48th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition
4 - 7 January 2010, Orlando, Florida

AIAA 2010-1268

Efficient Hessian Calculations using Automatic Differentiation and the Adjoint Method

Markus P. Rumpfkeil* and Dimitri J. Mavriplis †

University of Wyoming, Laramie, 82071, USA

In this paper an efficient general algorithm to calculate the Hessian of a steady or unsteady functional of interest in the context of computational fluid dynamics is outlined, validated and applied to an aerodynamic optimization and to an extrapolation example. The successful extrapolation is then applied to approximate Monte Carlo simulations for artificial geometric uncertainty analysis. The presented optimization examples demonstrate that the combination of automatic differentiation and an adjoint method to calculate the Hessian of a steady or unsteady objective function, and thereby obtaining second order information, can be an efficient tool for optimization and uncertainty quantification.

Nomenclature

α	Angle of attack
C_d^n	Drag coefficient at time step n
C_l^n	Lift coefficient at time step n
D	Design variables
$\mathfrak{D}_{jk}, \mathfrak{d}_{jk}$	Derivative operators
L^g	Unsteady objective function
L^n	Objective function at time step n
f	Steady objective function
$\frac{df}{dD_j}$	Gradient of steady objective function
$\frac{d^2 f}{dD_j dD_k}$	Hessian of steady objective function
$\mathcal{L}$	Lagrangian
M	Total number of design variables
M_∞	Free-stream Mach number
N	Total number of time steps
q^n	Flow variables at time step n
$q_j^n = \frac{dq^n}{dD_j}$	Derivative of flow variables w.r.t. design variables at time step n
R^n	Unsteady flow residual
$(\nabla_{q^n} R^n)^{-T}$	Inverse of the transpose of the unsteady flow Jacobian
R	Steady flow residual
s^n	Unsteady grid residual
$(\nabla_{x^n} s^n)^{-T}$	Inverse of the transpose of the unsteady grid Jacobian
s	Steady grid residual
T	Final time
$\mathcal{W}_i$	Weights
x^n	Grid variables at time step n
$x_j^n = \frac{dx^n}{dD_j}$	Derivative of grid variables w.r.t. design variables at time step n
λ^n	Mesh adjoint variables at time step n
$\mu_{\mathcal{J}}$	Mean of objective function
ψ^n	Flow adjoint variables at time step n

*Postdoctoral Research Associate, Department of Mechanical Engineering; mrumpfke@uwyo.edu, Member AIAA
†Professor, Department of Mechanical Engineering; mavripl@uwyo.edu, Associate Fellow AIAA

$\sigma_{\mathcal{J}}$	Standard deviation of objective function
σ_{D_j}	Standard deviation of design variable j
*	Target value

I. Introduction and Motivation

The concept of using automatic differentiation (AD) in combination with an adjoint method to calculate the Hessian was initially investigated by Taylor *et al.*[1] and refined for a steady computational fluid dynamics (CFD) code by Ghate and Giles.[2] Tortorelli and Michaleris[3] worked on structural optimization and computed the Hessian matrix using discrete direct and adjoint formulations and Papadimitriou and Giannakoglou[4] used a continuous adjoint formulation to calculate the Hessian for a Newton based optimizer to reconstruct ducts and cascade airfoils for a known pressure distribution at inviscid flow conditions. The Hessian, once obtained, has various applications such as optimization, extrapolation, Monte-Carlo (MC) simulations, surrogate modeling and uncertainty analysis.

The most widely used method of finite differencing to obtain the Hessian is sensitive to step-size selection and is computationally expensive. On the other hand, the use of AD to calculate the Jacobian or Hessian is appealing since this method is accurate to machine precision[5] and it helps through its automation to keep the linearized version synchronized with potentially frequent changes made to the nonlinear code (very easily accomplished with the help of a `Makefile`). There are many mature AD tools (ADOL-C, ADIFOR, TAPENADE, etc.) available and for this work TAPENADE[6] is employed.

The concept of calculating Hessians using AD has been addressed by the AD community for more than a decade.[7] There are two commonly used methods: forward-on-forward and forward-on-reverse. Forward-on-forward is a straightforward double differentiation of the entire original code in forward mode. Similarly, in forward-on-reverse the entire code is first differentiated in reverse and then in forward mode. However, the computational cost of using either of the two methods for entire large iterative solution codes is prohibitively expensive. The general algorithm presented here mitigates some of these expenses by employing an adjoint method as well as using AD very judiciously only on selected routines.

In Section II the basic algorithm for calculating the Hessian of a general steady problem using AD and an adjoint method is outlined. Section III then extends these ideas to unsteady problems and Section IV shows some validation results and the application to an aerodynamic inverse design optimization problem. Finally, Sections V.A and V.B show the use of the Hessian for the extrapolation of a functional and uncertainty analysis, respectively. Section VI concludes this paper.

II. Basic Formulation for General Steady Problems

We derive the Hessian of a steady functional of interest (such as lift or drag)

$$f(D) = F(D, x(D), q(D)), \quad f \in \mathbb{R} \tag{1}$$

with respect to the independent design variables $D \in \mathbb{R}^M$ such that the grid coordinate variables $x(D) \in \mathbb{R}^X$ and flow variables $q(D) \in \mathbb{R}^Q$ satisfy the grid deformation residual equation

$$s(D, x(D)) = 0, \quad s \in \mathbb{R}^X \tag{2}$$

and flow residual equation

$$R(D, x(D), q(D)) = 0, \quad R \in \mathbb{R}^Q. \tag{3}$$

The presented derivation is very similar to the one given by Ghate and Giles.[2]

The first derivative of f with respect to one individual component of D is given by

$$\frac{df}{dD_j} = \frac{\partial F}{\partial D_j} + \frac{\partial F}{\partial x}\frac{dx}{dD_j} + \frac{\partial F}{\partial q}\frac{dq}{dD_j}. \tag{4}$$

Differentiating equation (4) again yields

$$\frac{d^2 f}{dD_j dD_k} = \mathfrak{D}_{jk}F + \frac{\partial F}{\partial x}\frac{d^2 x}{dD_j dD_k} + \frac{\partial F}{\partial q}\frac{d^2 q}{dD_j dD_k}, \tag{5}$$

where

$$
\begin{aligned}
\mathfrak{D}_{jk}F &= \frac{\partial^2 F}{\partial D_j \partial D_k} + \frac{\partial^2 F}{\partial D_j \partial x}x_k + \frac{\partial^2 F}{\partial D_k \partial x}x_j + \frac{\partial^2 F}{\partial D_j \partial q}q_k + \frac{\partial^2 F}{\partial D_k \partial q}q_j \\
&+ \frac{\partial^2 F}{\partial x \partial q}\left(q_j x_k + x_j q_k\right) + \frac{\partial^2 F}{\partial x^2}x_j x_k + \frac{\partial^2 F}{\partial q^2}q_j q_k,
\end{aligned} \tag{6}
$$

with $x_j := \frac{dx}{dD_j}$ and $q_j := \frac{dq}{dD_j}$.

Thus, the calculation of the symmetric Hessian ($\frac{d^2 f}{dD_j dD_k} = \frac{d^2 f}{dD_k dD_j}$) requires the first as well as second order sensitivities of x and q with respect to the design variables. The computational cost of this calculation is

- one (nonlinear) baseline solution for x using the grid residual equation (2)
- one nonlinear baseline solution for q using the flow residual equation (3)
- M linear solutions each for $x_j = \frac{dx}{dD_j}$ and $q_j = \frac{dq}{dD_j}$ using equation (7) and (10), respectively
- $M(M+1)/2$ linear solutions each for $\frac{d^2x}{dD_j dD_k}$ and $\frac{d^2q}{dD_j dD_k}$ using equation (8) and (11), respectively
- $M(M+1)/2$ evaluations of the right-hand side of equation (5)

II.A. More Computationally Efficient Formulation using the Adjoint

Differentiating the grid residual equation (2) gives

$$
\frac{\partial s}{\partial D_j} + \frac{\partial s}{\partial x}\frac{dx}{dD_j} = 0, \tag{7}
$$

and differentiating again results in

$$
\mathfrak{d}_{jk}s + \frac{\partial s}{\partial x}\frac{d^2 x}{dD_j dD_k} = 0, \tag{8}
$$

with

$$
\mathfrak{d}_{jk}s = \frac{\partial^2 s}{\partial D_j \partial D_k} + \frac{\partial^2 s}{\partial D_j \partial x}x_k + \frac{\partial^2 s}{\partial D_k \partial x}x_j + \frac{\partial^2 s}{\partial x^2}x_j x_k. \tag{9}
$$

Similarly, differentiating the flow residual equation (3) gives

$$
\frac{\partial R}{\partial D_j} + \frac{\partial R}{\partial x}\frac{dx}{dD_j} + \frac{\partial R}{\partial q}\frac{dq}{dD_j} = 0, \tag{10}
$$

and differentiating again we obtain,

$$
\mathfrak{D}_{jk}R + \frac{\partial R}{\partial x}\frac{d^2 x}{dD_j dD_k} + \frac{\partial R}{\partial q}\frac{d^2 q}{dD_j dD_k} = 0, \tag{11}
$$

with $\mathfrak{D}_{jk}R$ defined analogues to equation (6)

$$
\begin{aligned}
\mathfrak{D}_{jk}R &= \frac{\partial^2 R}{\partial D_j \partial D_k} + \frac{\partial^2 R}{\partial D_j \partial x}x_k + \frac{\partial^2 R}{\partial D_k \partial x}x_j + \frac{\partial^2 R}{\partial D_j \partial q}q_k + \frac{\partial^2 R}{\partial D_k \partial q}q_j \\
&+ \frac{\partial^2 R}{\partial x \partial q}\left(q_j x_k + x_j q_k\right) + \frac{\partial^2 R}{\partial x^2}x_j x_k + \frac{\partial^2 R}{\partial q^2}q_j q_k.
\end{aligned} \tag{12}
$$

Solving equation (8) for $\frac{d^2x}{dD_j dD_k}$ and equation (11) for $\frac{d^2q}{dD_j dD_k}$ and substituting the resulting expressions into equation (5) yields

$$
\frac{d^2 f}{dD_j dD_k} = \mathfrak{D}_{jk}F - \frac{\partial F}{\partial x}\left(\frac{\partial s}{\partial x}\right)^{-1}\mathfrak{d}_{jk}s - \frac{\partial F}{\partial q}\left(\frac{\partial R}{\partial q}\right)^{-1}\left[\mathfrak{D}_{jk}R - \frac{\partial R}{\partial x}\left(\frac{\partial s}{\partial x}\right)^{-1}\mathfrak{d}_{jk}s\right]. \tag{13}
$$

Defining the following flow adjoint problem as an intermediate problem

$$\left(\frac{\partial R}{\partial q}\right)^T \psi = -\left(\frac{\partial F}{\partial q}\right)^T \tag{14}$$

simplifies equation (13) to

$$\frac{d^2 f}{dD_j dD_k} = \mathfrak{D}_{jk} F + \psi^T \mathfrak{D}_{jk} R - \left[\frac{\partial F}{\partial x} + \psi^T \frac{\partial R}{\partial x}\right] \left(\frac{\partial s}{\partial x}\right)^{-1} \mathfrak{d}_{jk} s. \tag{15}$$

Similarly, defining the grid deformation adjoint problem as

$$\left(\frac{\partial s}{\partial x}\right)^T \lambda = -\left[\frac{\partial F}{\partial x} + \psi^T \frac{\partial R}{\partial x}\right]^T \tag{16}$$

simplifies equation (15) to

$$\frac{d^2 f}{dD_j dD_k} = \mathfrak{D}_{jk} F + \psi^T \mathfrak{D}_{jk} R + \lambda^T \mathfrak{d}_{jk} s. \tag{17}$$

In order to calculate the complicated derivatives arising from the two derivative operators $\mathfrak{D}_{jk}$ and $\mathfrak{d}_{jk}$ one can use AD. One simple way of doing this is to have, for example, a routine which returns the grid residual s given the inputs D and x. This routine can then be double differentiated in the forward mode using the AD software.

Note that the two adjoint variables ψ and λ can also be used to calculate the first derivative of the functional of interest given by equation (4) more efficiently:

$$\frac{df}{dD_j} = \frac{\partial F}{\partial D_j} + \lambda^T \frac{\partial s}{\partial D_j} + \psi^T \frac{\partial R}{\partial D_j}. \tag{18}$$

In order to calculate the terms $x_j = \frac{dx}{dD_j}$ and $q_j = \frac{dq}{dD_j}$ required for $\mathfrak{D}_{jk} F$, $\mathfrak{D}_{jk} R$, and $\mathfrak{d}_{jk} s$ one can solve equations (7) and (10) to obtain

$$\frac{dx}{dD_j} = -\left(\frac{\partial s}{\partial x}\right)^{-1} \frac{\partial s}{\partial D_j}, \tag{19}$$

and

$$\frac{dq}{dD_j} = -\left(\frac{\partial R}{\partial q}\right)^{-1} \left[\frac{\partial R}{\partial D_j} + \frac{\partial R}{\partial x}\frac{\partial x}{\partial D_j}\right]. \tag{20}$$

The computational cost for calculating the Hessian is now reduced to

- one (nonlinear) baseline solution for x using the grid residual equation (2)
- one nonlinear baseline solution for q using the flow residual equation (3)
- one linear adjoint solution each for ψ and λ using equation (14) and (16), respectively
- M linear solutions each for $x_j = \frac{dx}{dD_j}$ and $q_j = \frac{dq}{dD_j}$ using equation (19) and (20), respectively
- $M(M+1)/2$ cheap evaluations of the right-hand side of equation (17)

Note that if one can employ a linear solver which efficiently supports multiple right-hand sides, the computational cost can be even further reduced since the left-hand sides in equations (19) and (20) do not change. Theoretically, one could even combine this solver with the linear adjoint solutions given by equations (14) and (16) which also use the same left-hand side, only transposed. It is almost needless to say that one should use expensive but effective forms of preconditioning since this cost is easily amortized over a large number of right-hand sides (design variables). One last idea is to parallelize the adjoint solutions for ψ and λ and the M linear solutions each for $x_j = \frac{dx}{dD_j}$ and $q_j = \frac{dq}{dD_j}$ as $M+1$ processes. Assuming no limitations in the number of processors available, one can obtain the Hessian at the same time one calculates the gradient.

Another important observation is that third order tensors are never explicitly needed (e.g. $\frac{\partial^2 R}{\partial q^2}$) but rather the result of these tensors pre-multiplied with the corresponding adjoint variable and post-multiplied with combinations of q_j and x_k. A very efficient way of obtaining the result of $\frac{\partial^2 R}{\partial q^2} q_j q_k$, for example, is simply to call the corresponding double differentiated routine with the two "perturbation" vectors q_j and q_k to get the desired result as an output from the routine with only one call.

II.B. Validation

As recommended by Ghate and Giles[2] one should always introduce validation checks to ensure the correctness of the implementation. Obviously, one should confirm that

$$s(D, x(D)) = 0, \quad R(D, x(D), q(D)) = 0,$$

as well as

$$\frac{\partial s}{\partial D_j} + \frac{\partial s}{\partial x} x_j = 0, \quad \text{and} \quad \frac{\partial R}{\partial D_j} + \frac{\partial R}{\partial x} x_j + \frac{\partial R}{\partial q} q_j = 0$$

are fulfilled to machine precision. One should also verify that the computed Hessian is symmetric, i.e. $\frac{d^2 f}{dD_j dD_k} = \frac{d^2 f}{dD_k dD_j}$.

However, the final validation is the comparison with finite difference results. A second-order accurate central finite difference approximation is given by

$$\frac{d^2 f}{dD_j dD_j} = \frac{f(D + h_j) - 2f(D) + f(D - h_j)}{h^2} \tag{21}$$

for the diagonal and

$$\frac{d^2 f}{dD_j dD_k} = \frac{f(D + h_j + h_k) - f(D + h_j - h_k) - f(D - h_j + h_k) + f(D - h_j - h_k)}{4h^2} \tag{22}$$

for the off-diagonal elements. Here, $h \approx 10^{-5}$ and h_j is a perturbation for the j^{th} design variable. Thus, calculating only the upper triangular portion of the Hessian matrix by central finite differences requires $2M + 4\frac{M(M-1)}{2} = 2M^2$ additional solutions of the grid and flow residual equations.

Slightly cheaper but also less accurate is a first-order forward finite difference approximation for the off-diagonal elements given by

$$\frac{d^2 f}{dD_j dD_k} = \frac{f(D + h_j + h_k) - f(D + h_j) - f(D + h_k) + f(D)}{h^2} \tag{23}$$

leading to only $2M + \frac{M(M-1)}{2} = 0.5M^2 + 1.5M$ additional solutions of the grid and flow residual equations. We present validation results in Section IV.

II.C. Approximation of the Hessian for Inverse-design-type Functionals

In the special case of inverse-design-type functionals for steady flows given by

$$f(D) = \frac{1}{2} \sum_{i=1}^{I} \mathcal{W}_i \Big(F_i(D, x(D), q(D)) - F_i^* \Big)^2, \tag{24}$$

a computationally cheap approximation of the Hessian can be derived. Here, F_i can be a quantity such as lift or drag, F_i^* is a target lift or drag and $\mathcal{W}_i$ are weights. The first derivative of f with respect to one individual component of D is given by

$$\frac{df}{dD_j} = \sum_{i=1}^{I} \mathcal{W}_i \left(\frac{dF_i}{dD_j} \right)^T \Big(F_i - F_i^* \Big). \tag{25}$$

Differentiating equation (25) again yields

$$\begin{aligned} \frac{d^2 f}{dD_j dD_k} &= \sum_{i=1}^{I} \mathcal{W}_i \left(\frac{dF_i}{dD_j} \right)^T \frac{dF_i}{dD_k} + \sum_{i=1}^{I} \mathcal{W}_i \Big(F_i - F_i^* \Big) \frac{d^2 F_i}{dD_j dD_k} \\ &\approx \sum_{i=1}^{I} \mathcal{W}_i \left(\frac{dF_i}{dD_j} \right)^T \frac{dF_i}{dD_k}. \end{aligned} \tag{26}$$

The last approximation is true if we are close to the optimum where $F_i \approx F_i^*$ for $i = 1, \ldots, I$. Equation (26) implies that we can approximate the Hessian by only determining the first derivatives $\frac{dF_i}{dD_j}$ for $i = 1, \ldots, I$ using equation (18). Unfortunately, an extension to unsteady inverse-design-type functionals is computationally expensive and one has to use the approach described in the next section instead.

III. Extension to Unsteady Problems

In the unsteady case a general functional of interest is given by

$$L^g(D) = \sum_{n=0}^{N} L^n(q^n(D), x^n(D), D), \quad L^g \in \mathbb{R}, \tag{27}$$

where N is the number of time steps and $D \in \mathbb{R}^M$ are the independent design variables. The time-dependent grid variables $x^n(D) \in \mathbb{R}^X$ satisfy the unsteady grid residual equations

$$s^n(x^n(D), D) = 0 \quad \text{for } n = 0, \ldots, N. \tag{28}$$

If one assumes a steady flow solve followed by one time step of a one-step time-marching method (e.g. implicit Euler) and the use of a two-step time-marching method thereafter (e.g. BDF2, Leapfrog, AB2) then the unsteady flow variables $q^n(D) \in \mathbb{R}^Q$ are implicitly defined via the following unsteady flow residuals

$$\begin{aligned} R^0(q^0(D), x^0(D), D) &= 0 \\ R^1(q^1(D), x^1(D), q^0(D), x^0(D), D) &= 0 \\ R^n(q^n(D), x^n(D), q^{n-1}(D), x^{n-1}(D), q^{n-2}(D), x^{n-2}(D), D) &= 0 \quad \text{for } n = 2, \ldots, N. \end{aligned} \tag{29}$$

The problem of minimizing the discrete objective function L^g given by equation (27) is then equivalent to the unconstrained optimization problem of minimizing the Lagrangian function

$$\begin{aligned} \mathcal{L} &= \mathcal{L}(q^0, \ldots, q^N, x^0, \ldots, x^N, \psi^0, \ldots, \psi^N, \lambda^0, \ldots, \lambda^N, D) \\ &= \sum_{n=0}^{N} L^n(q^n, x^n, D) + \sum_{n=0}^{N} (\psi^n)^T R^n + \sum_{n=0}^{N} (\lambda^n)^T s^n(x^n, D) \end{aligned} \tag{30}$$

with respect to $q^0, \ldots, q^N$, $x^0, \ldots, x^N$, $\psi^0, \ldots, \psi^N$, $\lambda^0, \ldots, \lambda^N$ and D, where ψ^n and λ^n are the Lagrange multipliers. The following derivation is based on work by Rumpfkeil and Zingg.[8,9] Since the states $x^0, \ldots, x^N$ and $q^0, \ldots, q^N$ are calculated using the residuals given by equations (28) and (29), it is automatically guaranteed that $\nabla_{\psi^n} \mathcal{L} = \nabla_{\lambda^n} \mathcal{L} = 0$ for $n = 0, \ldots, N$.

The Lagrange multipliers ψ^n (or flow adjoints) must now be chosen such that $\nabla_{q^n} \mathcal{L} = 0$ for $n = 0, \ldots, N$, which leads to

$$\begin{aligned} 0 &= \nabla_{q^n} L^n + (\psi^n)^T \nabla_{q^n} R^n + (\psi^{n+1})^T \nabla_{q^n} R^{n+1} + (\psi^{n+2})^T \nabla_{q^n} R^{n+2} \\ &\quad \text{for} \quad n = 0, \ldots, N-2 \\ 0 &= \nabla_{q^{N-1}} L^{N-1} + (\psi^N)^T \nabla_{q^{N-1}} R^N + (\psi^{N-1})^T \nabla_{q^{N-1}} R^{N-1} \\ 0 &= \nabla_{q^N} L^N + (\psi^N)^T \nabla_{q^N} R^N, \end{aligned} \tag{31}$$

which can be written equivalently as

$$\begin{aligned} \psi^N &= -(\nabla_{q^N} R^N)^{-T} \left[(\nabla_{q^N} L^N)^T \right] \\ \psi^{N-1} &= -(\nabla_{q^{N-1}} R^{N-1})^{-T} \left[(\nabla_{q^{N-1}} L^{N-1})^T + (\nabla_{q^{N-1}} R^N)^T \psi^N \right] \\ \psi^n &= -(\nabla_{q^n} R^n)^{-T} \left[(\nabla_{q^n} L^n)^T + (\nabla_{q^n} R^{n+1})^T \psi^{n+1} + (\nabla_{q^n} R^{n+2})^T \psi^{n+2} \right] \\ &\quad \text{for } n = N-2, \ldots, 0. \end{aligned} \tag{32}$$

Similarly, the Lagrange multipliers λ^n (or mesh adjoints) must fulfill $\nabla_{x^n} \mathcal{L} = 0$ for $n = 0, \ldots, N$, which gives

$$\begin{aligned} 0 &= \nabla_{x^n} L^n + (\psi^n)^T \nabla_{x^n} R^n + (\psi^{n+1})^T \nabla_{x^n} R^{n+1} + (\psi^{n+2})^T \nabla_{x^n} R^{n+2} + (\lambda^n)^T \nabla_{x^n} s^n \\ &\quad \text{for} \quad n = 0, \ldots, N-2 \\ 0 &= \nabla_{x^{N-1}} L^{N-1} + (\psi^N)^T \nabla_{x^{N-1}} R^N + (\psi^{N-1})^T \nabla_{x^{N-1}} R^{N-1} + (\lambda^{N-1})^T \nabla_{x^{N-1}} s^{N-1} \\ 0 &= \nabla_{x^N} L^N + (\psi^N)^T \nabla_{x^N} R^N + (\lambda^N)^T \nabla_{x^N} s^N, \end{aligned} \tag{33}$$

which can be written equivalently as

$$
\begin{aligned}
\lambda^N &= -(\nabla_{x^N} s^N)^{-T} \left[(\nabla_{x^N} L^N)^T + (\nabla_{x^N} R^N)^T \psi^N\right] \\
\lambda^{N-1} &= -(\nabla_{x^{N-1}} s^{N-1})^{-T} \left[(\nabla_{x^{N-1}} L^{N-1})^T + (\nabla_{x^{N-1}} R^N)^T \psi^N + (\nabla_{x^{N-1}} R^{N-1})^T \psi^{N-1}\right] \\
\lambda^n &= -(\nabla_{x^n} s^n)^{-T} \Big[(\nabla_{x^n} L^n)^T + (\nabla_{x^n} R^n)^T \psi^n + (\nabla_{x^n} R^{n+1})^T \psi^{n+1} + (\nabla_{x^n} R^{n+2})^T \psi^{n+2} \Big] \\
&\quad \text{for } n = N-2, \ldots, 0.
\end{aligned} \tag{34}
$$

Finally, one can calculate the gradient of the unsteady functional (27) with respect to one individual component of the design variables D:

$$
\begin{aligned}
\frac{dL^g}{dD_j} &= \left.\frac{\partial \mathcal{L}}{\partial D_j}\right|_{\frac{\partial \mathcal{L}}{\partial q^n} = \frac{\partial \mathcal{L}}{\partial x^n} = \frac{\partial \mathcal{L}}{\partial \lambda^n} = \frac{\partial \mathcal{L}}{\partial \psi^n} = 0} \\
&= \sum_{n=0}^{N} \frac{\partial L^n(q^n, x^n, D)}{\partial D_j} + \sum_{n=0}^{N} (\psi^n)^T \frac{\partial R^n}{\partial D_j} + \sum_{n=0}^{N} (\lambda^n)^T \frac{\partial s^n(x^n, D)}{\partial D_j}.
\end{aligned} \tag{35}
$$

The Hessian of the unsteady functional L^g is given by

$$
\frac{d^2 L^g}{dD_j dD_k} = \sum_{n=0}^{N} \left(\mathfrak{D}_{jk} L^n + \frac{\partial L^n}{\partial x^n} \frac{d^2 x^n}{dD_j dD_k} + \frac{\partial L^n}{\partial q^n} \frac{d^2 q^n}{dD_j dD_k} \right), \tag{36}
$$

where

$$
\begin{aligned}
\mathfrak{D}_{jk} L^n &= \frac{\partial^2 L^n}{\partial (q^n)^2} q_j^n q_k^n + \frac{\partial^2 L^n}{\partial q^n \partial x^n} \left(q_j^n x_k^n + x_j^n q_k^n \right) + \frac{\partial^2 L^n}{\partial (x^n)^2} x_j^n x_k^n \\
&+ \frac{\partial^2 L^n}{\partial D_j \partial q^n} q_k^n + \frac{\partial^2 L^n}{\partial D_k \partial q^n} q_j^n + \frac{\partial^2 L^n}{\partial D_j \partial x^n} x_k^n + \frac{\partial^2 L^n}{\partial D_k \partial x^n} x_j^n + \frac{\partial^2 L^n}{\partial D_j \partial D_k},
\end{aligned} \tag{37}
$$

with $x_j^n := \frac{dx^n}{dD_j}$ and $q_j^n := \frac{dq^n}{dD_j}$.

Differentiating the unsteady grid residual equations (28) gives

$$
\frac{\partial s^n}{\partial D_j} + \frac{\partial s^n}{\partial x^n} \frac{dx^n}{dD_j} = 0, \tag{38}
$$

and differentiating again results in

$$
\mathfrak{d}_{jk} s^n + \frac{\partial s^n}{\partial x^n} \frac{d^2 x^n}{dD_j dD_k} = 0, \tag{39}
$$

with

$$
\mathfrak{d}_{jk} s^n = \frac{\partial^2 s^n}{\partial D_j \partial D_k} + \frac{\partial^2 s^n}{\partial D_j \partial x^n} x_k^n + \frac{\partial^2 s^n}{\partial D_k \partial x^n} x_j^n + \frac{\partial^2 s^n}{\partial (x^n)^2} x_j^n x_k^n. \tag{40}
$$

Similarly, differentiating the flow residual equations (29) gives

$$
\begin{aligned}
\frac{\partial R^n}{\partial q^n} \frac{dq^n}{dD_j} + \frac{\partial R^n}{\partial x^n} \frac{dx^n}{dD_j} &+ \frac{\partial R^n}{\partial q^{n-1}} \frac{dq^{n-1}}{dD_j} + \frac{\partial R^n}{\partial x^{n-1}} \frac{dx^{n-1}}{dD_j} \\
&+ \frac{\partial R^n}{\partial q^{n-2}} \frac{dq^{n-2}}{dD_j} + \frac{\partial R^n}{\partial x^{n-2}} \frac{dx^{n-2}}{dD_j} + \frac{\partial R^n}{\partial D_j} = 0,
\end{aligned} \tag{41}
$$

and differentiating again we obtain,

$$
\begin{aligned}
&\frac{\partial R^n}{\partial q^n} \frac{d^2 q^n}{dD_j dD_k} + \frac{\partial R^n}{\partial x^n} \frac{d^2 x^n}{dD_j dD_k} + \mathfrak{D}_{jk} R_n^n \\
+ &\frac{\partial R^n}{\partial q^{n-1}} \frac{d^2 q^{n-1}}{dD_j dD_k} + \frac{\partial R^n}{\partial x^{n-1}} \frac{d^2 x^{n-1}}{dD_j dD_k} + \mathfrak{D}_{jk} R_{n-1}^n \\
+ &\frac{\partial R^n}{\partial q^{n-2}} \frac{d^2 q^{n-2}}{dD_j dD_k} + \frac{\partial R^n}{\partial x^{n-2}} \frac{d^2 x^{n-2}}{dD_j dD_k} + \mathfrak{D}_{jk} R_{n-2}^n = 0,
\end{aligned} \tag{42}
$$

with

$$\mathfrak{D}_{jk}R^n_n := \frac{\partial^2 R^n}{\partial (q^n)^2} q^n_j q^n_k + \frac{\partial^2 R^n}{\partial (x^n)^2} x^n_j x^n_k + \frac{\partial^2 R^n}{\partial q^n \partial x^n} \left(q^n_j x^n_k + x^n_j q^n_k \right) \\ + \frac{\partial^2 R^n}{\partial D_j \partial q^n} q^n_k + \frac{\partial^2 R^n}{\partial D_k \partial q^n} q^n_j + \frac{\partial^2 R^n}{\partial D_j \partial x^n} x^n_k + \frac{\partial^2 R^n}{\partial D_k \partial x^n} x^n_j + \frac{\partial^2 R^n}{\partial D_j \partial D_k}, \tag{43}$$

$$\begin{aligned}\mathfrak{D}_{jk}R^n_{n-1} := {} & \frac{\partial^2 R^n}{\partial (q^{n-1})^2} q^{n-1}_j q^{n-1}_k + \frac{\partial^2 R^n}{\partial (x^{n-1})^2} x^{n-1}_j x^{n-1}_k \\ + {} & \frac{\partial^2 R^n}{\partial q^{n-1} \partial x^{n-1}} \left(q^{n-1}_j x^{n-1}_k + x^{n-1}_j q^{n-1}_k \right) \\ + {} & \frac{\partial^2 R^n}{\partial q^n \partial q^{n-1}} \left(q^n_j q^{n-1}_k + q^{n-1}_j q^n_k \right) + \frac{\partial^2 R^n}{\partial x^n \partial x^{n-1}} \left(x^n_j x^{n-1}_k + x^{n-1}_j x^n_k \right) \\ + {} & \frac{\partial^2 R^n}{\partial q^n \partial x^{n-1}} \left(q^n_j x^{n-1}_k + x^{n-1}_j q^n_k \right) + \frac{\partial^2 R^n}{\partial x^n \partial q^{n-1}} \left(x^n_j q^{n-1}_k + q^{n-1}_j x^n_k \right) \\ + {} & \frac{\partial^2 R^n}{\partial D_j \partial q^{n-1}} q^{n-1}_k + \frac{\partial^2 R^n}{\partial D_k \partial q^{n-1}} q^{n-1}_j + \frac{\partial^2 R^n}{\partial D_j \partial x^{n-1}} x^{n-1}_k + \frac{\partial^2 R^n}{\partial D_k \partial x^{n-1}} x^{n-1}_j,\end{aligned} \tag{44}$$

and

$$\begin{aligned}\mathfrak{D}_{jk}R^n_{n-2} := {} & \frac{\partial^2 R^n}{\partial (q^{n-2})^2} q^{n-2}_j q^{n-2}_k + \frac{\partial^2 R^n}{\partial (x^{n-2})^2} x^{n-2}_j x^{n-2}_k \\ + {} & \frac{\partial^2 R^n}{\partial q^{n-2} \partial x^{n-2}} \left(q^{n-2}_j x^{n-2}_k + x^{n-2}_j q^{n-2}_k \right) \\ + {} & \frac{\partial^2 R^n}{\partial q^n \partial q^{n-2}} \left(q^n_j q^{n-2}_k + q^{n-2}_j q^n_k \right) + \frac{\partial^2 R^n}{\partial x^n \partial x^{n-2}} \left(x^n_j x^{n-2}_k + x^{n-2}_j x^n_k \right) \\ + {} & \frac{\partial^2 R^n}{\partial q^n \partial x^{n-2}} \left(q^n_j x^{n-2}_k + x^{n-2}_j q^n_k \right) + \frac{\partial^2 R^n}{\partial x^n \partial q^{n-2}} \left(x^n_j q^{n-2}_k + q^{n-2}_j x^n_k \right) \\ + {} & \frac{\partial^2 R^n}{\partial q^{n-1} \partial q^{n-2}} \left(q^{n-1}_j q^{n-2}_k + q^{n-2}_j q^{n-1}_k \right) + \frac{\partial^2 R^n}{\partial x^{n-1} \partial x^{n-2}} \left(x^{n-1}_j x^{n-2}_k + x^{n-2}_j x^{n-1}_k \right) \\ + {} & \frac{\partial^2 R^n}{\partial q^{n-1} \partial x^{n-2}} \left(q^{n-1}_j x^{n-2}_k + x^{n-2}_j q^{n-1}_k \right) + \frac{\partial^2 R^n}{\partial x^{n-1} \partial q^{n-2}} \left(x^{n-1}_j q^{n-2}_k + q^{n-2}_j x^{n-1}_k \right) \\ + {} & \frac{\partial^2 R^n}{\partial D_j \partial q^{n-2}} q^{n-2}_k + \frac{\partial^2 R^n}{\partial D_k \partial q^{n-2}} q^{n-2}_j + \frac{\partial^2 R^n}{\partial D_j \partial x^{n-2}} x^{n-2}_k + \frac{\partial^2 R^n}{\partial D_k \partial x^{n-2}} x^{n-2}_j.\end{aligned} \tag{45}$$

Substituting the adjoint variables from equations (32) and (34) into equation (36) and using equations (39) and (42) to simplify the resulting equations leads to the following expression for the Hessian of the unsteady functional L^g:

$$\frac{d^2 L^g}{dD_j dD_k} = \sum_{n=0}^{N} \left(\mathfrak{D}_{jk} L^n + (\lambda^n)^T \mathfrak{d}_{jk} s^n + (\psi^n)^T \sum_{m=n-2}^{n} \mathfrak{D}_{jk} R^n_m \right). \tag{46}$$

In order to calculate the terms $x^n_j = \frac{dx^n}{dD_j}$ and $q^n_j = \frac{dq^n}{dD_j}$ for $n = 0, \dots, N$ required for $\mathfrak{D}_{jk}L^n$, $\mathfrak{D}_{jk}R^n_m$, and $\mathfrak{d}_{jk}s^n$, one can solve equations (38) and (41) to obtain

$$\frac{dx^n}{dD_j} = -\left(\frac{\partial s^n}{\partial x^n} \right)^{-1} \frac{\partial s^n}{\partial D_j}, \tag{47}$$

and

$$\frac{dq^n}{dD_j} = -\left(\frac{\partial R^n}{\partial q^n} \right)^{-1} \left[\frac{\partial R^n}{\partial x^n} \frac{dx^n}{dD_j} + \frac{\partial R^n}{\partial q^{n-1}} \frac{dq^{n-1}}{dD_j} + \frac{\partial R^n}{\partial x^{n-1}} \frac{dx^{n-1}}{dD_j} \right. \\ \left. + \frac{\partial R^n}{\partial q^{n-2}} \frac{dq^{n-2}}{dD_j} + \frac{\partial R^n}{\partial x^{n-2}} \frac{dx^{n-2}}{dD_j} + \frac{\partial R^n}{\partial D_j} \right]. \tag{48}$$

The computational cost for calculating the unsteady Hessian is

- One time-dependent ($N+1$ (nonlinear) time steps) baseline solution for x^n using the unsteady grid residual equations (28)
- One time-dependent ($N+1$ nonlinear time steps) baseline solution for q^n using the unsteady flow residual equations (29)
- One time-dependent ($N+1$ linear steps) adjoint solution each for ψ^n and λ^n using equations (32) and (34), respectively
- M time-dependent ($N+1$ linear steps) solutions each for $x_j^n = \frac{dx^n}{dD_j}$ and $q_j^n = \frac{dq^n}{dD_j}$ using equations (47) and (48), respectively
- $(N+1) \cdot M(M+1)/2$ cheap evaluations of the right-hand side of equation (46)

IV. Optimization Examples

We consider the steady inviscid flow around a NACA 0012 airfoil, as well as the unsteady case of a sinusoidally pitching airfoil about its quarter-chord location as flow examples which are described in more detail in Mani and Mavriplis.[10] The governing Euler equations of the flow problem are formulated in the arbitrary Lagrangian-Eulerian (ALE) finite volume form and time marching is achieved with the second-order accurate backward difference formula (BDF2). The computational mesh has about 20,000 triangular elements and is shown in Figure 1. The required deformation and movement of the mesh is performed via a linear tension spring analogy.[10,11]

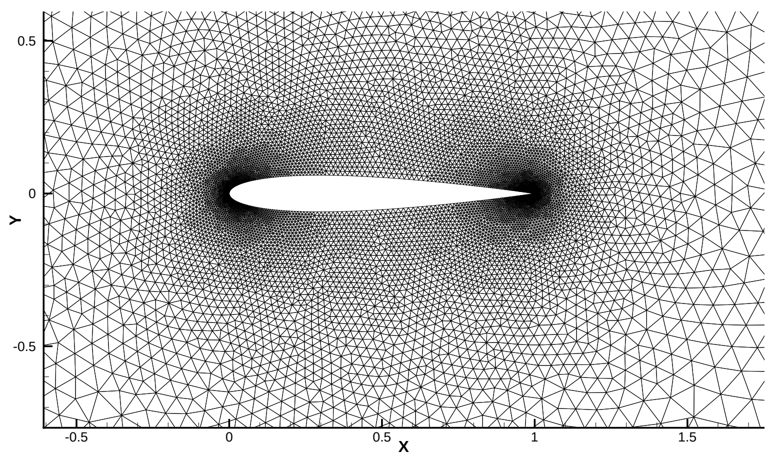

Figure 1. The computational mesh with approximately $20,000$ elements.

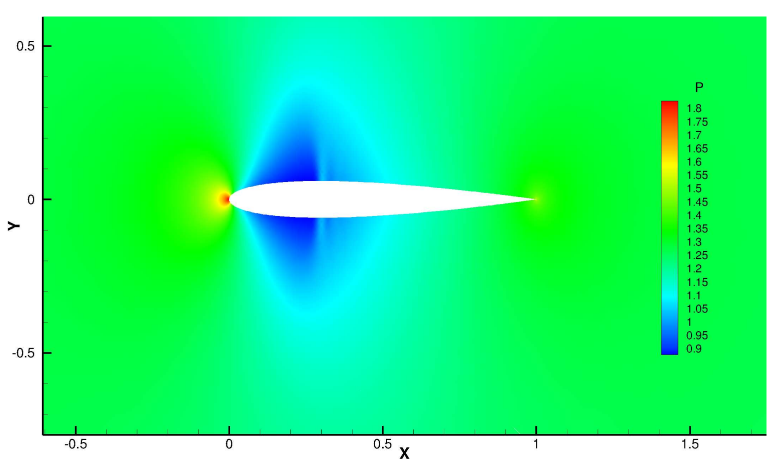

Figure 2. Non-dimensionalized pressure contours for $M_\infty = 0.755$ and $\alpha = 0.016$.

The free-stream Mach number is $M_\infty = 0.755$ with a mean angle of attack of 0.016 degrees. The non-dimensionalized pressure contours for the steady flow at the mean angle of attack are shown in Figure 2. For the unsteady case, identical free-stream conditions are employed, and the time-dependent pitch has an amplitude of 2.51 degrees and a reduced frequency of 0.0814. One pitching period is divided into 32 discrete time steps and the entire simulation consists of $N = 40$ time steps after a steady-state solution with the mean angle of attack. The resulting time-dependent lift and drag profiles are displayed in Figure 3.

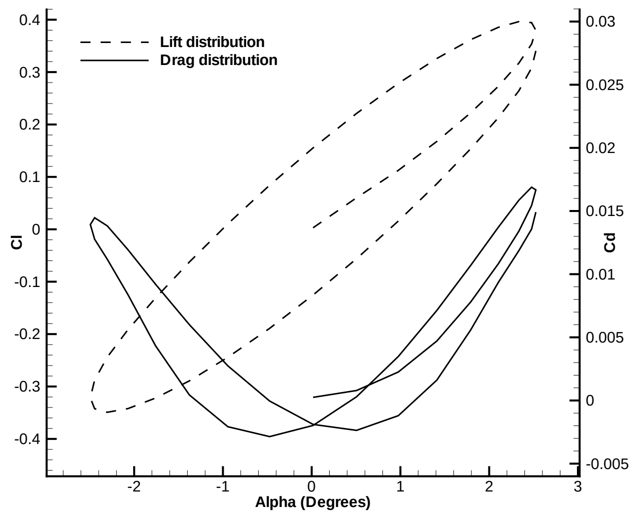

Figure 3. The time-dependent lift and drag profiles for the pitching NACA 0012.

The optimization examples consist of inverse designs given by the following unsteady objective function:

$$L^g(D) = \sum_{n=0}^{N} L^n(q^n(D), x^n(D)) = \sum_{n=0}^{N} \frac{1}{2}(C_l^n - C_l^{*n})^2 + \frac{100}{2}(C_d^n - C_d^{*n})^2, \tag{49}$$

where C_l^n and C_d^n are the lift and drag coefficients at time step n, respectively, a star denotes a target lift or drag coefficient and the factor of one hundred is introduced because the drag coefficient is about an order of magnitude smaller than the lift coefficient in this particular flow example. A steady case objective function is thus simply given by

$$f(D) = F(q(D), x(D)) = \frac{1}{2}(C_l - C_l^*)^2 + \frac{100}{2}(C_d - C_d^*)^2. \tag{50}$$

Both objective functions are always scaled such that their initial value is unity. We use two and six design variables placed at upper and lower surface points which control the magnitude of Hicks-Henne sine bump functions.[12] Note that in this case both L^g (or f) and R^n (or R) are not explicitly dependent on the design variables D which simplifies the equations presented in Sections II and III considerably.

The steady and unsteady inverse designs are initialized with the NACA 0012 airfoil profile and the target coefficients are obtained by perturbing the two and six design variables. The initial and target airfoils are shown in Figures 4 and 5.

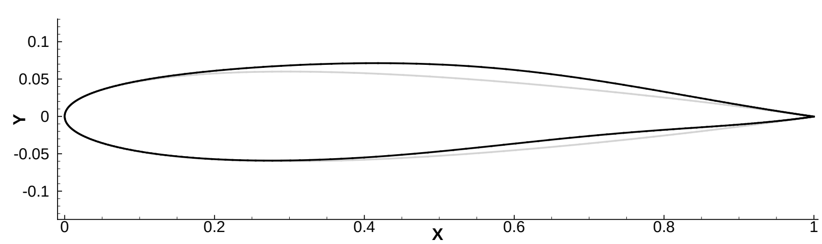

Figure 4. The initial NACA 0012 airfoil (in gray) and the target airfoil obtained through the perturbation of two design variables (in black).

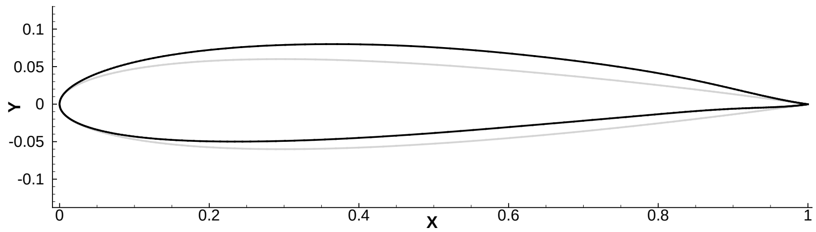

Figure 5. The initial NACA 0012 airfoil (in gray) and the target airfoil obtained through the perturbation of six design variables (in black).

In order to validate the gradient and Hessian calculations we compare the adjoint to finite difference approaches as described in Subsection II.B. If we use second-order finite differences (fd) with a stepsize of $h = 10^{-8}$ to calculate the gradient at the first optimization iteration, for the steady case with two design variables we obtain

$$\left(\frac{df}{dD_j}\right)_{fd} = (-76.26469,\ -68.65205)$$

whereas the adjoint (ad) gradient yields

$$\left(\frac{df}{dD_j}\right)_{ad} = (-76.26471,\ -68.65209).$$

Concurrently, for the second-order finite differenced Hessian with a stepsize of $h = 10^{-5}$ we obtain

$$\left(\frac{d^2 f}{dD_j dD_k}\right)_{fd} = \begin{pmatrix} 3330.738 & 2335.429 \\ 2335.429 & 1699.565 \end{pmatrix}$$

and the adjoint approach results in

$$\left(\frac{d^2 f}{dD_j dD_k}\right)_{ad} = \begin{pmatrix} 3330.736 & 2336.586 \\ 2336.586 & 1699.563 \end{pmatrix}.$$

Thus, both approaches yield very agreeable results. The approximation ($approx$) described in Subsection II.C and given by equation (26) yields

$$\left(\frac{d^2 f}{dD_j dD_k}\right)_{approx} = \begin{pmatrix} 2908.812 & 2619.079 \\ 2619.079 & 2358.785 \end{pmatrix}.$$

Similarly, for the unsteady case with two design variables we have

$$\left(\frac{dL^g}{dD_j}\right)_{fd} = (-75.83811,\ -70.04130)$$

whereas the adjoint yields

$$\left(\frac{dL^g}{dD_j}\right)_{ad} = (-75.83812,\ -70.04131).$$

For the unsteady Hessian with a stepsize of $h = 10^{-6}$ we obtain

$$\left(\frac{d^2 L^g}{dD_j dD_k}\right)_{fd} = \begin{pmatrix} 3606.608 & 2672.880 \\ 2672.880 & 2639.794 \end{pmatrix}$$

which compares reasonably well with the adjoint approach

$$\left(\frac{d^2 L^g}{dD_j dD_k}\right)_{ad} = \begin{pmatrix} 3625.432 & 2672.493 \\ 2672.493 & 2658.547 \end{pmatrix}.$$

Note that the finite difference approach for the calculation of the unsteady Hessian is very sensitive to the chosen stepsize h as can be inferred from Figure 6.

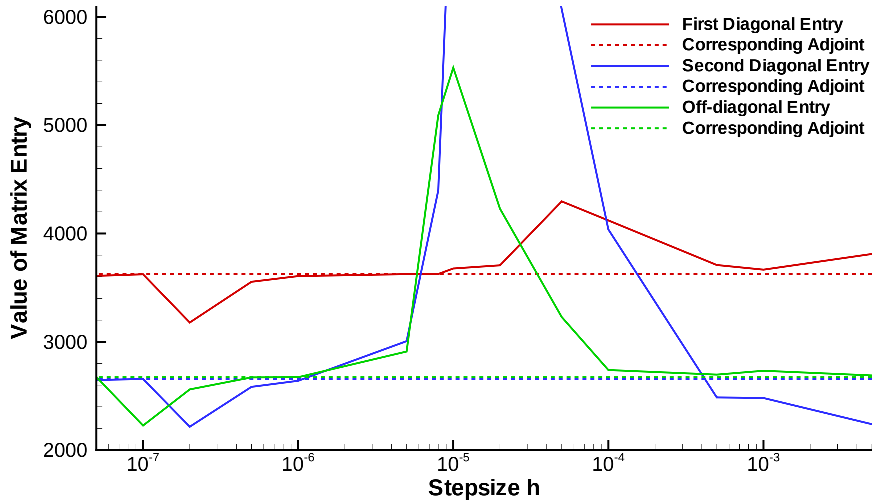

Figure 6. Stepsize sensitivity of the finite differenced entries of the unsteady Hessian for two design variables.

We use two different optimizers for the actual inverse designs: a quasi-Newton optimizer (LBFGS-B[13,14]) which uses only function and gradient evaluations as well as a full Newton optimizer (KNITRO[15]) which additionally requires the evaluation of the Hessian. Both optimizers can handle simple bound constraints on the design variables which must be used to prevent the generation of invalid geometries from the mesh movement algorithm. Figures 7 and 8 show the convergence histories for the steady and unsteady inverse designs using two and six design variables, respectively.

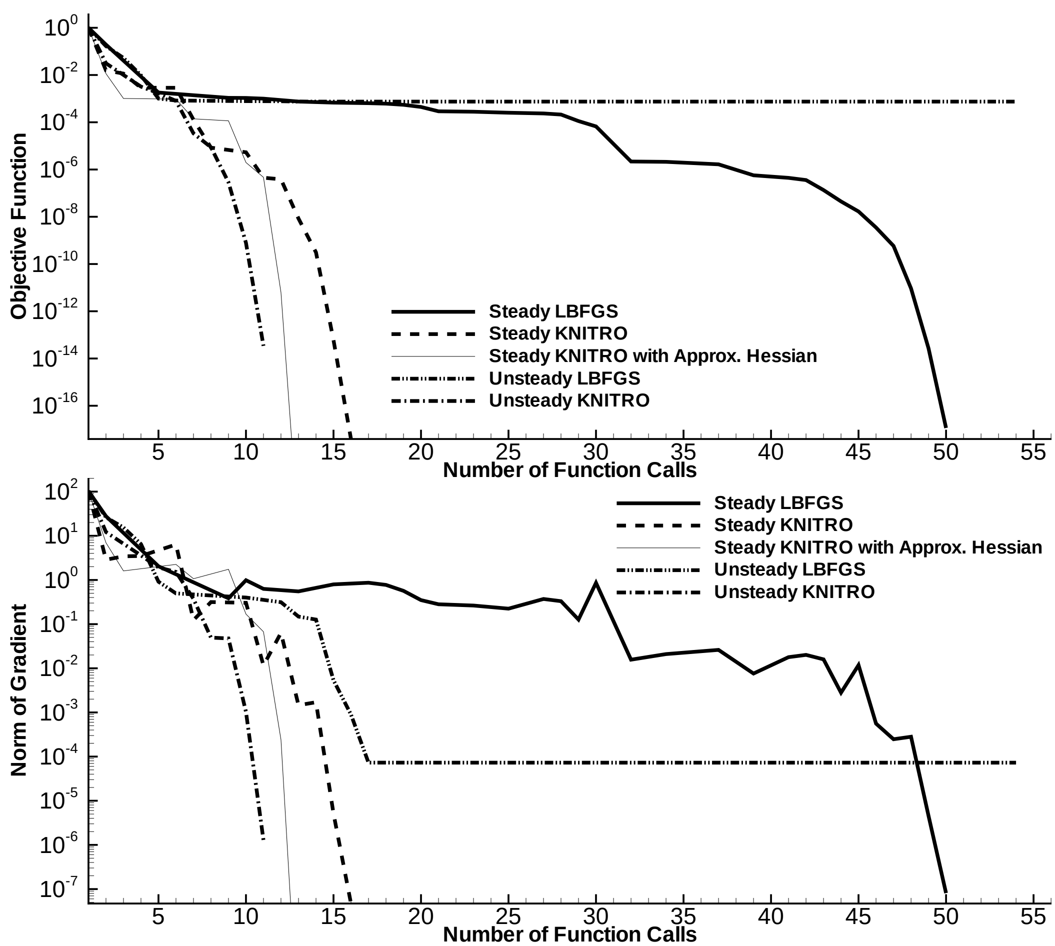

Figure 7. Convergence histories of the steady and unsteady inverse designs using two design variables.

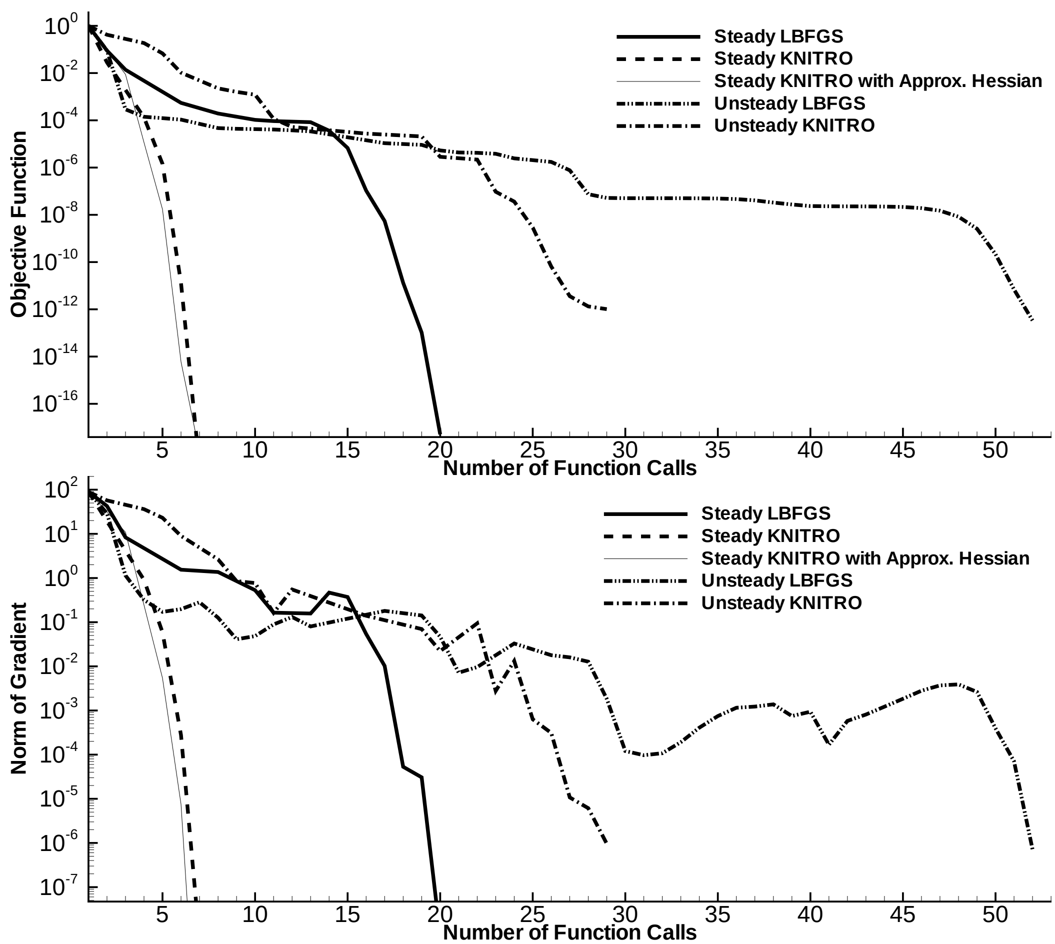

Figure 8. Convergence histories of the steady and unsteady inverse designs using six design variables.

Note that for the LBFGS-B optimizer the number of gradient calls is equal to the number of function calls and that the line search algorithm stalls for the unsteady inverse design using two design variables after the objective function is reduced by about three orders of magnitude. For the two design variable case KNITRO required 14 and 10 gradient calls and 13 and 10 Hessian calls for the steady and unsteady optimization case, respectively. For the six design variable case 6 and 21 gradient calls and 5 and 20 Hessian calls were required. All examples show that it can be very beneficial in terms of computational cost to use the Hessian information for optimization, at least if only a few design variables are involved since the cost of computing the Hessian grows linearly with the number of design variables. As can be inferred from the two figures, the use of the approximate Hessian as described in Subsection II.C for steady inverse designs is a very promising technique since the cost of evaluating this approximate Hessian does not increase with the number of design variables.

V. Extrapolation and Uncertainty Analysis

Another useful application of the Hessian is for extrapolation as discussed in Ghate and Giles.[2] The extrapolated function values can, for example, be used for an inexpensive Monte Carlo (IMC) simulation[16] for uncertainty analysis since it is much cheaper to extrapolate the function value than to perform a full nonlinear function evaluation for every sampling point. Uncertainty analysis is important since high fidelity computations typically assume perfect knowledge of all parameters. In reality, however, there is much uncertainty due to manufacturing tolerances,[17] in-service wear-and-tear, and approximate modeling parameters[18] which one should account for.

V.A. Basic Extrapolation

For our particular example, one shape design variable on the upper surface is varied from -1.2×10^{-2} to 1.2×10^{-2} in steps of 5×10^{-4} around the base solution of a NACA 0012 airfoil corresponding to a design variable value of $D_0 = 0.0$ (see Figure 9).

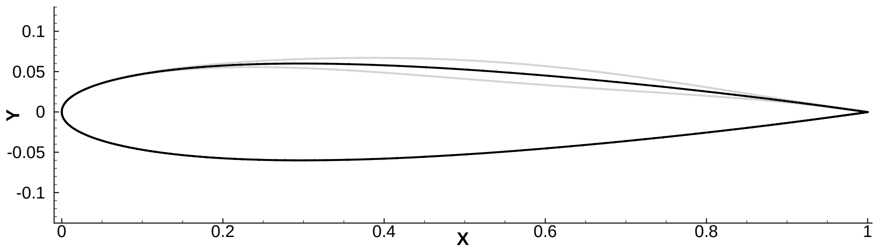

Figure 9. The baseline NACA 0012 (in black) and the upper and lower bounds (in gray) for the one design variable variation.

We compare linear, quadratic and adjoint corrected linear extrapolation as well as adjoint corrected function evaluations of linearly extrapolated terms with the full nonlinear solutions of two different cases:

1. The steady flow problem described in the previous section with objective function $\mathcal{J}(D) := C_l$
2. The unsteady flow problem described in the previous section but only using five time steps rather than fourty with objective function $\mathcal{J}(D) := \frac{1}{6}\sum_{n=0}^{5} C_l^n$

The linear extrapolation (Lin) is given by

$$\mathcal{J}_{\text{Lin}}\Big(D, x(D), q(D)\Big) = \mathcal{J}\Big(D_0, x(D_0), q(D_0)\Big) + \left.\frac{d\mathcal{J}}{dD}\right|_{D_0} \cdot (D - D_0), \tag{51}$$

the quadratic extrapolation (Quad) is

$$\mathcal{J}_{\text{Quad}}\Big(D, x(D), q(D)\Big) = \mathcal{J}_{\text{Lin}}\Big(D, x(D), q(D)\Big) + \frac{1}{2}\left.\frac{d^2\mathcal{J}}{dD^2}\right|_{D_0} \cdot (D - D_0)^2, \tag{52}$$

the adjoint corrected linear extrapolation (ACLin) is

$$\begin{aligned} \mathcal{J}_{\text{ACLin}}\Big(D, x(D), q(D)\Big) &= \mathcal{J}_{\text{Lin}}\Big(D, x(D), q(D)\Big) + \lambda_{D_0}^T \cdot s\Big(D, x(D_0) + \left.\frac{dx}{dD}\right|_{D_0} \cdot (D - D_0)\Big) \\ &+ \psi_{D_0}^T \cdot R\Big(D, x(D_0) + \left.\frac{dx}{dD}\right|_{D_0} \cdot (D - D_0), q(D_0) + \left.\frac{dq}{dD}\right|_{D_0} \cdot (D - D_0)\Big), \end{aligned} \tag{53}$$

and the adjoint corrected function evaluation of linearly extrapolated terms (ACLT) is

$$\begin{aligned} \mathcal{J}_{\text{ACLT}}\Big(D, x(D), q(D)\Big) &= \mathcal{J}\Big(D, x(D_0) + \left.\frac{dx}{dD}\right|_{D_0} \cdot (D - D_0), q(D_0) + \left.\frac{dq}{dD}\right|_{D_0} \cdot (D - D_0)\Big) \\ &+ \lambda_{D_0}^T \cdot s\Big(D, x(D_0) + \left.\frac{dx}{dD}\right|_{D_0} \cdot (D - D_0)\Big) \\ &+ \psi_{D_0}^T \cdot R\Big(D, x(D_0) + \left.\frac{dx}{dD}\right|_{D_0} \cdot (D - D_0), q(D_0) + \left.\frac{dq}{dD}\right|_{D_0} \cdot (D - D_0)\Big). \end{aligned} \tag{54}$$

One could also use an adjoint corrected function evaluation of constant terms (ACCT) given by

$$\begin{aligned} \mathcal{J}_{\text{ACCT}}\Big(D, x(D), q(D)\Big) &= \mathcal{J}\Big(D, x(D_0), q(D_0)\Big) + \lambda_{D_0}^T \cdot s\Big(D, x(D_0)\Big) \\ &+ \psi_{D_0}^T \cdot R\Big(D, x(D_0), q(D_0)\Big). \end{aligned} \tag{55}$$

However, in our particular case neither $\mathcal{J}$ nor R are explicitly dependent on the design variable D and s is linear in D which means that this approach is exactly equal to the linear extrapolation and is thus omitted.

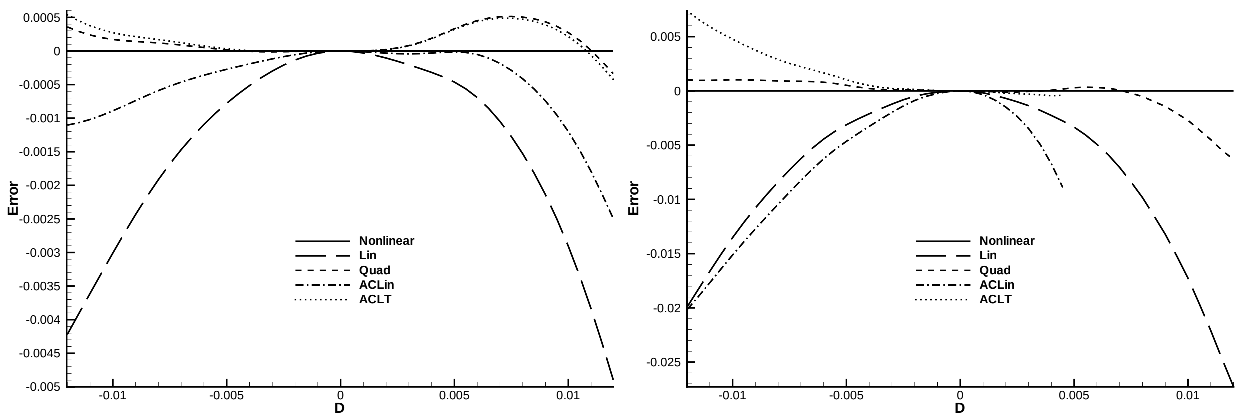

Figure 10. Error for the various extrapolation methods (left: steady case, right: unsteady case).

Figure 10 shows the errors between these extrapolations and the actual values of the objective functions against the variation in the design variable value.

Overall, the quadratic extrapolation performs best and is essentially equivalent in cost to the adjoint corrected linear extrapolation (ACLin) and the adjoint corrected function evaluation of linearly extrapolated terms (ACLT) since calculating the terms $\frac{dx}{dD}\Big|_{D_0}$ and $\frac{dq}{dD}\Big|_{D_0}$ comprises the majority of the cost for all three approaches. One can also see the quadratic error behavior for the linear extrapolation and the higher order error behavior for all the other extrapolation methods. Note that in the unsteady case the adjoint corrected approaches have difficulties with the evaluation of the flow residual for larger positive perturbations which result in "not a number". Lastly, the excellent agreements between the extrapolated and actual lift coefficients for small perturbations are displayed in Figure 11.

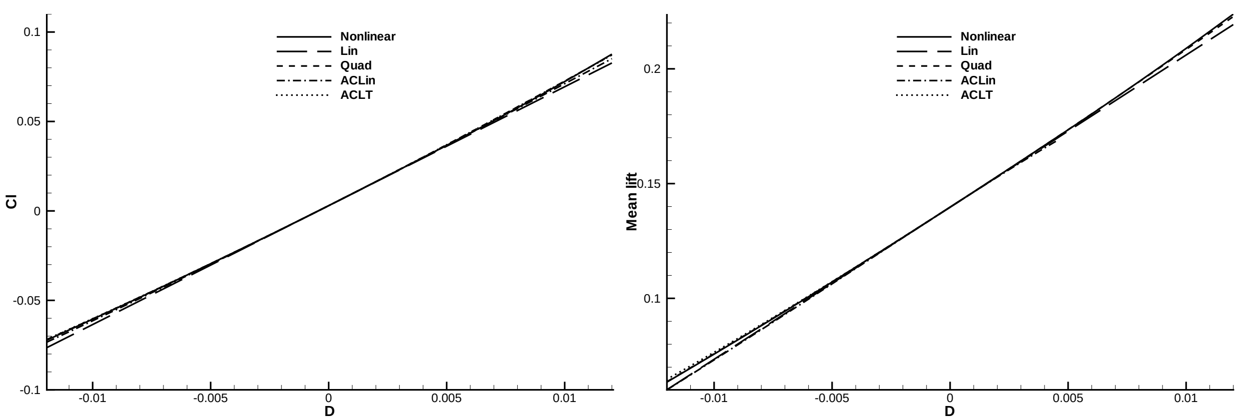

Figure 11. Comparison between the various extrapolation methods and the actual objective function values (left: steady case, right: unsteady case).

V.B. Uncertainty Analysis

The easiest and most accurate method for uncertainty analysis is a full nonlinear MC simulation[16] which is still prohibitively expensive for high fidelity computations. If one is only interested in the mean and standard deviation of a random variable, moment methods can be a good choice.[1,19] Unfortunately, higher order moment methods require the computation of higher derivatives and no information about the probability density function (PDF) is obtained. Moment methods are based on Taylor series expansions of the original nonlinear objective function $\mathcal{J}(D)$ about the mean of the input D_0 given standard deviations σ_{D_j}. The resulting mean $\mu_{\mathcal{J}}$ and standard deviation $\sigma_{\mathcal{J}}$ of the objective function are given to first order (MM1) by

$$\mu_{\mathcal{J}}^{(1)} = \mathcal{J}(D_0) \tag{56}$$

$$\sigma_{\mathcal{J}}^{(1)} = \sum_{j=1}^{M}\left(\left.\frac{d\mathcal{J}}{dD_j}\right|_{D_0}\sigma_{D_j}\right)^2, \tag{57}$$

and to second order (MM2) by

$$\mu_{\mathcal{J}}^{(2)} = \mu_{\mathcal{J}}^{(1)} + \frac{1}{2}\sum_{j=1}^{M}\left(\left.\frac{d^2\mathcal{J}}{dD_j^2}\right|_{D_0}\sigma_{D_j}^2\right) \tag{58}$$

$$\sigma_{\mathcal{J}}^{(2)} = \sigma_{\mathcal{J}}^{(1)} + \frac{1}{2}\sum_{j=1}^{M}\sum_{k=1}^{M}\left(\left.\frac{d^2\mathcal{J}}{dD_j dD_k}\right|_{D_0}\sigma_{D_j}\sigma_{D_k}\right)^2. \tag{59}$$

As already mentioned in the introduction of this section, extrapolation can be used for an IMC simulation with the advantage of being much cheaper than a full nonlinear MC simulation while still being able to obtain an approximate PDF. For the extrapolation, all the methods presented in Subsection V.A can theoretically be applied. In practice however, the adjoint corrected approaches have difficulties with the evaluation of the flow residual for large perturbations as discussed in the previous subsection. Another promising approach for an IMC simulation is to use a hybrid of extrapolation and interpolation involving a few data points $D_i,\ i = 0, \ldots, I$. The function values and the available derivatives at each data point are used to construct an extrapolating function. At the point of evaluation D the extrapolations from all data points are then weighted with a radial basis function (RBF) interpolant. This approach has been coined Dutch Intrapolation[20] (DI) and it has been shown that the order of accuracy of the intrapolant is equal to its polynomial order, which is the highest order of accuracy that can be obtained. The Dutch extrapolation functions are normal multivariate Taylor expansions of order n with a correction term given in multi-index notation by[20]

$$\mathcal{T}^n(D, D_i) = \sum_{|k|\geq 0}^{|k|\leq n} \frac{a_k^n}{k!}(D - D_i)^k \partial^k \mathcal{J}(D_i) \quad \text{for} \quad i = 0, \ldots, I \tag{60}$$

with $a_k^n = 1 - k/(n+1)$. The solution of an interpolation problem using RBFs is given in the form[21]

$$\mathcal{J}_{\text{DI}}(D) = \sum_{i=0}^{I} \beta_i \phi(||D - D_i||) + p(D). \tag{61}$$

Here, $\mathcal{J}_{\text{DI}}(D)$ is the interpolated function value at location D, ϕ is the adopted form of basis function (see Wendland[22] for options), and the D_i are the locations of the centers for the RBFs. In this work $\phi(||D - D_i||) = \phi_{D,D_i} = ||D - D_i||^3$ has been found to produce good quality results. $p(D)$ is an added polynomial term to give the interpolation an underlying trend, and up to linear polynomials are added here to ensure that translations and rotations are recovered. The coefficients β_i are found by requiring exact recovery of the original function; in our case the extrapolated function values $\mathcal{T}^n(D, D_i)$ given by equation (60). When the polynomial term is included, the linear system to be solved is completed by the additional "side condition"

$$\sum_{i=0}^{I} \beta_i p'(D) = 0 \tag{62}$$

for all polynomials $p'(D)$ with degree less than or equal to that of $p(D)$. It is important to note that although the Dutch Taylor expansions are discussed here for general order n, practical applications are usually restricted to low values of n. The range of practical applicability is similar to that of "normal" Taylor expansions. High order Taylor expansions are often used in theoretical formulations, however, in practical applications their use is limited because the convergence with increasing order is typically very slow, and the region of convergence very small. Thus, the Dutch Taylor expansions are to be used in small regions where the function to be approximated is well represented by a low order polynomial, that is where the Taylor expansion coefficients decrease quickly for increasing order. In this paper we use only up to first-order terms in the Dutch Taylor expansions.

As a test case we allow one shape design variable on the upper surface and one on the lower surface to vary in the same unsteady flow problem as described in the optimization section. The two design variables are treated as random variables with normal distribution. The mean is set to zero (corresponding to the NACA 0012 airfoil) and the standard deviations are taken to be $\sigma_{D_1} = \sigma_{D_2} = 0.01$. Figure 12 shows the NACA 0012 airfoil and the airfoils resulting from perturbations of $\pm\sigma_{D_j}$.

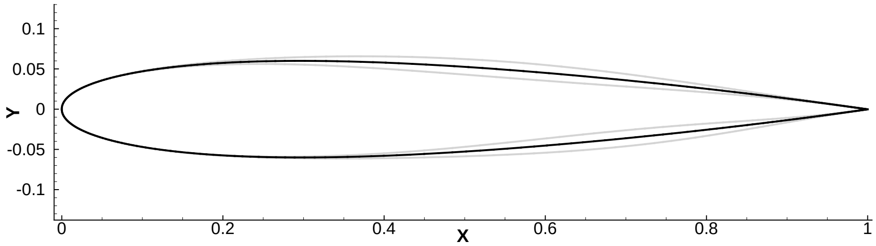

Figure 12. The NACA 0012 airfoil (in black) and airfoils resulting from perturbations of $\pm\sigma_{D_j}$ (in gray).

We use five data points for the Dutch intrapolation forming a square around the center given by the mean value $D_0 = (0.0, 0.0)$. The four corners are $D_1 = (-\sigma_{D_1}, -\sigma_{D_2})$, $D_2 = (\sigma_{D_1}, -\sigma_{D_2})$, $D_3 = (\sigma_{D_1}, \sigma_{D_2})$, and $D_4 = (-\sigma_{D_1}, \sigma_{D_2})$. Thus, the intrapolated function for each new point $D = (D_x, D_y)$ is given by

$$\mathcal{J}_{\text{DI}}(D) = (1\ D_x\ D_y\ \phi_{D,D_0}\ \phi_{D,D_1}\ \phi_{D,D_2}\ \phi_{D,D_3}\ \phi_{D,D_4}) \cdot a_s \qquad \text{with} \qquad C_{ss} a_s = F_s$$

where

$$a_s = \begin{pmatrix} \gamma_0 \\ \gamma_x \\ \gamma_y \\ \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_4 \end{pmatrix} \qquad F_s = \begin{pmatrix} 0 \\ 0 \\ 0 \\ \mathcal{T}^1(D, D_0) \\ \mathcal{T}^1(D, D_1) \\ \mathcal{T}^1(D, D_2) \\ \mathcal{T}^1(D, D_3) \\ \mathcal{T}^1(D, D_4) \end{pmatrix} \qquad C_{ss} = \begin{pmatrix} 0 & 0 & 0 & 1 & \cdots & 1 \\ 0 & 0 & 0 & D_{0,x} & \cdots & D_{4,x} \\ 0 & 0 & 0 & D_{0,y} & \cdots & D_{4,y} \\ 1 & D_{0,x} & D_{0,y} & \phi_{D_0,D_0} & \cdots & \phi_{D_0,D_4} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & D_{4,x} & D_{4,y} & \phi_{D_4,D_0} & \cdots & \phi_{D_4,D_4} \end{pmatrix}$$

and $p(D) = \gamma_0 + \gamma_x D_x + \gamma_y D_y$. Stratified sampling with a sample size of $10,000$ is used. One flow solve takes about 15 minutes on four AMD processors with 2 GHz each and the adjoint solve for the gradient as well as the forward solves for each design variable for the Hessian calculation take about the same time. Comparisons of the mean and standard deviation predictions of the objective function (time-averaged lift) using the various methods as well as approximate running times are displayed in Table 1.

Table 1. Comparison of Mean and Standard deviation predictions.

	Mean	Standard deviation	Run time (minutes)
Nonlinear	5.55×10^{-2}	1.07×10^{-2}	$150,000$
MM1	5.81×10^{-2}	1.05×10^{-2}	30
MM2	5.39×10^{-2}	1.05×10^{-2}	60
Lin	5.82×10^{-2}	1.05×10^{-2}	30
Quad	5.39×10^{-2}	1.06×10^{-2}	60
DI	5.66×10^{-2}	1.13×10^{-2}	150

The 99 per cent confidence interval for the mean calculated with the full nonlinear MC simulation is $[5.52 \times 10^{-2}, 5.58 \times 10^{-2}]$. As can be seen MM1 and Lin yield very similar results as expected from the leading error. Also, MM2 and Quad give similar results for the same reason. Overall, the Dutch Intrapolation is the closest to the full nonlinear MC simulation results and it is beneficial to invest the extra time in calculating the additional function and gradient values. Finally, as can be seen in Figure 13 the IMC methods capture the actual histograms and consequently PDFs of the time-averaged lift distribution quite well.

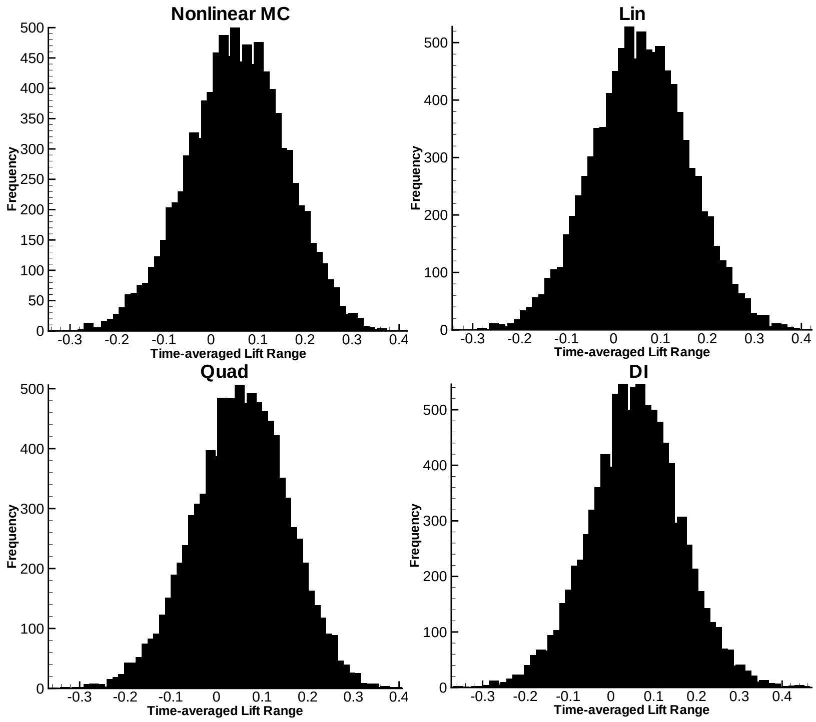

Figure 13. Histograms for time-averaged lift perturbations using various methods.

VI. Conclusions

An efficient general algorithm to calculate the Hessian of a steady or unsteady functional of interest in the context of computational fluid dynamics has been described, validated, and applied to an aerodynamic optimization problem and to an extrapolation of a functional of interest. The extrapolation, in turn, has been successfully applied to inexpensive Monte Carlo simulations which yield a good estimate for the mean and standard deviation of a time-averaged lift distribution as well as the probability density function for a fraction of the cost of a full nonlinear Monte Carlo simulation. The validation and optimization results show that this algorithm is accurate, effective, and efficient for practical applications. The concepts presented in this paper are very general and easily allow the extension to more sophisticated, higher-order time-marching methods for the unsteady CFD simulation. Lastly, the algorithm can be applied to different flow solvers and to three dimensions in space in a straightforward manner.

Acknowledgments

This work was partially supported by the US Air Force Office of Scientific Research under AFOSR Grant number FA9550-07-1-0164. We are also very grateful to Karthik Mani for making his flow and adjoint solver available to us.

References

[1]Sherman, L. L., Taylor III, A. C., Green, L. L., and Newman, P. A., "First- and second-order aerodynamic sensitivity derivatives via automatic differentiation with incremental iterative methods," *Journal of Computational Physics*, Vol. 129, 1996, pp. 307 – 331.

[2]Ghate, D. P. and Giles, M. B., "Efficient Hessian Calculation using Automatic Differentiation," AIAA, 2007-4059, 2007.

[3]Tortorelli, D. and Michaleris, P., "Design sensitivity analysis: overview and review," *Inverse Problems Eng*, Vol. 1, 1994, pp. 71 – 105.

[4]Papadimitriou, D. I. and Giannakoglou, K. C., "Computation of the Hessian matrix in aerodynamic inverse design using continuous adjoint formulations," *Computers and Fluids*, Vol. 37, 2008, pp. 1029 – 1039.

[5]Greiwank, A., *Evaluating Derivatives*, SIAM, Frontiers in Applied Mathematics, 2000.

[6]Hascoët, L., "TAPENADE: a tool for Automatic Differentiation of programs," *Proceedings of 4^{th} European Congress on*

Computational Methods, ECCOMAS'2004, Jyvaskyla, Finland, 2004.

[7]Christianson, B., "Automatic Hessians by reverse accumulation," *IMA Journal of Numerical Analysis*, Vol. 12, 1992, pp. 135 – 150.

[8]Rumpfkeil, M. P. and Zingg, D. W., "A General Framework for the Optimal Control of Unsteady Flows with Applications," AIAA, 2007-1128, 2007.

[9]Rumpfkeil, M. P. and Zingg, D. W., "The Optimal Control of Unsteady Flows with a Discrete Adjoint Method," *Optimization and Engineering*, doi:10.1007/s11081-008-9035-5, 2008.

[10]Mani, K. and Mavriplis, D. J., "Unsteady Discrete Adjoint Formulation for Two-Dimensional Flow Problems with Deforming Meshes," *AIAA Journal*, Vol. 46 No. 6, 2008, pp. 1351–1364.

[11]Batina, J. T., "Unsteady Euler Airfoil Solutions Using Unstructured Dynamic Meshes," *AIAA Journal*, Vol. 28, No. 8, 1990, pp. 1381 – 1388.

[12]Hicks, R. and Henne, P., "Wing Design by Numerical Optimization," *Journal of Aircraft*, Vol. 15, No. 7, 1978, pp. 407 – 412.

[13]Byrd, R. H., Lu, P., Nocedal, J., and Zhu, C., "A Limited Memory Algorithm for Bound Constrained Optimization," *SIAM Journal on Scientific Computing 16*, Vol. 5, 1995, pp. 1190–1208.

[14]Zhu, C., Byrd, R. H., Lu, P., and Nocedal, J., "L-BFGS-B: A Limited Memory FORTRAN Code for Solving Bound Constrained Optimization Problems," Tech. Rep. NAM-11, EECS Department, Northwestern University, 1994.

[15]Byrd, R. H., Nocedal, J., and Waltz, R. A., "KNITRO: An Integrated Package for Nonlinear Optimization," *Large-Scale Nonlinear Optimization, di Pillo, G., Roma, M.(eds)*, Springer-Verlag, pp. 35-59, 2006.

[16]Ghate, D. and Giles, M. B., "Inexpensive Monte Carlo uncertainty analysis," *Recent Trends in Aerospace Design and Optimization, Tata McGraw-Hill, New Delhi*, 2006, pp. 203–210.

[17]Gumbert, C. R., Newman, P. A., and Hou, G. J., "Effect of random geometric uncertainty on the computational design of 3-D wing," AIAA, 2002-2806, June 2002.

[18]Luckring, J. M., Hemsch, M. J., and Morrison, J. H., "Uncertainty in computational aerodynamics," AIAA, 2003-0409, Jan. 2003.

[19]Putko, M. M., Newmann, P. A., Taylor III, A. C., and Green, L. L., "Approach for uncertainty propagation and robust design in CFD using sensitivity derivatives," AIAA, 2001-2528, 2001.

[20]Kraaijpoel, D. A., *Seismic ray fields and ray field maps: theory and algorithms*, Ph.D. thesis, Universiteit Utrecht, 2003.

[21]Buhmann, M., *Radial Basis Functions (1st edn)*, Cambridge University Press: Cambridge, 2005.

[22]Wendland, H., *Scattered Data Approximation (1st edn)*, Cambridge University Press: Cambridge, 2005.