse mode of algorithmic differentiation. The use of algorithmic differentiation eliminates the problem of treating turbulence models. A disadvantage involving the use of the reverse mode is that it requires a lot of memory and is therefore less computationally efficient. In this context, the discrete adjoint approach is applied and a discrete problem is derived.

In the following one assumes that the PDE constraints are solved with the fixed point iteration $y_{k+1} = G(y_k, u)$ with a contraction rate $||G_y|| \leq \rho < 1$ that converges to $y^* = G(y^*, u)$. This fixed point iteration is also referred to as the primal iteration. The conditions for a KKT point of the discrete problem are given as

$$\begin{aligned} y^* &= G(y^*, u) \\ \bar{y}^* = N_y(y^*, \bar{y}^*, u^*)^T &= \nabla_y f(y^*, u^*)^T + G_y(y^*, u^*)^T \bar{y}^* \\ 0 = N_u(y^*, \bar{y}^*, u^*)^T &= f_u(y^*, u^*)^T + G_u(y^*, u^*)^T \bar{y}^* \end{aligned} \tag{7}$$

where $N(y, \bar{y}, u) := f(y, u) + G(y, u)^T \bar{y}$ is the shifted Lagrangian that appears in the original Lagrangian $L(y, \bar{y}, u) = N(y, \bar{y}, u) - y^T \bar{y}$.

2.3 Additional Constraints

An additional constraint to the original drag minimization problem could be set for the lift coefficient C_l. One wants to minimize the drag coefficient while keeping the lift coefficient above a prescribed value $C_{l,min}$ which means that the additional inequality constraint is given as

$$C_l \geq C_{l,max}. \tag{8}$$

The resulting constrained optimization problem can be treated with a penalty multiplier approach, as done in [7]. The inequality constraint is added to the objective function with a penalty parameter λ. This results in the optimization problem

$$\min_{u \in D} C_d(u, y) + \lambda(C_{l,min} - C_l(u, y)). \tag{9}$$

As proposed in [7], the penalty parameter can be chosen iteratively as $\lambda_{k+1} = \lambda_k(1+\nu(C_{l,min} - C_l(u_k, y_k)))$ with $\nu > 0$ and $\lambda_0 = \frac{||\nabla C_d||}{||\nabla(C_{l,min} - C_l(u_k, y_k))||}$. The penalty parameter should be set to zero if the constraint is fulfilled.

3 The One-Shot Method

3.1 The Steady One-Shot Method

The conventional approach for solving problem (7) is to fully converge the fixed point iteration of the state equation and use the result for fully converging a fixed point iteration for the adjoint equation that is $\bar{y}_{k+1} = N_y(y, \bar{y}_k, u)^T$. Afterwards the gradient N_u is computed and used for updating the design variable. Then, the whole process is repeated until the optimization algorithm has fulfilled a given convergence criterion.

In the one-shot approach one iterates for the state variable, the adjoint variable and the design variable simultaneously in a coupled iteration step that is given by

$$\begin{aligned} y_{k+1} &= G(y_k, u_k) \\ \bar{y}_{k+1} &= N_y(y_k, \bar{y}_k, u_k) \\ u_{k+1} &= u_k - B_k^{-1} N_u(y_k, \bar{y}_k, u_k). \end{aligned} \tag{10}$$

The primal iteration together with the adjoint iteration are referred to as piggy-back iteration and converge with the same asymptotic rate to the fixed point $(y^*, \bar{y}^*)$ with a certain time lag of the adjoint variable.

The one-shot method was invented by Ta'asan et al. ([8],[9]). It can be applied to any kind of optimization problem or optimal control problem that is constrained by PDEs that are solved with a contractive fixed point iteration. The one-shot approach can be applied in combination with continuous and discrete adjoints. The combination of the one-shot approach with continuous adjoints is for example done in [6].

The design iteration corresponds to a gradient-based optimization procedure with a preconditioner B_k. The choice of a suitable preconditioner can

guarantee the convergence of the one-shot method. Hamdi and Griewank ([3], [4]) show for a sufficiently large preconditioner descent on the exact penalty function of the doubly augmented Lagrangian type that is

$$L^a(y,\bar{y},u) = \frac{\alpha}{2}||G(y,u)-y||^2+\frac{\beta}{2}||N_y(y,\bar{y},u)^T-\bar{y}||^2+N(y,\bar{y},u)-\bar{y}^T y \quad (11)$$

with the two strictly positive weighting coefficients α and β fulfilling the so-called correspondance condition

$$\alpha\beta(I-G_y)^T(I-G_y) \succ I+\beta N_{yy}. \quad (12)$$

The gradient of L^a is given as

$$\begin{pmatrix} \nabla_y L^a \\ \nabla_{\bar{y}} L^a \\ \nabla_u L^a \end{pmatrix} = -\begin{pmatrix} \alpha(I-G_y)^T & -I-\beta N_{yy} & 0 \\ -I & \beta(I-G_y) & 0 \\ -\alpha G_u^T & -\beta N_{yu}^T & B \end{pmatrix} \begin{pmatrix} G-y \\ N_y-\bar{y}^T \\ -B^{-1}N_u^T \end{pmatrix} \quad (13)$$
$$= -Ms(y,\bar{y},u).$$

The increment vector $s=(\Delta y,\Delta\bar{y},\Delta u)^T$, that is at the same time the one-shot increment vector, is proven in [3] to yield descent on L^a for all large positive preconditioners B if the condition

$$\alpha\beta\Delta\bar{G}_y \succ (I+\frac{\beta}{2}N_{yy})(\Delta\bar{G}_y)^{-1}(I+\frac{\beta}{2}N_{yy}) \quad (14)$$

with $\Delta\bar{G}_y=\frac{1}{2}(I-G_y+(I-G_y)^T)$ is fulfilled. If α and β satisfy

$$\sqrt{\alpha\beta}(1-\rho) > 1+\frac{\beta}{2}||N_{yy}||, \quad (15)$$

condition (12) and (14) are automatically fulfilled. Thus, the increment vector s yields descent on L^a and because of the fact that s is the one-shot increment vector also on L for sufficiently large B. Furthermore, it has been proven by Hamdi and Griewank in [4] that any preconditioner that satisfies

$$B \succcurlyeq B_0 := \frac{1}{\sigma}(\alpha G_u^T G_u+\beta N_{yu}^T N_{yu}) \quad (16)$$

with

$$\sigma = 1-\rho-\frac{(1+\frac{||N_{yy}||}{2}\beta)^2}{\alpha\beta(1-\rho)} \quad (17)$$

yields descent on L^a.

The matrix B_0 is strongly related to the Hessian $\nabla_{uu}L^a$ with respect to the design. A preconditioner is derived by considering the optimization problem

$$\min_{\Delta u} L^a(y + \Delta y, \bar{y} + \Delta\bar{y}, u + \Delta u) \tag{18}$$

and finding B from the condition $\Delta u = -B^{-1}N_u^T$. When using a sequential quadratic programming approach for problem (18) one obtains the minimization problem

$$\min_{\Delta u} s^T \nabla L^a + \frac{1}{2} s^T \nabla^2 L^a s. \tag{19}$$

which is equivalent to

$$\begin{aligned} \min_{\Delta u} \Delta u^T (\nabla_u L^a + \nabla_{uy} L^a \Delta y + \nabla_{u\bar{y}} L^a \Delta\bar{y}) + \frac{1}{2}\Delta u^T \nabla_{uu} L^a \Delta u \\ \approx \Delta u^T \nabla_u L^a (y + \Delta y, \bar{y} + \Delta\bar{y}, u) + \frac{1}{2}\Delta u^T \nabla_{uu} L^a \Delta u. \end{aligned} \tag{20}$$

The solution to (20) is given by $\Delta u = -\nabla_{uu}^{-1} L^a(y, \bar{y}, u)\nabla_u L^a(y + \Delta y, \bar{y} + \Delta\bar{y}, u)$. One identifies $B \approx \nabla_{uu}L^a$ and gets $B = \alpha G_u^T G_u + \beta N_{yu}^T N_{yu} + N_{uu}$ which is equivalent to the Hessian $\nabla_{uu}L^a$ if primal and dual feasibility are satisfied. One chooses

$$B = \frac{1}{\sigma}(\alpha G_u^T G_u + \beta N_{yu}^T N_{yu} + N_{uu}) \tag{21}$$

such that condition (16) is fulfilled for $N_{uu} \succcurlyeq 0$.

B is not computed exactly, but the inverse of the Hessian is approximated by a BFGS update. Since one has $B \approx \nabla_{uu}L^a$

$$B\Delta u \approx \nabla_u L^a(y, \bar{y}, u + \Delta u) - \nabla_u L^a(y, \bar{y}, u) \tag{22}$$

is fulfilled and thus, if one identifies H with the approximated inverse of B,

$$H_{k+1} R_k = \Delta u_k \tag{23}$$

with $R_k := \nabla_u L^a(y_k, \bar{y}_k, u_k + \Delta u_k) - \nabla_u L^a(y_k, \bar{y}_k, u_k)$. H_k is updated with the BFGS update formula

$$H_{k+1} = (I - \frac{\Delta u_k R_k^T}{R_k^T \Delta u_k}) H_k (I - \frac{\Delta u_k R_k^T}{R_k^T \Delta u_k}) + \frac{\Delta u_k \Delta u_k^T}{R_k^T \Delta u_k}. \tag{24}$$

It is important to apply this update only if the positive definitiveness of H is maintained which means $R_k^T \Delta u_k > 0$. If this is not the case, it is convenient to set $B = I$.

3.2 The Unsteady One-Shot Method

An optimization problem might also have an unsteady PDE constraint that is given by

$$\frac{\partial y(t)}{\partial t} + c(y(t), u) = 0 \ \forall t \in [0, T] \qquad (25)$$
$$y(0) = y_0.$$

As a result, the original objective function f of (1) depends on the time t and has to be averaged. The new objective function is given by

$$F := \frac{1}{T} \int_0^T f(y(t), u) dt. \qquad (26)$$

The spatial discretization and the temporal discretization with $y^i \approx y(i\Delta t)$ for $i = 0, ..., N$ of the PDE constraint result for example in the implicit scheme

$$\frac{y^i - y^{i-1}}{\Delta t} + c_h(y^i, u) = 0 \qquad (27)$$

for all $i = 1, ..., N$.

As before, one assumes that there exists a fixed point solver G for solving the PDE constraint that iterates k and converges to a pseudo-steady-state solution y_*^i for each time step i such that

$$y_{k+1}^i = G(y_k^i, y_*^{i-1}, u) \overset{k\to\infty}{\to} y_*^i = G(y_*^i, y_*^{i-1}, u) \qquad (28)$$

for all $i = 1, ..., N$ with the contractivity property $||G_{y^i}(y^i, y^{i-1}, u)|| \leq \rho < 1$ for all points of interest.

The discrete optimization problem reads

$$\min_{u \in D} \frac{1}{N} \sum_{i=1}^{N} f(y^i, u) \qquad (29)$$
$$D = \{u \in U : y^i = G(y^i, y^{i-1}, u) \ \forall i = 1, ..., N\}$$

and the corresponding Lagrange function is given by

$$L = \sum_{i=1}^{N} \left(f(y^i, u) + (y^i - G(y^i, y^{i-1}, u))^T \bar{y}^i \right). \qquad (30)$$

The conditions for a KKT point are

$$
\begin{aligned}
y_*^i &= G(y_*^i, y_*^{i-1}, u_*) \ \forall i = 1, ..., N \\
\bar{y}_*^i &= \frac{1}{N} f_{y^i}(y_*^i, u_*) + (G_{y^i}(y_*^i, y_*^{i-1}, u_*))^T \bar{y}_*^i \\
&+ (G_{y^i}(y_*^{i+1}, y_*^i, u_*))^T \bar{y}_*^{i+1} \ \forall i = N-1, ..., 1 \\
\bar{y}^N &= \frac{1}{N} f_{y^N}(y_*^N, u_*) + (G_{y^N}(y_*^N, y_*^{N-1}, u_*))^T \bar{y}_*^N \\
0 = L_u &= \sum_{i=1}^{N} \frac{1}{N} f_u(y_*^i, u_*) + (G_u(y_*^i, y_*^{i-1}, u_*))^T \bar{y}_*^i
\end{aligned}
\tag{31}
$$

The fixed point iteration for the adjoint is written as $\bar{y}_*^i = A(\bar{y}_*^i, \bar{y}_*^{i+1}, y_*^{i-1}, y_*^i, y_*^{i+1}, u_*)$. The classical approach to solve the optimization problem is to iterate the optimization loop l and in each iteration step

- $\forall i = 1, ..., N$: iterate k to solve primal equation $y_{k+1}^i = G(y_k^i, y_*^{i-1}, u_l)$,
- $\forall i = N, ..., 1$: iterate k to solve adjoint equation $\bar{y}_{k+1}^i = A(\bar{y}_k^i, \bar{y}_*^{i+1}, y_*^{i-1}, y_*^i, y_*^{i+1}, u_l)$,
- update design variable $u_{l+1} = u_l - \sigma B^{-1} L_u(y_*, \bar{y}_*, u_l)$.

This results in three nested loops. When using the unsteady one-shot approach one exchanges the time loop i and the k-loop. In the resulting one-shot approach one iterates k to solve in each iteration step

- $\forall i = 1, ..., N$: $y_{k+1}^i = G(y_k^i, y_{k+1}^{i-1}, u_k)$,
- $\forall i = N, ..., 1$: $\bar{y}_{k+1}^i = A(\bar{y}_k^i, \bar{y}_{k+1}^{i+1}, y_{k+1}^{i-1}, y_k^i, y_k^{i+1}, u_l)$,
- $u_{k+1} = u_k - \sigma B^{-1} L_u(y_k, \bar{y}_k, u_k)$.

It is shown in [2] that the mapping H that maps the vector $y_k = (y_k^1, ..., y_k^N)$ is a contractive fixed point iteration that converges to the unsteady solution of the PDE constraint such that all strategies of the steady one-shot method can be transfered to the unsteady one-shot approach.

4 Flow Simulation and Shape Parameterization

4.1 Flow Simulation with ELAN solver

The used solver ELAN is the in-house solver of the Technical University Berlin and is based on the PhD thesis of Xue ([10]). It is written in Fortran 77. The governing equations that are treated by the solver ELAN are the incompressible Reynolds-averaged Navier-Stokes (RANS) equations with turbulence modeling. Incompressible flow solvers are usually used for subsonic flow problems as they cannot treat shocks. The incompressible Navier-Stokes equations can be written in tensor notation without any body forces as

$$\frac{\partial \rho u_i}{\partial x_i} = 0, \tag{32}$$

$$\frac{\partial \rho u_i}{\partial t} + \frac{\partial \rho u_i u_j}{\partial x_i} - \frac{\partial}{\partial x_i}\left((\mu_l)\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)\right) + \frac{\partial p}{\partial x_j} = 0. \tag{33}$$

Note that x_i stands for the spatial coordinates and u_i for the velocity components. The equations contain the pressure p and the density ρ. The first equation is the equation of mass conservation, the following ones conserve momentum for $j = 1, 2, 3$. The RANS equations are derived by a Reynolds averaging

$$\Phi(x_i, t) = \bar{\Phi}(x_i) + \Phi'(x_i, t) \tag{34}$$

that decomposes the velocity components into a time-averaged part $\bar{\Phi}$ and its fluctuation in time Φ'. The resulting RANS model is given by

$$\frac{\partial \rho \bar{u}_i}{\partial x_i} = 0, \tag{35}$$

$$\frac{\partial \rho \bar{u}_i}{\partial t} + \frac{\partial \rho \bar{u}_i \bar{u}_j}{\partial x_i} - \frac{\partial}{\partial x_i}\left(\mu\left(\frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i} - \frac{2}{3}\frac{\partial \bar{u}_k}{\partial x_k}\delta_{ij}\right)\right) + \frac{\partial(\bar{p} + \rho k)}{\partial x_j} = 0. \tag{36}$$

The viscosity coefficient $\mu = \mu_l + \mu_t$ is split into the molecular dynamic viscosity μ_l and the turbulent viscosity μ_t. For the $k - \omega$ turbulence model the turbulent viscosity is modeled as

$$\mu_t = \rho \frac{k}{\omega} \tag{37}$$

with a partial differential equation for the turbulent kinetic energy k and the specific dissipation ω.

The solver ELAN solves the incompressible, steady RANS equation with a finite volume approach. The pressure-velocity coupling is done with the help of the SIMPLE algorithm. Looking at the correction step for the velocity components and the pressure, the resulting scheme can be interpreted as a fix point solver with the iteration procedure $y_{k+1} = G(y_k, u)$ where G describe the correction rule for the flow variables. Features of the ELAN solver are the use of multi-blocks and the possibility to work in parallel processor mode with Message Passing Interface (MPI) communication. The solver can also treat the unsteady RANS equation.

4.2 Shape Parametrization

There exist different approaches for creating the design variables. Özkaya and Gauger [7] apply the so-called free node parametrization. In this approach the design variables are the positions of all grid points that are located on the shape of the airfoil. The advantage of this approach is that there is no need to differentiate a geometric modeling tool. Additionally, one gets the highest possible degree of freedom for the optimization of the shape. A disadvantage of free-node parametrization is that there is a high risk for producing uneven and wavy shapes. In [7], this problem can is overcome by applying Sobolev smoothing to the reduced gradient that is used for updating the design.

5 Algorithmic Differentiation

As already explained above, it is advantageous to use the reverse mode of algorithmic differentiation (Griewank and Walther, [1]) to calculate the needed derivatives N_y and N_{yu} if the BFGS update is used. A feature of algorithmic differentiation is that the function gradients can be calculated accurately to working precision which is advantageous for the convergence of the used optimization method. There exist two different approaches for the implementation of algorithmic differentiation. These are the application of operator overloading and the source transformation approach. Operator overloading does not require as much code transformation as source transformation but it is not well optimized by the compiler and very expensive in storage when using the reverse mode of algorithmic differentiation. In this work it is made use of algorithmic differentiation by source transformation with the tool TAPENADE from INRIA Sophia-Antipolis ([5]). TAPENADE can deal with FORTRAN and C code. The original code is augmented with its

derivative. This is done by dividing up the program into elementary operations for which the analytical derivatives are known. These are propagated via the chain rule for differentiation. There exist two different modes for algorithmic differentiation. When differentiating a function $F : \mathbb{R}^n \rightarrow \mathbb{R}^k$ the tangent-linear mode will result in an evaluation

$$y_d = \nabla F(x) x_d,\ y_d \in \mathbb{R}^k,\ x_d \in \mathbb{R}^n. \tag{38}$$

To obtain ∇F, x_d has to be chosen as $x_d = e_i$ for $i = 1, ..., n$. The resulting computational costs increase linearly with n. The adjoint mode results in

$$x_b = \nabla F(x)^T y_b,\ y_b \in \mathbb{R}^k,\ x_b \in \mathbb{R}^n. \tag{39}$$

In this case the computational costs for obtaining ∇F increase linearly with k. As the shape optimization problem is large in the number of design variables n, it is made use of the reverse mode. The reverse mode consists of a forward sweep and a reverse sweep. The forward sweep is composed of the original algorithm and the saving of variables on a stack that are later needed in the reverse sweep. In the reverse sweep the program statements are differentiated in reverse order. This means that each call of the adjoint routine calculates the original output and the needed derivatives. As a result, the use reverse mode of algorithmic differentiation requires only one run of the augmented code in each iteration of the one-shot approach. TAPENADE generates a new source code for each subroutine that implements the augmented code. The letter d that occurs for variables and routines indicates the forward mode while the letter b indicates the reverse mode. The gradient N_{yu} demands for differentiating twice. It can be made use of the reverse over tangent mode which means that the forward differentiated subroutine is differentiated again by using the reverse mode.

6 Results

6.1 Shape Optimization of Airfoil

In their paper Özkaya and Gauger ([7]) show numerical results obtained from applying the one-shot approach to the shape optimization of a NACA4412 profile with an angle of attack of 5°. The one-shot method converges successfully to an optimum shape of the airfoil. The optimization history is shown in Figure 6.1.

The one-shot approach turns out to be efficient as it needs only a number of three times as much of coupled iterations than the number of iterations needed for calculating a flow solution.

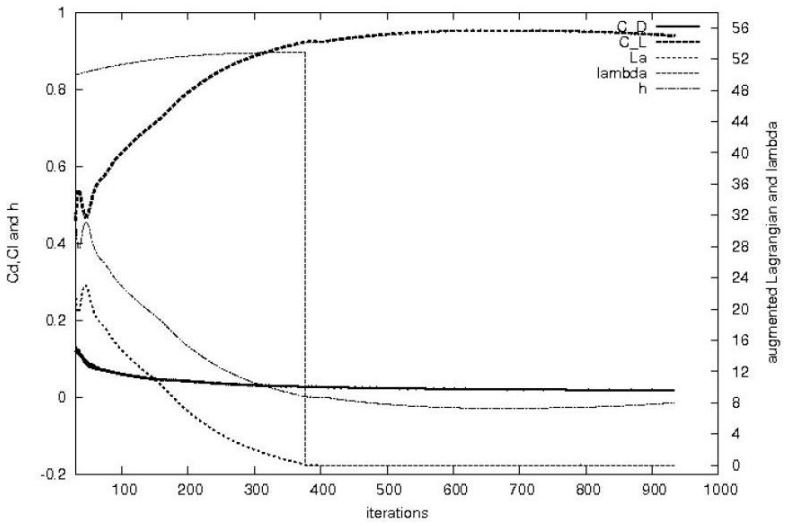

Figure 6.1: History of Optimization (taken from [7])

6.2 Using and Testing TAPENADE

A task of the seminar work was to differentiate the latest version of the ELAN code with TAPENADE and validate the calculated derivatives. The differentiated code is tested with a NACA 4412 profile with slits. The angle of attack of the flow is 20 degrees and the Reynolds number is 10^6. The profile is shown in Figure 6.2.

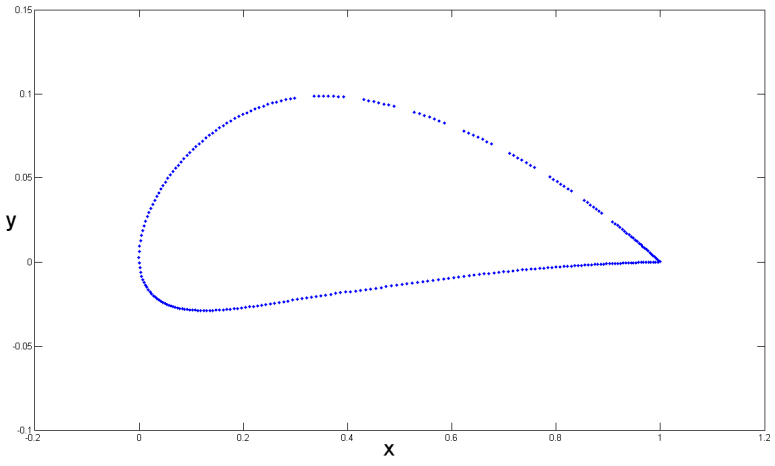

Figure 6.2: Profile of Test Case

The test case runs on eight processors that communicate via MPI.
The subroutine that is responsible for the outer iterations of the SIMPLE

algorithm is called `MGSOLVE`. It calls for example the subroutine `CALUVW` for solving the momentum balance, the subroutine `CALCP` for solving the pressure correction equation and the subroutines that are responsible for the turbulence model. The drag coefficient C_d shall be differentiated with respect to the design variables that are the spatial coordinates called `x` and `y`. The input to TAPENADE is the dependent output variable C_d and the independent input variables `x` and `y` with respect to which C_d is differentiated. The user has to provide the main subroutine `MGSOLVE` and all underlying subroutines that shall be differentiated by TAPENADE. TAPENADE recognizes the dependencies of the subroutines. Until now, TAPENADE cannot deal with MPI communication. The MPI commands in forward and reverse mode have to be adjusted by the user.

The test case needs 600 outer iterations to converge with an error of $4 \cdot 10^{-2}$. The derivatives that are obtained with the forward mode and the reverse mode of algorithmic differentiation are tested on 10 points on the shape of the airfoil. These points are marked with crosses in Figure 6.3.

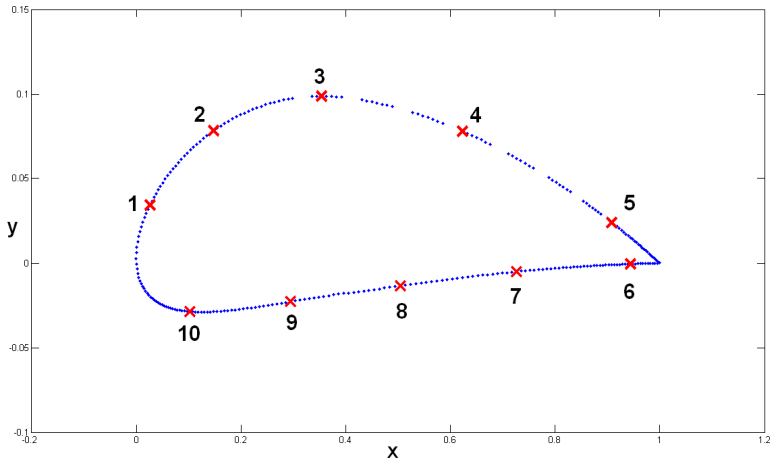

Figure 6.3: Points on Profile

The forward mode of algorithmic differentiation is validated with the help of finite differences. The derivative of C_d with respect to y is approximated with the finite difference formula

$$\frac{\partial C_d}{\partial y} \approx \frac{C_d(y + h \cdot y) - C_d(y)}{h \cdot y}.$$

The step size parameter h is chosen as $h = 10^{-8}$. The flow solution is calculated for each perturbed value of y.

	Finite Differences	Forward Mode	Reverse Mode
1	23.0056974716413	23.0054276597268	23.0054276597304
2	10.4676286015349	10.4674474595968	10.4674474594959
3	3.88053970223180	3.88049932759306	3.88049932758693
4	-2.29003882333091	-2.29004278781264	-2.29004278778940
5	-2.41020237346647	-2.41015086538022	-2.41015086491081
6	-1.92376728931625	-1.92635672856333	-1.92635672856339
7	-2.39357500996972	-2.39342832081042	-2.39342832054581
8	-10.2184037272716	-10.2183011127713	-10.2183011128433
9	-21.1215289411818	-21.1213253599733	-21.1213253645688
10	-44.1726074764843	-44.1719451427174408	-44.1719451437600

Table 6.1: Derivatives Obtained with Finite Differences and Forward Mode

The results when applying the forward mode are obtained by setting the tangent of y that in the code is referred to as yd to 1.0. Then the flow solution is calculated and the derivative with respect to y is stored in the tangent of C_d.

The calculations are done for each of the 10 points on the shape.

For the reverse mode one has to set the adjoint of C_d to 1.0 and calculate the flow solution once. Each gradient information can now be obtained from the adjoints of y.

The results of the gradients calculated with finite differences, the forward mode and the reverse mode are shown in Table 6.2.

The deviations of the finite difference results from the results of the forward mode are marked with a red color. They result from the inaccuracy of the finite difference formula. The results of the forward and the reverse mode match at least upon to the tenth digit, which is a good validation for the values of the reverse mode.

7 Conclusion

In this paper, the main reference by Özkaya and Gauger was reused. The one-shot approach was presented together with the use of discrete adjoints obtained from algorithmic differentiation. The one-shot approach needs much smaller numerical effort than a conventional optimization procedure. In the main reference the one-shot approach was applied for the shape optimization of a NACA airfoil. The results in the main reference show that it needs only three times as much iterations as the iterations needed for calculating a flow solution. The tool TAPENADE for algorithmic differentiation was success-

fully applied to the recent version of the ELAN code in forward and reverse mode. The results can be used in the future for implementing a one-shot approach.

References

[1] A. Griewank, A. Walther. Evaluating derivatives: principles and techniques of algorithmic differentiation. SIAM, Philadelphia, 2008.

[2] S. Günther, N. R. Gauger, Q. Wang. Extension of the One-Shot Method for Optimal Control with Unsteady PDEs. Submitted for EUROGEN 2013.

[3] A. Hamdi, A. Griewank. Properties of an augmented Lagrangian for design optimization. *Optimization Methods and Software* 25(4): 645-664, 2010.

[4] A. Hamdi, A. Griewank. Reduced Quasi-Newton Method for Simultaneous Design and Optimization. *Computational Optimization and Applications* 49(3):521-548, 2011.

[5] L. Hascoet, V. Pascual. Tapenade 2.1 user's guide. Technical Report 0300, INRIA, 2004.

[6] S. B. Hazra, V. Schulz, J. Brezillon, N. R. Gauger. Aerodynamic shape optimization using simultaneous pseudo-timestepping. *Journal of Computational Physics* 204:46-64, 2004.

[7] E. Özkaya, N. R. Gauger. Automatic transition from simulation to one-shot shape optimization with Navier-Stokes equations. *GAMM-Mitt.* 33(2):133-147, 2010.

[8] S. Ta'asan. One-shot methods for optimal control of distributed parameter systems I: Finite dimensional control. *ICASE 91-2*, 1991.

[9] S. Ta'asan, G. Kuruvila, M. D. Salas. Aerodynamic design and optimization in one shot. *AIAA 92-0025*, 1992.

[10] L. Xue. Entwicklung eines effizienten parallelen Lösungsalgorithmus zur dreidimensionalen Simulation komplexer turbulenter Strömungen. PhD thesis, TU Berlin, 1998.