11'th Annual AeSI CFD Symposium, 11-12 August 2009, Bangalore

Airfoil shape optimization using adjoint method and automatic differentiation

Praveen. C
TIFR Center for Applicable Mathematics
Post Bag No. 6503, Bangalore 560065, India
Email: praveen@math.tifrbng.res.in

The adjoint method for shape optimization in aerodynamics is becoming a powerful tool with the development of automatic differentiation tools for the generation of adjoint solvers. In this work we construct an efficient adjoint solver for a complex, multi-block, finite-volume CFD code. The developed tools are applied to solve 2-D airfoil shape optimization problems and several different optimization strategies are compared for their efficiency and effectiveness.

Keywords: Finite volume method, adjoint approach, automatic differentiation, shape optimization,

1 Introduction

The adjoint method is a powerful tool for the optimization of systems governed by partial differential equations whose numerical solution is computationally expensive. It allows the efficient and accurate computation of gradient of a cost function that depends on many design parameters and also on the solution of an expensive state equation like the Euler or Navier-Stokes equations. The use of adjoint equations for optimization in fluid mechanics goes back to the work of Pironneau [5]. Later, Jameson [3] used adjoint equations in a control theory framework for wing design using full potential, Euler and Navier-Stokes equations, which were solved using Computational Fluid Dynamics (CFD) techniques. These works make use of the *continuous adjoint* approach wherein the adjoint partial differential equations and boundary conditions are first derived and then discretized using a suitable numerical scheme. In contrast, several other researchers have used the *discrete adjoint* approach [1] in which the adjoint equations are derived for the finite-volume discretized form of the Euler/Navier-Stokes equations. Early works made use of hand differentiation to develop the adjoint solver, which can be a very cumbersome and error-prone process. Moreover maintaining the code requires continuous work since the CFD code may be modified quite often due to addition of improvements and bug fixes.

In this work we develop the adjoint solver starting from a flow solver for the Euler equations on structured, multi-block grids using finite volume method. The automatic differentiation tool TAPENADE [8] is used for differentiating the flow solver and construct the adjoint solver. The

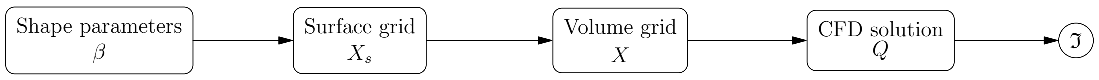

Figure 1: Steps to compute cost function

2-D shapes are parameterized using Hicks-Henne functions and the grid is deformed using radial basis function interpolation. The developed optimization methodology is applied for several 2-D inviscid shape optimization problems. A simple steepest descent method is compared with more sophisticated methods like quasi-Newton, sequential quadratic programming and primal-dual interior point methods.

2 Shape optimization framework

Let $\Im(\beta, Q)$ be a cost function which depends on certain design variables β representing the shape, and the flow solution Q which satisfies the steady Euler/Navier-Stokes equations, $R(\beta, Q) = 0$. We want to solve the following constrained optimization problem

$$\min_{\beta} \Im(\beta, Q), \quad \text{subject to} \quad R(\beta, Q) = 0 \tag{1}$$

The computation of the cost function involves three major steps as indicated in Fig. 1. (1) The parameters β which are usually part of a CAD system, define the shape of the flow domain and a surface grid X_s has to be generated on the shape. This is fairly easy for 2-D problems but can be very involved for 3-D problems. (2) Once a surface grid is available, the computational domain has to be discretized by generating a volume grid X. (3) The volume grid X is used in a CFD code to compute the flow solution Q from which the desired cost function is computed. By X_s we denote the coordinates of all the grid points that lie on the shape to be optimized and X denotes the coordinates of all the points in the volume grid.

In order to compute the gradient of the cost function with respect to the design variables β, we have to differentiate all the components in the above process. If we regenerate the grid X every time, then we would have to differentiate the grid generator which may not be possible due to lack of availability of source code or due to non-differentiability of the grid generation procedure. Hence we deform the grid corresponding to some reference shape (usually taken as the starting shape) to obtain the grid for the current shape. Using chain rule of differentiation we have

$$\frac{\mathrm{d}\Im}{\mathrm{d}\beta} = \frac{\mathrm{d}\Im}{\mathrm{d}X} \frac{\mathrm{d}X}{\mathrm{d}X_s} \frac{\mathrm{d}X_s}{\mathrm{d}\beta} \tag{2}$$

In the above equation, $\frac{\mathrm{d}\Im}{\mathrm{d}X}$ is computed from the adjoint flow solution, $\frac{\mathrm{d}X}{\mathrm{d}X_s}$ is computed by differening the grid deformation procedure and $\frac{\mathrm{d}X_s}{\mathrm{d}\beta}$ is computed by differentiating the CAD system. To reduce the computational cost, the transpose of the above equation is used which corresponds to the efficient adjoint approach to be explained in the next sections; the grid deformation and shape parameterization programs are differentiated by hand since they are all linear operations. Once the gradient is computed, it is used to compute a descent direction which is used to update the design variables in an optimization loop.

3 Flow solution method

We consider inviscid compressible flows governed by the Euler equations of gas dynamics. These equations can be written in conservation form using body fitted coordinates as

$$\frac{\partial \hat{Q}}{\partial t} + \frac{\partial \hat{F}}{\partial \xi} + \frac{\partial \hat{G}}{\partial \eta} = 0 \tag{3}$$

where $\hat{Q}$ is the vector of dependent variables and $(\hat{F}, \hat{G})$ are the flux vectors. The semi-discrete finite volume approximation of Eq. (3) is given by

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}\hat{Q}_{i,j} &+ [\hat{F}\nabla\xi/J]_{i+1/2,j} - [\hat{F}\nabla\xi/J]_{i-1/2,j} \\ &+ [\hat{G}\nabla\eta/J]_{i,j+1/2} - [\hat{G}\nabla\eta/J]_{i,j-1/2} = 0\end{aligned} \tag{4}$$

In the above equation, the interfacial fluxes are computed using the approximate Riemann solver of Roe [7]. Second order spatial accuracy is attained by using the MUSCL scheme of van Leer. The variables extrapolated are the primitive quantities ρ, u, v, p together with Koren limiter [4] to avoid spurious oscillations in the neighbourhood of a discontinuity. Since we are only interested in steady state problems, the conservation law is integrated in time until a steady solution is reached. The CFL condition restricts the time step that can be used in a stable manner, leading to large number of iterations to reach steady state. To accelerate convergence to steady state, an Euler implicit time integration scheme is used together with diagonal approximation [6] of jacobians using the eigensystems of the flux jacobians.

4 Adjoint method

The adjoint method is an efficient approach to compute derivatives of cost functional $\mathfrak{I}$ that depends on many variables and implicitly on the solution of a state equation. In the case of shape optimization for fluids using CFD, the cost function depends on the grid coordinates X and the flow solution Q, i.e., $\mathfrak{I} = \mathfrak{I}(X, Q)$. Differentiating the cost function we have

$$\frac{\mathrm{d}\mathfrak{I}}{\mathrm{d}X} = \frac{\partial \mathfrak{I}}{\partial X} + \frac{\partial \mathfrak{I}}{\partial Q}\frac{\partial Q}{\partial X}$$

which contains the flow sensitivity $\partial Q/\partial X$ with respect to grid perturbations. This can be computed by differentiating the state equation

$$\frac{\partial R}{\partial X} + \frac{\partial R}{\partial Q}\frac{\partial Q}{\partial X} = 0$$

Introducing an *adjoint variable* Ψ, we can write

$$\frac{\mathrm{d}\mathfrak{I}}{\mathrm{d}X} = \frac{\partial \mathfrak{I}}{\partial X} + \frac{\partial \mathfrak{I}}{\partial Q}\frac{\partial Q}{\partial X} + \Psi^{\mathrm{T}}\left[\frac{\partial R}{\partial X} + \frac{\partial R}{\partial Q}\frac{\partial Q}{\partial X}\right]$$

The flow sensitivity can be eliminated from the above equation if Ψ satisfies the adjoint equation

$$\frac{\partial \mathfrak{I}}{\partial Q} + \Psi^{\mathrm{T}}\frac{\partial R}{\partial Q} = 0 \tag{5}$$

Once the adjoint equation is solved the gradient can be computed from

$$\frac{\mathrm{d}\mathfrak{I}}{\mathrm{d}X} = \frac{\partial \mathfrak{I}}{\partial X} + \Psi^{\mathrm{T}} \frac{\partial R}{\partial X} \tag{6}$$

The adjoint equation is a large system of coupled equations which is solved in pseudo-time by introducing a time variable

$$\frac{\partial \Psi}{\partial t} + \left(\frac{\partial R}{\partial Q}\right)^{\mathrm{T}} \Psi + \left(\frac{\partial \mathfrak{I}}{\partial Q}\right)^{\mathrm{T}} = 0 \tag{7}$$

Starting with $\Psi = 0$, the above equation is solved iteratively until the adjoint residual is driven to convergence. Since the adjoint equation has the same eigenvalues as the flow equations, its stability is governed by the same CFL condition. To accelerate convergence to steady state, an implicit scheme based on the same approximate factorization and diagonal approximation as for the flow solver is used.

5 Automatic differentiation

Automatic differentiation refers to a technique to compute derivatives of functions that are implemented as a computer program. Any function implemented in a computer code consists of elementary arithmetic operations and mathematical functions whose derivatives are known. The derivatives of the function can be computed using the chain rule of differentiation. An AD tool automates the implementation of the chain rule; given a computer program, an AD tool generates a new program which computes the derivatives.

AD can be applied in two modes known as *forward* mode and *reverse* mode. Given a program `P` which computes a function $Y = F(X), X \in \mathbb{R}^m, Y \in \mathbb{R}^n$, automatic differentiation of `P` in the forward mode generates a new program $\dot{\mathtt{P}}$ which computes $\dot{Y} = F'(X)\dot{X}$ given any vector $\dot{X} \in \mathbb{R}^m$. Differentiation of `P` in reverse mode generates a new program $\bar{\mathtt{P}}$ which computes $\bar{X} = [F'(X)]^{\mathrm{T}}\bar{Y}$ given any vector $\bar{Y} \in \mathbb{R}^n$. Note that the reverse mode of AD is useful to compute the term $(\partial R/\partial Q)^{\mathrm{T}}\Psi$ appearing in the adjoint equation.

Let us explain these two modes in greater detail. A computer program consists of a sequence of lines, each of which performs some arithmetic computation. If X is the input or independent variables let $T_0 = X$. Then each line of the program creates a new vector of values; the k'th line of the program creates $T_k = F_k(T_{k-1})$ takes the previous set of computed values T_{k-1}. The final function $F(X)$ is thus a composition of all the individual lines of instruction in the code

$$F(X) = F_p(T_{p-1}) \circ F_{p-1}(T_{p-2}) \circ \ldots \circ F_1(T_0)$$

Applying the chain rule of differentiation, we obtain

$$\dot{Y} = F'(X)\dot{X} = F'_p(T_{p-1})F'_{p-1}(T_{p-2}) \ldots F'_1(T_0)\dot{X}$$

where $\dot{X}$ is a given direction vector along which the gradient is required. In the above procedure, the computation of $\dot{Y}$ starts from the first line and proceeds sequentially to the last line; the values T_k are accessed in the forward mode, i.e., from $k = 1$ to $k = p$. This is illustrated schematically in the following flowchart.

$$\begin{tikzcd}
\text{Function:} & T_0 \arrow[r] & F_1 \arrow[r, "T_1"] & F_2 \arrow[r, "T_2"] & \ldots \arrow[r, "T_{p-1}"] & F_p \arrow[r] & F \\
\text{Forward:} & T_0, \dot{X} \arrow[r] & F'_1 \arrow[r, "T_1"] & F'_2 \arrow[r, "T_2"] & \ldots \arrow[r, "T_{p-1}"] & F'_p \arrow[r] & \dot{Y}
\end{tikzcd}$$

If $F(X)$ is a scalar, i.e., $n = 1$, then computation of the full gradient vector $F'(X) \in \mathbb{R}^m$ requires the computation $\dot{Y}$ for m different vectors $\dot{X}$. For example, if $\dot{X} = [1, 0, 0, \ldots, 0]^T \in \mathbb{R}^m$, then $\dot{Y} = \partial F/\partial X_1$; the cost of computing the full gradient is proportional to the number of independent variables m.

In the reverse mode of AD, we compute the following transposed expression

$$\bar{X} = [F'(X)]^{\mathrm{T}}\bar{Y} = [F_1'(T_0)]^{\mathrm{T}}[F_2'(T_1)]^{\mathrm{T}} \ldots [F_p'(T_{p-1})]^{\mathrm{T}}\bar{Y}$$

given $\bar{Y} \in \mathbb{R}^n$. If $F(X)$ is a scalar, then if we set $\bar{Y} = 1$, then the reverse mode give the full gradient in a single computation; the cost of computing the full gradient does not depend on the number of independent variables. The flow of variables in the reverse mode is illustrated below:

$$\begin{tikzcd}
\text{Function:} & T_0 \arrow[r] & F_1 \arrow[r, "T_1"] & F_2 \arrow[r, "T_2"] & \ldots \arrow[r, "T_{p-1}"] & F_p \arrow[r] & F \\
\text{Reverse:} & T_{p-1}, \bar{Y} \arrow[r] & {[F_p']^T} \arrow[r, "T_{p-2}"] & {[F_{p-1}']^T} \arrow[r, "T_{p-3}"] & \ldots \arrow[r, "T_0"] & {[F_1']^T} \arrow[r] & \bar{X}
\end{tikzcd}$$

The order in which the variables are required for reverse mode is opposite to the order in which they are computed during function evaluation. This means that all the variables T_k that are required in the reverse sweep must be first computed in a forward sweep, and then the reverse mode is executed. If some of these variables are overwritten, they must be either recomputed or stored in a memory stack so that they can be accessed during reverse sweep.

In this work we use TAPENADE to differentiate a CFD code written in fortran 77; full details will be given in the presentation. Any linear portions of the code are hand differentiated in reverse mode which saves memory, with only the non-linear portions of the code being differentiated using AD. The non-linear parts of the code like flux computation and limiters are encapsulated in separate subroutines and AD is applied only to these routines.

6 Shape parameterization and grid deformation

A 2-D shape consists of a collection of curves. For example, an airfoil is made up of two curves, one for the upper surface and another for the lower surface. The curves can be represented using Bezier polynomials, B-splines or NURBS. Alternatively, we can parameterize the shape deformation itself instead of the shape. In this approach, there is a reference shape which is deformed by the application of some smooth functions. Consider a 2-D curve joinings points A and B; the unit vector (n_x, n_y) is perpendicular to the line AB joining the end-points of the curve and $\xi \in [0, 1]$ is a local coordinate along this line. A linear combination of smooth functions $\{B_k\}$ defined on the line AB is used to deform the shape. The coordinates of the points on the curve are given by

$$\begin{bmatrix} x_s \\ y_s \end{bmatrix} = \begin{bmatrix} x_s^{(o)} \\ y_s^{(o)} \end{bmatrix} + \begin{bmatrix} n_x \\ n_y \end{bmatrix} h(\xi), \quad h(\xi) = \sum_{k=1}^{m} \beta_k B_k(\xi)$$

where $\beta = (\beta_1, \ldots, \beta_m)$ is the set of design variables. This results in moving the points on the curve only along the normal to the line AB. For airfoils, an effective set of functions were developed by Hicks and Henne [2] and will be used in the present work.

When the shape is changed, the grid must be deformed to conform to the new shape. There are several methods available to deform a grid. Some methods are based on a spring analogy and others use elasticity equations to propagate the deformation. A third approach is based

Method	$\mathfrak{I}$	$100C_d$	C_l	N_{fun}	N_{grad}
Initial	1.000	2.072	0.295	-	-
`conmin_frcg`	0.170	0.389	0.325	50	13
`optpp_q_newton`	0.142	0.329	0.329	50	39
`steep`	0.166	0.335	0.287	50	45

Table 1: Maximizing L/D

on interpolation using radial basis functions which is adopted in the present work and leads to good quality grids even under large deformations.

Given N data points $f_j = f(x_j, y_j)$, $j = 1, \ldots, N$, the TPS interpolation function is of the form

$$\tilde{f}(x, y) = a_0 + a_1 x + a_2 y + \sum_{j=1}^{N} b_j |\vec{r} - \vec{r}_j|^2 \log |\vec{r} - \vec{r}_j|, \quad \vec{r} = (x, y) \tag{8}$$

where the coefficients $a_0, a_1, a_2, \{b_j\}$ must be determined from the interpolation conditions $\tilde{f}(x_j, y_j) = f_j$, $j = 1, \ldots, N$. Let there be N_s grid points on the boundary of the flow domain which includes moving points on the shape being optimized and fixed points on remaining parts of the boundary like the outer boundary of an airfoil. A TPS interpolation is constructed for the x and y coordinates using the displacement of the boundary points. The displacement for an interior point is obtained from the TPS interpolation which allows us to deform the entire grid.

7 Numerical Results

We apply the optimization framework developed here to solve some 2-D inviscid airfoil shape optimization problems involving transonic flows. The gradients obtained using the adjoint approach are first validated to be correct by comparing them against finite difference gradients. We solved different optimization problems using simple steepest descent method and more sophisticated algorithms like conjugate gradient (`conmin_frcg`), method of feasible directions (`conmin_mfd`), quasi-Newton (`optpp_q_newton`), sequential quadratic programming (`fsqp`) and interior point method (`ipopt`).

7.1 Maximizing L/D

The lift to drag ratio is maximized without any other constraints; the cost fuction is the reciprocal, which will be minimized.

$$\mathfrak{I} = \frac{C_d}{C_l} \frac{C_{l_0}}{C_{d_0}}$$

This is an unconstrained problem and hence even steepest descent methods can be used. The starting airfoil is the NACA0012 section; an O-grid of size 161 × 41 is used with 161 points on the airfoil. The upper and lower surfaces are parameterized using 10 Hicks-Henne functions each leading to a total of 20 design variables.

Table (1) and figure (2) show the result of optimization using three different methods. The quasi-newton gave the best reduction in objective function with the steepest-descent being a

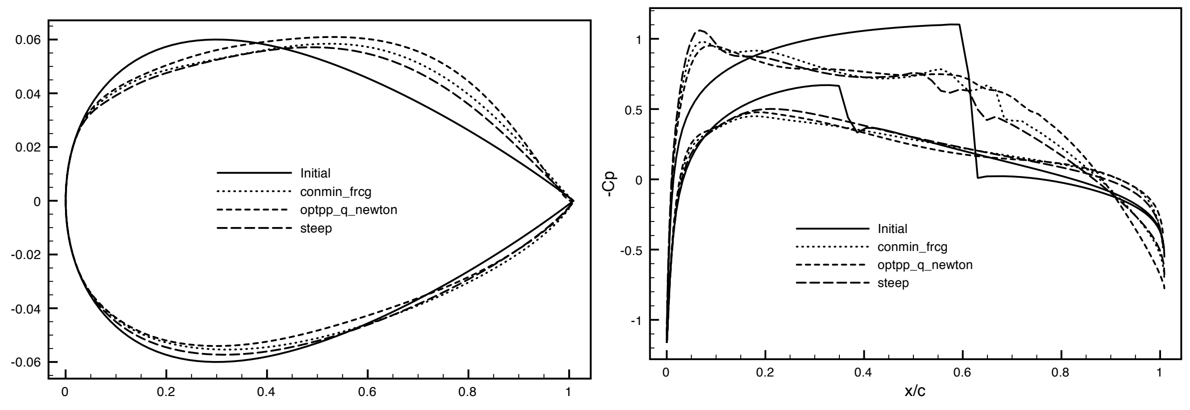

Figure 2: Maximizing L/D for NACA0012 airfoil

close second while the conjugate-gradient method did not perform well. All methods are able to eliminate the weak shock on the bottom surface of the airfoil. The shock on the upper surface is also eliminated by all the three methods but the quasi-newton gives the smoothest pressure distribution. In terms of computational cost, quasi-newton and steepest-descent fare similarly; the conjugate-gradient took fewer adjoint solutions but the objective function reduction is not as good as with the other two methods. Comparing the final shapes, the quasi-newton method leads to the largest modification in the shape from initial shape.

7.2 Lift-constrained drag minimization

Minimizing the drag of the airfoil so that the lift satisfies some constraint is an important aerodynamic optimization problem. The starting airfoil is RAE2822 at a Mach number of 0.729 and angle of attack of 2.31 degrees. A C-grid of size 243×43 is used with 200 points on the airfoil. The shape is parameterized using 20 Hicks-Henne functions with 10 functions each for upper and lower parts of the airfoil.

In this test case, the drag is minimized by treating the lift as a non-linear constraint. Three methods are used: `conmin_mfd` , `fsqp` and `ipopt` . Table (2) and figure (3) show the results of optimization using the three methods. All three methods yield similar amounts of drag reduction and are able to satisfy the lift constraint. In terms of number of adjoint solutions, `conmin_mfd` is the most efficient while `ipopt` requires many adjoint solutions. Both `conmin_mfd` and `ipopt` yield similar shapes and they do not modify the bottom surface. But `fsqp` makes considerable change to the bottom surface also. Another problem with it is that it tries to reduce the airfoil thickness close to the trailing edge.

8 Summary and conclusions

The adjoint method is used to develop a shape optimization framework for compressible flows governed by Euler equations. An efficient adjoint solver is constructed using automatic differentiation tools. The developed framework is applied to optimize the shape of inviscid, transonic airfoils. The result of optimization is the elimination of shocks. A comparison between a simple steepest descent method and more sophisticated algorithms is performed. We find that the

Method	$\mathfrak{J}$	$100C_d$	C_l	N_{fun}	N_{grad}
Initial	1.000	1.150	0.887	-	-
`conmin_mfd`	0.342	0.394	0.881	50	22
`fsqp`	0.325	0.374	0.887	50	32
`ipopt`	0.335	0.385	0.887	45	62

Table 2: Lift-constrained drag minimization for RAE2822

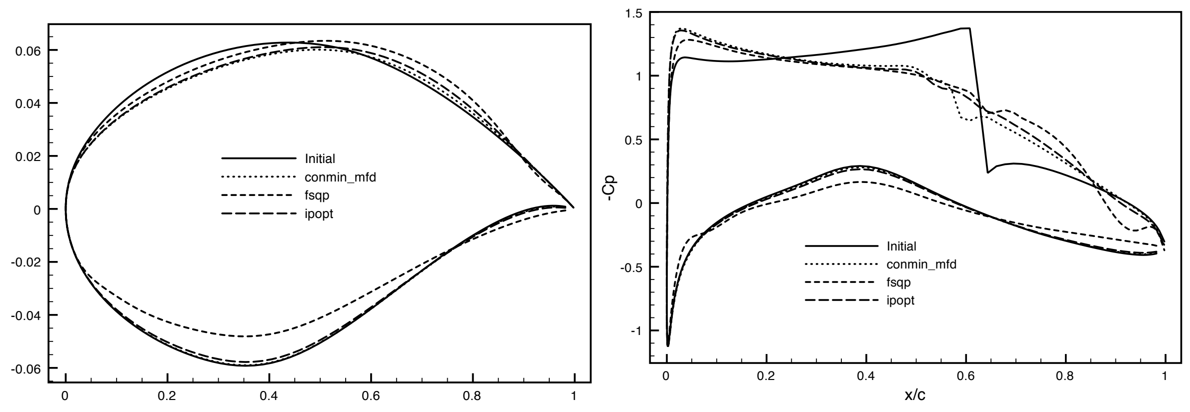

Figure 3: Lift-constrained drag minimization of RAE2822 airfoil

steepest descent is quite competitive with some of the most sophisticated methods available today.

References

[1] Elliott, J. and Peraire, J., *AIAA J.*, Vol. 35, pp. 1479-1485, 1997.

[2] Hicks, R. M. and Henne, P. A., *J. Aircraft*, Vol. 15, No. 7, 1978.

[3] Jameson A., *J. Sci. Comp.*, Vol. 3, No. 3, pp. 233-260, 1988.

[4] Koren, B., *Proc. 11'th Int. Conf. Num. Meth. Fluids*, editor, D. Dwoyer, M. Hussaini and R. Voigt, Springer, 1989.

[5] Pironneau, O., *J. Fluid Mech.*, Vol. 64, pp. 97-110, 1974.

[6] Pulliam, T.H., in *VKI Lecture Series : Numerical Techniques for Viscous Flow Computation In Turbomachinery Bladings*, von Kármán Institute, Rhode-St-Genese, Belgium , 1985

[7] Roe, P. L., *J. Comp. Phy.*, Vol. 43, pp. 357-372, 1981.

[8] TAPENADE, `http://www-sop.inria.fr/tropics`