Numerical Computation of Second Derivatives[1] with Applications to Optimization Problems

Philip Caplan – pcaplan@mit.edu

Abstract

Newton's method is applied to the minimization of a computationally expensive objective function. Various methods for computing the exact Hessian are examined, notably adjoint-based methods and the hyper-dual method. The hyper-dual number method still requires $\mathcal{O}(N^2)$ function evaluations to compute the exact Hessian during each optimization iteration. The adjoint-based methods all require $\mathcal{O}(N)$ evaluations. In particular, the fastest method is the direct-adjoint method as it requires N evaluations as opposed to the adjoint-adjoint and adjoint-direct methods which both require $2N$ evaluations. Applications to a boundary-value problem are presented.

Index Terms

Second derivative, Hessian, adjoint, direct, hyper-dual, optimization

I. Introduction

NEWTON'S method is a powerful optimization algorithm which is well known to converge quadratically upon the root of the gradient of a given objective function. The drawback of this method, however, is the calculation of the second derivative matrix of the objective function, or Hessian. When the objective function is computationally expensive to evaluate, the adjoint method is suitable to compute its gradient. Such expensive function evaluations may, for example, depend on some nonlinear partial differential equation such as those enountered in aerodynamics [7], [12]. With this gradient information, steepest-descent or conjugate-gradient algorithms can then be applied to the optimization problem; however, convergence will often be slow [9].

Quasi-Newton methods will successively approximate the Hessian matrix with either Broyden-Fletcher-Goldfarb-Shanno (BFGS) or Davidon-Fletcher-Powell (DFP) updates [13] which can dramatically improve convergence. However, this enhanced convergence is dependent on the line search algorithm used to find the optimal step in the search direction; this translates to an increased number of function evaluations to satisfy the first Wolfe condition and gradient evaluations to satisfy the second Wolfe condition.

The goal of this paper is to examine various methods used to numerically compute the Hessian matrix. Approximate methods include finite difference or complex-step techniques [10],[15]. Johnson also presents a method of computing second derivatives with the Fast-Fourier Transform [8]. Exact methods include the use of hyper-dual numbers [2],[3],[4],[5],[6] which requires $\mathcal{O}(N^2)$ function evaluations. Papadimitriou and Giannakoglou examine adjoint and direct methods for exactly computing the Hessian matrix [14]. This paper largely follows the methods presented by the latter authors with the exception that the direct-direct method is not examined since it requires $\mathcal{O}(N^2)$ equivalent function evaluations. Instead, the adjoint-adjoint, adjoint-direct and direct-adjoint methods are presented as they all provide the *exact* Hessian in $\mathcal{O}(N)$ equivalent function evaluations. The hyper-dual method is also presented since it is a fair comparison to the adjoint-based methods for computing exact Hessians.

The function, here, is obtained from discretizing a linear differential equation. The theory presented in Section 2, however, is more generally derived for nonlinear systems of equations. A few terms, notably the second-order partial derivatives of the residual equation, vanish in the case of a linear operator.

[1] 18.335 Final Project submitted to Professor Steven G. Johnson, Massachusetts Institute of Technology, 12 December 2012

II. Computing the Hessian Matrix

A. Motivation

Suppose we want to solve the unconstrained optimization problem

$$\min_{\mathbf{x}} \quad F(\mathbf{u}(\mathbf{x}), \mathbf{x}) \tag{1}$$

where $\mathbf{u} \in \mathbb{R}^M$ is the vector of state variables, obtained from a system of M linear or nonlinear equations,

$$\mathcal{R}_m(\mathbf{u}(\mathbf{x}), \mathbf{x}) = 0 \qquad m \in [1, M] \tag{2}$$

and $\mathbf{x} \in \mathbb{R}^N$ is the vector of design variables. Newton's method requires the computation of the gradient and Hessian of the objective function. The components of the gradient, $\mathbf{g} \in \mathbb{R}^N$, are given by

$$g_i = \frac{\mathrm{d}F}{\mathrm{d}x_i} = \frac{\partial F}{\partial x_i} + \frac{\partial F}{\partial u_k}\frac{\mathrm{d}u_k}{\mathrm{d}x_i} \qquad k \in [1, M] \tag{3}$$

The entries of the Hessian, $\mathbf{H} \in \mathbb{R}^{N \times N}$, are given by

$$h_{i,j} = \frac{\mathrm{d}^2 F}{\mathrm{d}x_i \mathrm{d}x_j} = \frac{\partial^2 F}{\partial x_i \partial x_j} + \frac{\partial^2 F}{\partial x_i \partial u_k}\frac{\mathrm{d}u_k}{\mathrm{d}x_j} + \frac{\mathrm{d}^2 F}{\mathrm{d}u_k \mathrm{d}x_j}\frac{\mathrm{d}u_k}{\mathrm{d}x_i} + \frac{\mathrm{d}^2 F}{\mathrm{d}u_k \mathrm{d}u_m}\frac{\mathrm{d}u_k}{\mathrm{d}x_i}\frac{\mathrm{d}u_m}{\mathrm{d}x_j} + \frac{\partial F}{\partial u_k}\frac{\mathrm{d}^2 u_k}{\mathrm{d}x_i \mathrm{d}x_j} \qquad k, m \in [1, M] \tag{4}$$

A Newton-based optimization algorithm can be performed by updating the design vector at each iteration as

$$\mathbf{x}^{t+1} = \mathbf{x}^t + \Delta\mathbf{x}^t \tag{5}$$

where $\Delta\mathbf{x}^t$ is obtained by solving

$$\mathbf{H}^t \Delta\mathbf{x}^t = -\mathbf{g}^t \tag{6}$$

In the steepest descent method, the Hessian reverts back to the $N \times N$ identity matrix. In quasi-Newton methods, the Hessian is updated with a rank-one outer product at each optimization iteration. Here, we will explore some methods for computing the exact Hessian.

B. Adjoint and direct methods for computing gradients

The adjoint and direct methods for computing exact gradients are a prerequisite for the adjoint and direct methods for computing exact Hessians and deserve a brief introduction. Before deriving these methods, first consider the first and second total derivatives of the primal residual, which are also equal to zero:

$$\frac{\mathrm{d}\mathcal{R}_n}{\mathrm{d}x_i} = \frac{\partial \mathcal{R}_n}{\partial x_i} + \frac{\partial \mathcal{R}_n}{\partial u_k}\frac{\mathrm{d}u_k}{\mathrm{d}x_i} = 0 \qquad k, n \in [1, M] \tag{7}$$

$$\frac{\mathrm{d}^2\mathcal{R}_n}{\mathrm{d}x_i \mathrm{d}x_j} = \frac{\partial^2 \mathcal{R}_n}{\partial x_i \partial x_j} + \frac{\partial^2 \mathcal{R}_n}{\partial x_i \partial u_k}\frac{\mathrm{d}u_k}{\mathrm{d}x_j} + \frac{\partial^2 \mathcal{R}_n}{\partial u_k \partial x_j}\frac{\mathrm{d}u_k}{\mathrm{d}x_i} + \frac{\partial^2 \mathcal{R}_n}{\partial u_k \partial u_m}\frac{\mathrm{d}u_k}{\mathrm{d}x_i}\frac{\mathrm{d}u_m}{\mathrm{d}x_j} + \frac{\partial \mathcal{R}_n}{\partial u_k}\frac{\mathrm{d}^2 u_k}{\mathrm{d}x_i \mathrm{d}x_j} = 0 \qquad k, m, n \in [1, M] \tag{8}$$

In the adjoint formulation, a multiple of eq.(7) is added to eq.(3) to produce an augmented functional, $\mathcal{I}$. The terms in this functional are then rearranged so that the sensitivity of the solution vector on the design variables, $\mathrm{d}\mathbf{u}/\mathrm{d}\mathbf{x}$, is removed from the gradient computation. Denote the multiplier as $\boldsymbol{\psi} \in \mathbb{R}^M$; the adjoint equation becomes

$$\mathcal{R}_k^{\psi} = \frac{\partial F}{\partial u_k} + \psi_m \frac{\partial \mathcal{R}_m}{\partial u_k} = 0 \qquad k, m \in [1, M] \tag{9}$$

The gradient of the functional is computed from

$$\frac{\mathrm{d}\mathcal{I}}{\mathrm{d}x_i} = \frac{\partial F}{\partial x_i} + \psi_m \frac{\partial \mathcal{R}_m}{\partial x_i} \qquad m \in [1, N] \tag{10}$$

The advantage of the adjoint method lies in the fact that only one system solution is required to obtain $\boldsymbol{\psi}$, thus providing an inexpensive method for computing the gradient.

The direct method involves *directly* computing $\mathrm{d}\mathbf{u}/\mathrm{d}\mathbf{x} \in \mathbb{R}^{M\times N}$ from eq.(7) and substituting it into eq.(3). The term $\partial\mathcal{R}/\partial\mathbf{x}$ is an $M \times N$ matrix which means there are N right-hand sides to accompany each i^{th} unknown sensitivity vector, resulting in N system solutions. The direct method is unsuitable for a single objective function; however, it becomes practical if the number of objective functions exceeds the number of design variables, in which case the computed $\mathrm{d}\mathbf{u}/\mathrm{d}\mathbf{x}$ sensitivity matrix can be recycled during the gradient computation. Although only one objective function is used in this paper, the direct method can be used efficiently when computing the Hessian, as will be shown later.

C. Adjoint-Adjoint method

The first method to compute the Hessian matrix extends the above adjoint formulation by introducing new adjoint variables and is referred to as the adjoint-adjoint method. Following [14], the original Hessian is augmented with two adjoint matrices $\boldsymbol{\mu} \in \mathbb{R}^{N\times M}$ and $\boldsymbol{\nu} \in \mathbb{R}^{N\times M}$ as

$$\frac{\mathrm{d}^2\widetilde{\mathcal{I}}}{\mathrm{d}x_i\mathrm{d}x_j} = \frac{\mathrm{d}^2\mathcal{I}}{\mathrm{d}x_i\mathrm{d}x_j} + \mu_{i,m}\frac{\mathrm{d}\mathcal{R}_m}{\mathrm{d}x_j} + \nu_{i,n}\frac{\mathrm{d}\mathcal{R}_n^{\psi}}{\mathrm{d}x_j} \qquad m,n \in [1,M] \tag{11}$$

After taking derivatives of the primal and adjoint systems of equations with respect to the j^{th} design variable, this twice-augmented functional becomes

$$\begin{aligned}\frac{\mathrm{d}^2\widetilde{\mathcal{I}}}{\mathrm{d}x_i\mathrm{d}x_j} = {} & \frac{\partial^2 F}{\partial x_i\partial x_j} + \psi_m\frac{\partial^2\mathcal{R}_m}{\partial x_i\partial x_j} + \mu_{i,m}\frac{\partial\mathcal{R}_m}{\partial x_j} + \nu_{i,n}\frac{\partial^2 F}{\partial u_n\partial x_j} \\ & + \nu_{i,n}\psi_m\frac{\partial^2\mathcal{R}_m}{\partial u_n\partial x_j} + \left(\frac{\partial^2 F}{\partial x_i\partial u_k} + \psi_m\frac{\partial^2\mathcal{R}_m}{\partial x_i\partial u_k} + \mu_{i,m}\frac{\partial\mathcal{R}_m}{\partial u_k} + \nu_{i,n}\frac{\partial^2 F}{\partial u_n\partial u_k} + \nu_{i,n}\psi_m\frac{\partial^2\mathcal{R}_m}{\partial u_n\partial u_k}\right)\frac{\mathrm{d}u_k}{\mathrm{d}x_j} \\ & + \frac{\mathrm{d}\psi_m}{\mathrm{d}x_j}\left(\frac{\partial\mathcal{R}_m}{\partial x_i} + \nu_{i,n}\frac{\partial\mathcal{R}_m}{\partial u_n}\right) \qquad k,m,n \in [1,M]\end{aligned} \tag{12}$$

Since we do not want to compute $\mathrm{d}\mathbf{u}/\mathrm{d}\mathbf{x}$ nor $\mathrm{d}\boldsymbol{\psi}/\mathrm{d}\mathbf{x} \in \mathbb{R}^{M\times N}$, we can select $\boldsymbol{\mu}$ and $\boldsymbol{\nu}$ such that

$$\nu_{i,n}\frac{\partial\mathcal{R}_m}{\partial u_n} = -\frac{\partial\mathcal{R}_m}{\partial x_i} \qquad m,n \in [1,M] \quad i \in [1,N] \tag{13}$$

and

$$\mu_{i,m}\frac{\partial\mathcal{R}_m}{\partial u_k} = -\frac{\partial^2 F}{\partial x_i\partial u_k} - \psi_m\frac{\partial^2\mathcal{R}_m}{\partial x_i\partial u_k} - \nu_{i,n}\frac{\partial^2 F}{\partial u_n\partial u_k} - \nu_{i,n}\psi_m\frac{\partial^2\mathcal{R}_m}{\partial u_n\partial u_k} \qquad k,m,n \in [1,M] \quad i \in [1,N] \tag{14}$$

Eq.(13) is first solved for $\boldsymbol{\nu}$ which is equivalent to N system solutions; notice the left-hand side is once again the Jacobian of the primal equations, specifically the transposed matrix for linear systems. Subsequently, eq.(14) is solved for the N rows of $\boldsymbol{\mu}$, accumulating another N system solutions. The total number of system solutions is thus $2N + 1$ (recall we need one for the original adjoint vector) in the adjoint-adjoint formulation. The Hessian can be computed from

$$h_{i,j} = \frac{\mathrm{d}^2\widetilde{\mathcal{I}}}{\mathrm{d}x_i\mathrm{d}x_j} = \frac{\partial^2 F}{\partial x_i\partial x_j} + \psi_m\frac{\partial^2\mathcal{R}_m}{\partial x_i\partial x_j} + \mu_{i,m}\frac{\partial\mathcal{R}_m}{\partial x_j} + \nu_{i,n}\frac{\partial^2 F}{\partial u_n\partial x_j} + \nu_{i,n}\psi_m\frac{\partial^2\mathcal{R}_m}{\partial u_n\partial x_j} \qquad m,n \in [1,M] \tag{15}$$

Operation count: We will not include the operation count involved in solving eq.(13) since it is simply N system solutions and is counted separately. The operation count in constructing the right-hand sides of eq.(14) is dominated by the term containing the second-order derivative of the residual with respect to the state variables. This term is looped over three indices, accumulating $2M^3$ (both adjoint variable multiplications) operations for each $\boldsymbol{\nu}_i$ for a total of $\mathcal{O}(NM^3)$. The computation of each entry in the Hessian is dominated by the mixed second-order partial derivative of the residual, costing $2M^2$ operations. Only $N(N+1)/2$ components of the Hessian need to be computed because of symmetry; the operation count of forming the Hessian is still $\mathcal{O}(N^2M^2)$.

D. Adjoint-Direct method

Instead of introducing two new sets of adjoint vectors, consider differentiating the simply-augmented objective function with respect to x_j. Each i, j entry is given by

$$\frac{\mathrm{d}^2 F}{\mathrm{d}x_i \mathrm{d}x_j} = \frac{\partial^2 F}{\partial x_i \partial x_j} + \frac{\partial^2 F}{\partial x_i \partial u_k}\frac{\mathrm{d}u_k}{\mathrm{d}x_j} + \psi_m \frac{\partial^2 \mathcal{R}_m}{\partial x_i \partial x_j} + \psi_m \frac{\partial^2 \mathcal{R}_m}{\partial x_i \partial u_k}\frac{\mathrm{d}u_k}{\mathrm{d}x_j} + \frac{\mathrm{d}\psi_m}{\mathrm{d}x_j}\frac{\partial \mathcal{R}_m}{\partial x_i} \qquad k, m \in [1, M] \tag{16}$$

The trick now comes in computing $\mathrm{d}\psi_m/\mathrm{d}x_j$. We can use the total derivative of the adjoint equation with respect to x_j, directly, to obtain the desired vectors.

$$\frac{\mathrm{d}\psi_m}{\mathrm{d}x_j}\frac{\partial \mathcal{R}_m}{\partial u_n} = -\frac{\partial^2 F}{\partial u_n \partial x_j} - \frac{\partial^2 F}{\partial u_n \partial u_k}\frac{\mathrm{d}u_k}{\mathrm{d}x_j} - \psi_m \frac{\partial^2 \mathcal{R}_m}{\partial u_n \partial x_j} - \psi_m \frac{\partial^2 \mathcal{R}_m}{\partial u_n \partial u_k}\frac{\mathrm{d}u_k}{\mathrm{d}x_j} \qquad k, m, n \in [1, M] \quad i \in [1, N] \tag{17}$$

There are N right-hand side vectors which translates to N system solutions. Since this method relies on initally computing $\mathrm{d}\mathbf{u}/\mathrm{d}\mathbf{x}$ (direct method), N system solutions have already been performed along with one additional solution for the adjoint vector, $\boldsymbol{\psi}$. The total number of system solutions is then $2N+1$ similar to the adjoint-adjoint method. The Hessian matrix is then evaluated from eq.(16).

Operation count: When forming the right-hand side of eq.(17), the operation count is dominated by the second-order partial derivative of the resiudal with respect to the state variables. This term is index over k, m, n and accumulates $2M^3$ operations per solve, $\mathcal{O}(NM^3)$ total. Subsequently, the Hessian computation is again dominated by the mixed partial derivative of the residual. This term contains two multiplications, accumulating $2M^2$ operations for a total of $\mathcal{O}(N^2M^2)$ for the entire Hessian.

We have now seen two methods for computing the Hessian matrix with $2N + 1$ system solutions. An obvious question arises: can we do better?

E. Direct-Adjoint method

The direct-adjoint method assumes the M sensitivity vectors, $\mathrm{d}\mathbf{u}/\mathrm{d}\mathbf{x}$, have been computed with the direct method mentioned above. Papadimitriou and Giannakoglou [14] define a new augmented cost functional as

$$\frac{\mathrm{d}^2 \mathcal{L}}{\mathrm{d}x_i \mathrm{d}x_j} = \frac{\mathrm{d}^2 F}{\mathrm{d}x_i \mathrm{d}x_j} + \hat{\psi}_n \frac{\mathrm{d}^2 \mathcal{R}_n}{\mathrm{d}x_i \mathrm{d}x_j} \tag{18}$$

Substituting eqs.(4) and (8) yields

$$\begin{aligned}\frac{\mathrm{d}^2 \mathcal{L}}{\mathrm{d}x_i \mathrm{d}x_j} = &\frac{\partial^2 F}{\partial x_i \partial x_j} + \hat{\psi}_n \frac{\partial^2 \mathcal{R}_n}{\partial x_i \partial x_j} + \frac{\partial^2 F}{\partial u_k \partial u_m}\frac{\mathrm{d}u_k}{\mathrm{d}x_i}\frac{\mathrm{d}u_m}{\mathrm{d}x_j} \\ &+ \hat{\psi}_n \frac{\partial^2 \mathcal{R}_n}{\partial u_k \partial u_m}\frac{\mathrm{d}u_k}{\mathrm{d}x_i}\frac{\mathrm{d}u_m}{\mathrm{d}x_j} + \frac{\partial^2 F}{\partial x_i \partial u_k}\frac{\mathrm{d}u_k}{\mathrm{d}x_j} + \hat{\psi}_n \frac{\partial^2 \mathcal{R}_n}{\partial x_i \partial u_k}\frac{\mathrm{d}u_k}{\mathrm{d}x_j} + \frac{\partial^2 F}{\partial u_k \partial x_j}\frac{\mathrm{d}u_k}{\mathrm{d}x_i} + \hat{\psi}_n \frac{\partial^2 \mathcal{R}_n}{\partial u_k \partial x_j}\frac{\mathrm{d}u_k}{\mathrm{d}x_i} \\ &+ \left(\frac{\partial F}{\partial u_k} + \hat{\psi}_n \frac{\partial \mathcal{R}}{\partial u_k}\right)\frac{\mathrm{d}^2 u_k}{\mathrm{d}x_i \mathrm{d}x_j} \qquad [k, m, n] \in [1, M]\end{aligned} \tag{19}$$

This time we would like to avoid computing $\mathrm{d}^2 u_k/\mathrm{d}x_i \mathrm{d}x_j$. We can do so by forcing

$$\hat{\psi}_n \frac{\partial \mathcal{R}_n}{\partial u_k} = -\frac{\partial F}{\partial u_k} \qquad [k, n] \in [1, M] \tag{20}$$

which is exactly the same as eq.(9), requiring only one system solution. Each i, j entry in the Hessian is then given by

$$\begin{aligned}\frac{\mathrm{d}^2 \mathcal{L}}{\mathrm{d}x_i \mathrm{d}x_j} = &\frac{\partial^2 F}{\partial x_i \partial x_j} + \hat{\psi}_n \frac{\partial^2 \mathcal{R}_n}{\partial x_i \partial x_j} + \frac{\partial^2 F}{\partial u_k \partial u_m}\frac{\mathrm{d}u_k}{\mathrm{d}x_i}\frac{\mathrm{d}u_m}{\mathrm{d}x_j} + \hat{\psi}_n \frac{\partial^2 \mathcal{R}_n}{\partial u_k \partial u_m}\frac{\mathrm{d}u_k}{\mathrm{d}x_i}\frac{\mathrm{d}u_m}{\mathrm{d}x_j} + \frac{\partial^2 F}{\partial x_i \partial u_k}\frac{\mathrm{d}u_k}{\mathrm{d}x_j} \\ &+ \hat{\psi}_n \frac{\partial^2 \mathcal{R}_n}{\partial x_i \partial u_k}\frac{\mathrm{d}u_k}{\mathrm{d}x_j} + \frac{\partial^2 F}{\partial u_k \partial x_j}\frac{\mathrm{d}u_k}{\mathrm{d}x_i} + \hat{\psi}_n \frac{\partial^2 \mathcal{R}_n}{\partial u_k \partial x_j}\frac{\mathrm{d}u_k}{\mathrm{d}x_i} \qquad [k, m, n] \in [1, M]\end{aligned} \tag{21}$$

Since N system solutions were required to obtain $\mathrm{d}\mathbf{u}/\mathrm{d}\mathbf{x}$, the total number of system solutions required in the direct-adjoint approach is $N+1$. The operation count, however, is compromised by the three loops over k, m, n which amount to $\mathcal{O}(M^3)$ for each entry, giving a combined $\mathcal{O}(N^2M^3)$ for the entire Hessian. In general, these systems arise from discretizing a linear or nonlinear partial differential equation which lead to lightly coupled equations (sparse matrix in the case of a linear system). Thus, in real applications the operation count of the matrix multiplications may not be too worrisome; the computational cost is often dominated by the system solutions. In addition, Papadimitriou and Giannakoglou report omitting some of these expensive computations without drastically compromising the accuracy of the second derivatives [14]. The impact of this claim, however, should be problem-dependent.

F. Hyper-dual numbers

Some researchers advocate the use of the complex-step method to compute first derivatives. The advantage of this method is that first derivatives can be computed with second-order accuracy, with one additional function call and without any subtractive cancellation errors [10]. The disadvantage, however, is that the resulting solver must be capable of handling complex numbers, hence increasing the operation count of a system solution. The first and second derivatives of a complex function can be approximated by

$$\frac{\mathrm{d}f}{\mathrm{d}x} = \frac{\mathrm{Imag}[f(x+ih)]}{h} + \mathcal{O}(h^2) \tag{22}$$

$$\frac{\mathrm{d}^2f}{\mathrm{d}x^2} = \frac{2(f(x) - \mathrm{Re}[f(x+ih)])}{h^2} + \mathcal{O}(h^2) \tag{23}$$

Notice that the second derivative is susceptible to subtractive cancellation errors.

Jeffrey Fike introduced hyper-dual numbers, originally "fikequats," which can, similar to the complex-step method, be used to compute second derivatives. A hyper-dual number contains four components,

$$x = x_0 + \epsilon_i x_i + \epsilon_j x_j + \epsilon_i\epsilon_j x_{ij} \tag{24}$$

where $\epsilon_i^2 = \epsilon_j^2 = (\epsilon_i\epsilon_j)^2 = 0$ but $\epsilon_i,\ \epsilon_j,\ \epsilon_{ij} \neq 0$. The Taylor series expansion for a hyper-dual perturbation, $h = h_i\epsilon_i + h_j\epsilon_j$ truncates after the second derivative since $\epsilon_i^2 = \epsilon_j^2 = 0$; therefore,

$$f(x+h) = f(x) + h_i\frac{\mathrm{d}f}{\mathrm{d}x}\epsilon_i + h_j\frac{\mathrm{d}f}{\mathrm{d}x}\epsilon_j + h_ih_j\frac{\mathrm{d}^2f}{\mathrm{d}x^2}\epsilon_i\epsilon_j \tag{25}$$

Hyper-dual numbers can also be used to compute first derivatives. Extracting the ϵ_i and ϵ_j of the above expansion gives

$$\frac{\mathrm{d}f}{\mathrm{d}x} = \frac{\epsilon_i\text{part of }\ [f(x+h_i\epsilon_i+h_j\epsilon_j)]}{h_i} \tag{26}$$

or

$$\frac{\mathrm{d}f}{\mathrm{d}x} = \frac{\epsilon_j\text{part of }\ [f(x+h_i\epsilon_i+h_j\epsilon_j)]}{h_j} \tag{27}$$

Extracting the $\epsilon_i\epsilon_j$ component of $f(x+h)$ yields the exact second derivative,

$$\frac{\mathrm{d}^2f}{\mathrm{d}x^2} = \frac{\epsilon_i\epsilon_j\text{part of }\ [f(x+h_i\epsilon_i+h_j\epsilon_j)]}{h_ih_j} \tag{28}$$

For a function of several variables, the first and second partial derivatives are given by

$$\frac{\partial f}{\partial x_i} = \frac{\epsilon_i\text{part of }\ [f(x+h_i\epsilon_i+h_j\epsilon_j)]}{h_i} \tag{29}$$

or

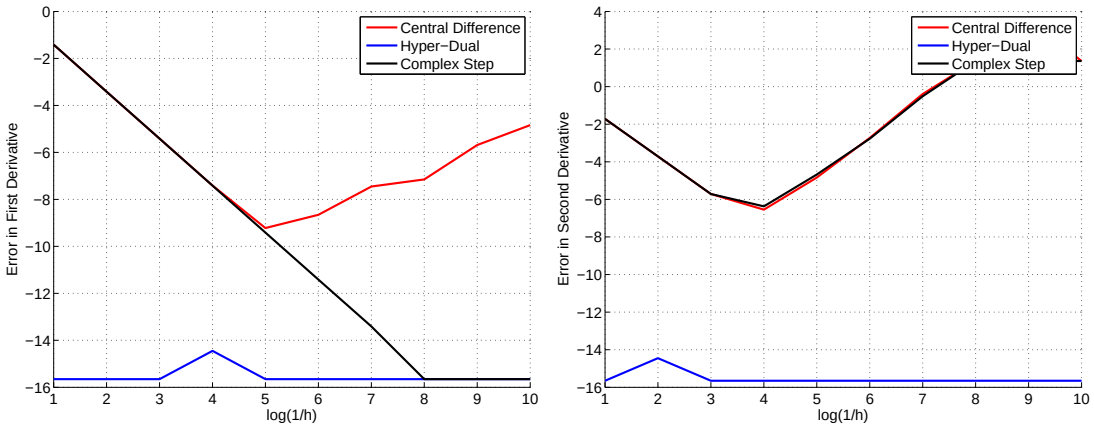
$$\frac{\partial f}{\partial x_j} = \frac{\epsilon_j\text{part of }\ [f(x+h_i\epsilon_i+h_j\epsilon_j)]}{h_j} \tag{30}$$

and

$$\frac{\partial^2 f}{\partial x_i \partial x_j} = \frac{\epsilon_i \epsilon_j \text{part of } [f(x + h_i\epsilon_i + h_j\epsilon_j)]}{h_i h_j} \tag{31}$$

Note that for N design variables, only $N/2$ additional function evaluations are required since eqs.(29) and (30) can be used to compute first derivatives with respect to different design variables with the same function evaluation. The second derivative, on the other hand, requires N^2 function evaluations; however, this can be reduced to $N(N+1)/2$ if the symmetry of $\mathbf{H}$ is exploited.

In contrast to the complex-step method, Fike's hyper-dual number method allows first and second derivatives to be computed *exactly*. This phenomenon is shown in Fig. 1 where the hyper-dual method outperforms both the complex step method and finite differences for the computation of second derivatives.



(a) Error in first derivative

(b) Error in second derivative

Fig. 1. Comparison of errors in computing first and second derivatives of e^x at $x = \pi$

Operation count: Although the computational savings and exactness of the hyper-dual number method are impressive, one should recall that adding (subtracting) two hyper-dual numbers requires 4 floating-point operations and multiplying (dividing) requires nine additional operations [5]; the details of these operations can be found in [4]. Although the gradient and Hessian can be computed exactly with an acceptable number of function calls, the cost of each function call increases dramatically. An example of the operation count needed to solve a tridiagonal system of equations of hyper-dual numbers is given in the Appendix.

G. Summary

Let's now summarize the different methods for computing exact Hessians. In Table I, the number of system solutions, intermediate operations and operations required for constructing the Hessian are given for each method. The number of intermediate operations refers to the operation count in forming the right-hand side vectors for the intermediate adjoint or direct solutions before computing the Hessian. The cost of forming the gradient is omitted because the focus, here, is on computing the Hessian. In all cases, however, an adjoint-based method can be used independently of the Hessian computation to form the gradient.

TABLE I
SUMMARY OF THE FOUR METHODS FOR COMPUTING HESSIANS

Method	System solutions	Intermediate operations	Hessian operations
Adjoint-adjoint	$2N$	$\mathcal{O}(NM^3)$	$\mathcal{O}(N^2M^2)$
Adjoint-direct	$2N$	$\mathcal{O}(NM^3)$	$\mathcal{O}(N^2M^2)$
Direct-adjoint	N	-	$\mathcal{O}(N^2M^3)$
Hyper-dual	$N(N+1)/2$	-	hyper-dual operations

III. Numerical Experiments

In this section, the Hessian obtained from all four methods will be compared and used to solve optimization problems. In all cases, the differential equation studied is a mix between Poisson's equation and the Helmholtz equation (with spatially varying wavenumber); the boundary value problem for the case of homogeneous Dirichlet conditions is

$$\frac{\mathrm{d}^2 u}{\mathrm{d}z^2} + \gamma(z)u = -f(z) \qquad u \in [0,1], \qquad u(0) = u(L) = 0 \tag{32}$$

where $\gamma(z)$ is referred to as a source (or sink) distribution. A second-order finite-difference discretization is used to solve for the solution u at each of the N grid points. Each of the N residual equations thus appears as

$$\mathcal{R}_n = f_n - \left[\frac{u_{n-1}}{\Delta z^2} + u_n\left(\gamma_n - \frac{2}{\Delta z^2}\right) + \frac{u_{n+1}}{\Delta z^2}\right] = 0 \qquad n \in [1, N] \tag{33}$$

with the exception of the first and last rows. Here, the forcing function is set to $f(z) = z^p$ for some integer y. The goal of the optimization is to find a discrete representation of $\boldsymbol{\gamma} \in \mathbb{R}^N$ to minimize the error between the computed u and some target distribution,

$$\min_{\boldsymbol{\gamma}} \quad \frac{1}{2}\sum_{i=1}^{N}(u_i - u_{i,t})^2 \tag{34}$$

where u is obtained from the solution of N linear equations given by eq.(33), or

$$\mathcal{R} = \mathbf{b} - \mathbf{A}\mathbf{u} = \mathbf{0} \tag{35}$$

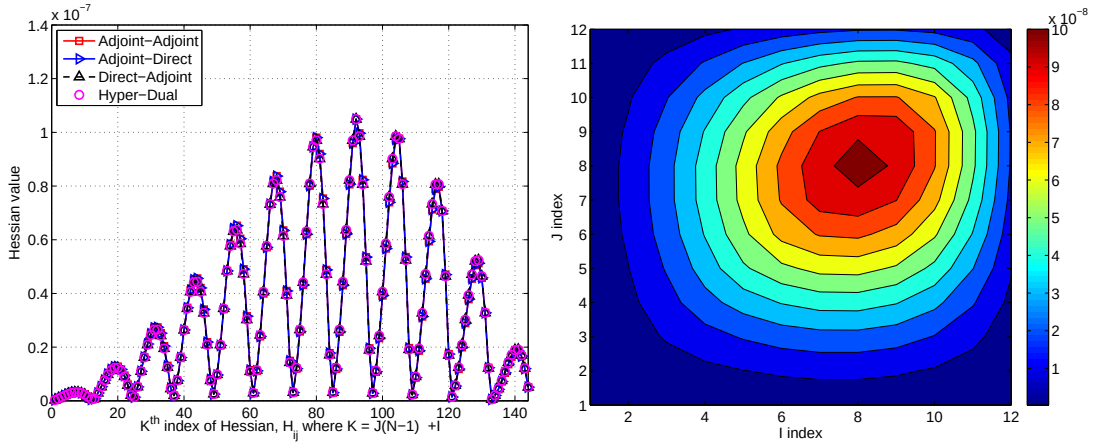
The target distribution, u_t, is chosen to be

$$u_t(z) = -\frac{z^q}{(q+1)(q+2)} + \frac{L^q z}{(q+1)(q+2)} \tag{36}$$

for some integer q. This form for the target solution was chosen because it allows for a few interesting cases to be studied, notably when $p = q$ and $p \neq q$. Also note the matrix resulting from discretizing eq.(32) is tridiagonal. For the case of real numbers, the structure of $\mathbf{A}$ does not matter because MATLAB's backslash operator is used to solve the system of equations. When solving the system for the case of a hyper-dual perturbation, MATLAB's internal solver cannot be used. Thus, a simple tridiagonal solver [1] was specifically written to handle hyper-dual numbers which can be shown to require seven times the operations of a real number solution (please see the Appendix).

A. *Case I:* $p = q = 8$, $N = 12$

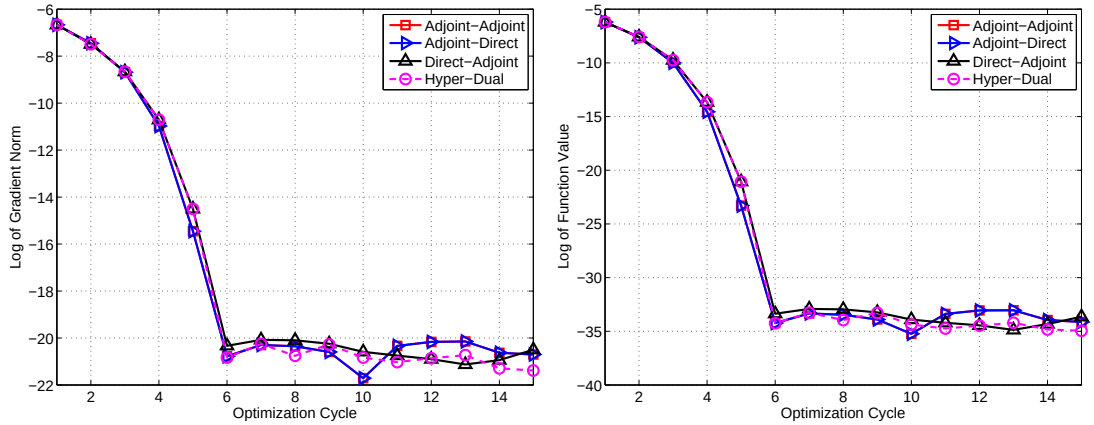
When $p = q$, the optimization problem can be stated in words as follows: find the source distribution, $\gamma(z)$, so that the numerical solution becomes the analytic solution without any source distribution, $\gamma(z) = 0$. If we used an excessive amount of grid points (or a small enough p), the numerical solution would be exactly the analytic solution and we should get $\gamma(z) = 0$. However, let's use a small number of design variables, $N = 12$, so that there is a noticeable amount of error between the numerical and target (analytic) solutions. The design variables are set to $\gamma(z) = 1$ as the initial guess. Two depictions of the Hessian at the initial design point are shown in Fig. 2. In Fig. 2(a), each Hessian entry is plotted with an outer loop over the row indices and an inner loop over the column indices. The values obtained from all four methods are in good agreement with each other. Fig. 2(b) depicts the Hessian obtained from the direct-adjoint method with the h_{11} entry in the bottom left corner; observe the symmetry about a diagonal drawn from the bottom left corner to the upper right corner.



(a) Hessian line plot (b) Hessian contour plot

Fig. 2. Hessian plot at initial design point for the case $p = q = 8$, $N = 12$

The optimization problem was solved with each of the four methods used to compute the exact Hessians. Fig. 3 shows the convergence behaviour of all four methods. Notice that each method exhibits the expected quadratic convergence characteristic of Newton's method. Also, it was observed that the adjoint-adjoint and adjoint-direct methods produced identical Hessians at each iteration; however, these slightly differed from the Hessian obtained with the direct-adjoint method. This difference can be explained as follows. Recall the discretization of the differential equation introduces error in the computed solution. Since the adjoint method, here, is solved discretely using the same representation of the linear differential operator, the adjoint solution itself is also approximate. Since both the adjoint-adjoint and adjoint-direct methods require $2N$ function evaluations during each iteration, they introduce more numerical error in the computed Hessian, whereas the direct-adjoint method performs only N evaluations and can be expected to introduce less error. Equivalently, the Hessian from the direct-adjoint method differed slightly from that obtained with the hyper-dual method. Again, this is explained by the frequent function evaluations performed by the hyper-dual method which all introduce numerical error based on the discretization of the differential equation.

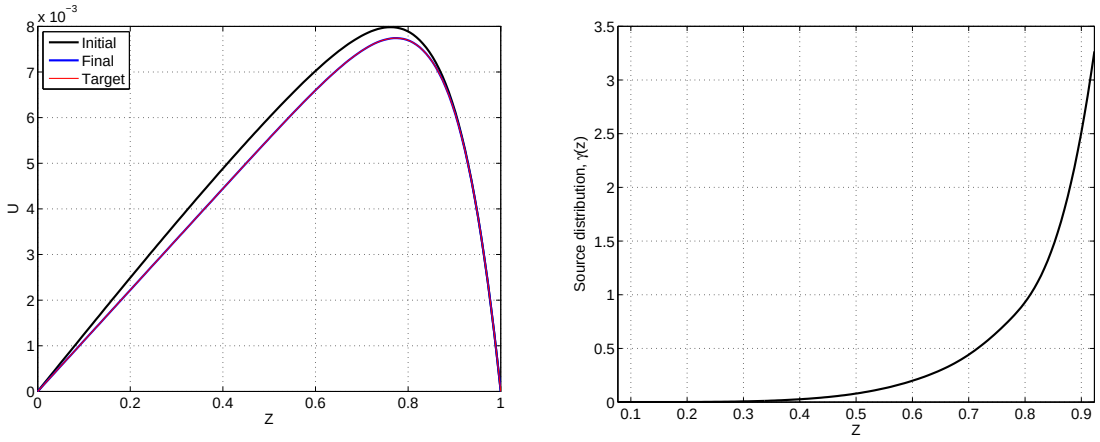


(a) Convergence of gradient for all four methods (b) Convergence of objective function

Fig. 3. Convergence of the optimization algorithm with the different methods to compute the Hessian for the case $p = q = 8$, $N = 12$

All methods converge on the target solution after approximately six optimization iterations. Some methods were more costly than others. The adjoint-based methods were all extremely quick whereas the hyper-dual method suffered from the additional operations needed for addition and multiplication required by the tridiagonal solver. Although the speed of the hyper-dual solver is perhaps limited by the specific implementation, it nonetheless dramatically

increases the time needed for one function evaluation, making it impractical. Unfortunately, an optimized hyper-dual linear solver has yet to be developed so speed comparisons with MATLAB's backslash operator are meaningless. However, a comment will be made about speed: the full optimization with any of the adjoint-based methods converged on the order of seconds whereas the hyper-dual optimization converged in about 6 minutes for this problem. The initial, final and target solutions are shown in Fig. 4(a); the optimal source distribution is given in Fig. 4(b).



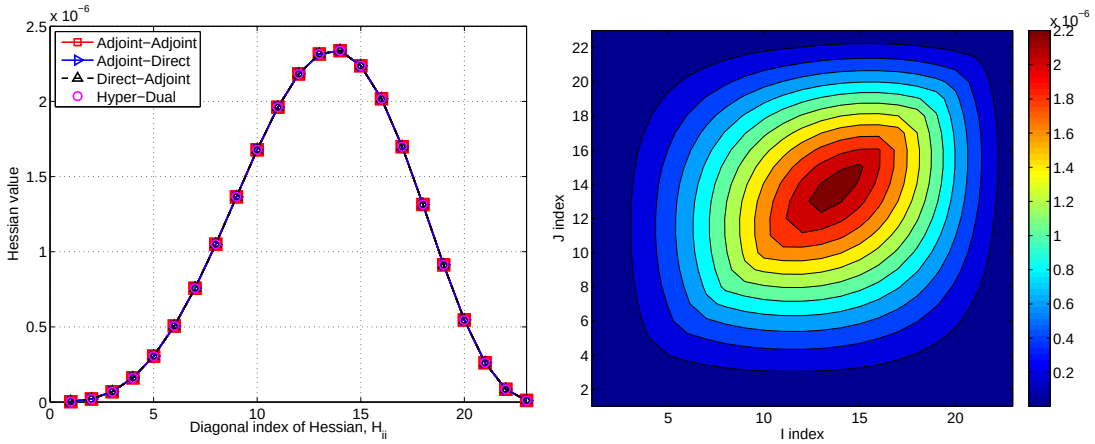
(a) Initial, final and target solutions

(b) Optimal $\gamma(z)$

Fig. 4. Initial, final and target solutions along with optimal source distribution for the case $p = q = 8$, $N = 12$

B. Case II: $p = 3, q = 8,\ N = 23$

Let's now apply the different methods with a larger number of design variables ($N = 23$ was as high as I could patiently go while waiting for the hyper-dual solver to complete) when $p \neq q$. We cannot physically describe the optimization problem as in the previous subsection; we can only hope the algorithm with each of the Hessians finds the optimal source distribution for the arbitrary target solution. Instead of plotting all Hessian entries, Fig. 5(a) shows the entries along the diagonal of the Hessian at the initial design point, $\gamma(z) = 1$. Once again, all derivatives obtained with the four methods are in excellent agreement. Similar to the previous case, observe the symmetry in the Hessian in Fig. 5(b).

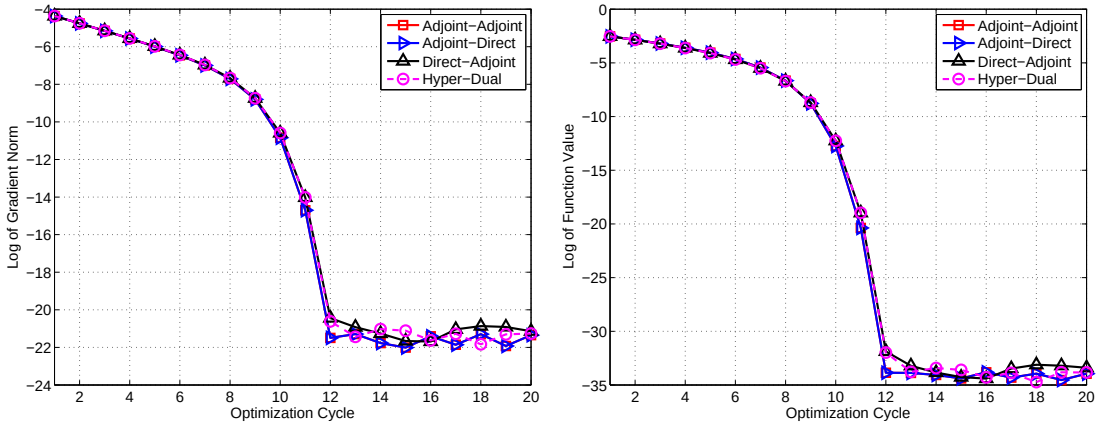


(a) Hessian line plot

(b) Hessian contour plot

Fig. 5. Hessian plot at initial design point for the case $p = 3, q = 8,\ N = 23$

The quadratic behaviour of Newton's method is well-observed in the convergence of the gradient and function values as shown in Fig. 6. Once again, the adjoint-adjoint and adjoint-direct methods produced identical Hessians at each iteration which differed slightly from the direct-adjoint method. However, this difference was observed to be smaller than the previous case which is explained by the increased number of grid points, giving a more accurate primal and adjoint solution.

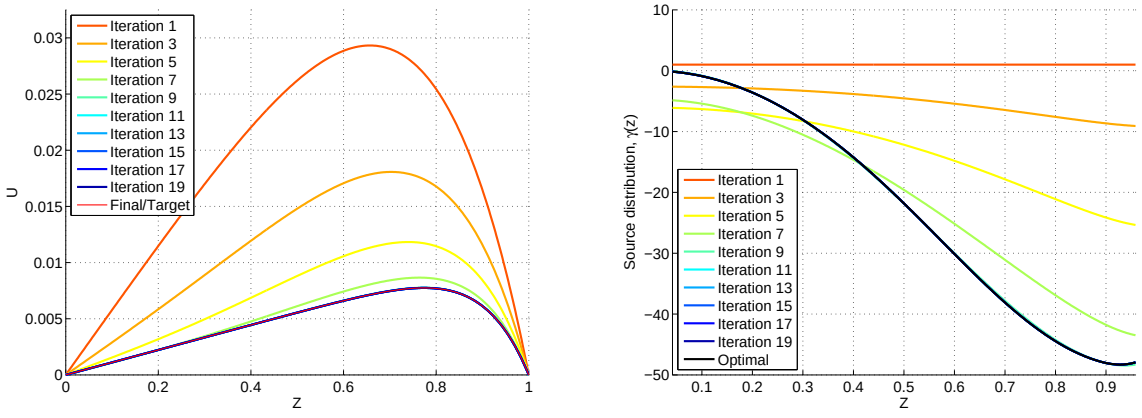


(a) Convergence of gradient for all four methods

(b) Convergence of objective function

Fig. 6. Convergence of the optimization algorithm with the different methods to compute the Hessian for the case $p = 3$, $q = 8$, $N = 23$

The histories of the solution and source distribution are shown in Fig. 7 which demonstrate how quickly the algorithm converges with Newton's method. The steps during each iteration, however, are quite aggressive and sometimes overshoot the optimal set of design variables. Also note that doubling the number of design variables led to more than twice the time for the hyper-dual implementation to complete; with 23 design variables, the hyper-dual optimization algorithm finished in 14 minutes (again, these are just to get a feel for the numbers – my hyper-dual code is not optimized).



(a) Initial, final and target solutions

(b) Optimal $\gamma(z)$

Fig. 7. Initial, final and target solutions along with optimal source distribution for the case $p = 3$, $q = 8$, $N = 23$

C. Case III: Comparison with Quasi-Newton Method

The previous cases demonstrate the accelerated convergence of an optimization algorithm when the exact Hessian is computed. Although this is perhaps not a fair comparison with the exact Hessian methods, let's see how well a

quasi-Newton method compares with the direct-adjoint method. This adjoint-based method was selected because it performed very well and required the fewest number of function evaluations ($N + 1$ total per optimization cycle). The method is compared with MATLAB's `fminunc` which uses the BFGS algorithm for medium scale problems. Instead of plotting the convergence history for each cycle, Fig. 8 depicts the convergence over the equivalent number of function evaluations, which is perhaps a fairer comparison between the two methods. Notice that the direct-adjoint method still outperforms the quasi-Newton method over equivalent function evaluations. The BFGS method used 16 major iterations to convergence; however, it performed 576 function evaluations which means the line search method was the primary source of function calls. Papadimitriou and Giannakoglou report similar results in their airfoil reconstruction studies: the exact Newton methods converged quicker than quasi-Newton methods over equivalent Navier-Stokes flow solutions [14].

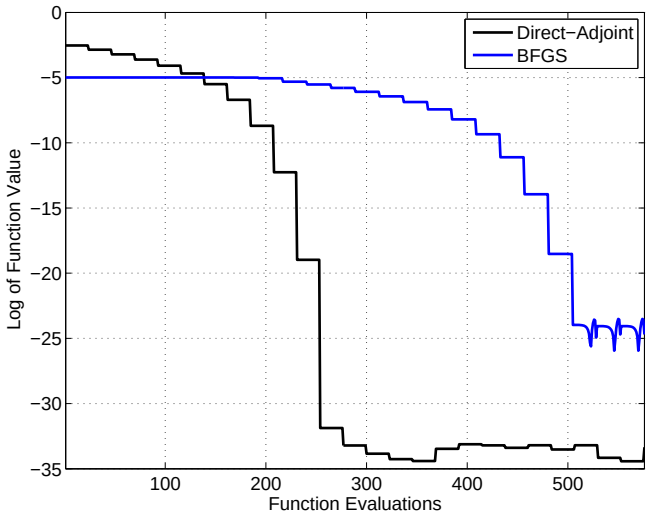


Fig. 8. Convergence comparison between direct-adjoint and BFGS method for the case $p = 3$, $q = 8$, $N = 23$

IV. Conclusion

This paper has examined the various methods used to exactly compute the Hessian of an expensive objective function. The exactness of each method (adjoint-adjoint, adjoint-direct, direct-adjoint, hyper-dual) was shown through examples of an optimization problem in which the objective function was evaluated from the solution to a linear system of equations. The methods, however, should be understood as exact within the discretization of the differential equation which influences the linear system. In other words, methods that used $2N$ function evaluations (adjoint-adjoint, adjoint-direct) produced identical Hessians but slightly differed from the Hessians obtained from methods that used N evaluations (direct-adjoint) or $N(N+1)/2$ evalutions (hyper-dual). The best method is clearly the direct-adjoint method as this requires the fewest number of function evaluations. The hyper-dual method is not very effective and suffers from the increased computational cost of a single function evaluation; in addition, it requires $N(N + 1)/2$ function evaluations to obtain the exact Hessian. Finally, the direct-adjoint method was compared with a quasi-Newton method (BFGS) over equivalent function evaluations. The Hessian obtained with the direct-adjoint method provided faster convergence, despite incurring N additional function evaluations for the Hessian. Although this behaviour may be desirable, the BFGS method may be more suitable to certain problems with a much larger number of design variables. Storing the Hessian as well as the intermediate matrices needed in the adjoint-based methods can become very memory-intensive. In such cases, the limited-memory BFGS method may be a more suitable option.

APPENDIX
FLOATING-POINT OPERATIONS FOR HYPER-DUAL TRIDIAGONAL SOLVER

The Thomas algorithm for solving a tridiagonal system of equations:

$$a_i x_{i-1} + b_i x_i + c_i x_{i+1} = d_i \qquad i = 1, \ldots, N \tag{37}$$

is a special case of Gaussian elimination [1]. The algorithm is given below

```
cp(1) = c(1)/b(1);
dp(1) = d(1)/b(1);

% Forwards sweep
for i=2:N
  den   = b(i)  -cp(i-1)*a(i);
  cp(i) = c(i)/den;
  dp(i) = (d(i)  -dp(i-1)*a(i))/den;
end

% Backwards sweep
x(N) = dp(N);
for i=N-1:-1:1
  x(i)  = dp(i)  -cp(i)*x(i+1);
end
```

For the forward sweep, we have 4 multiplications (divisions) and two additions (subtractions) for each row. The backward sweep creates one additional addition and multiplication per row. The total number of additions is then $3(N-1)$; the total number of multiplications is $5(N-1)+2$ which becomes $8N-6$ floating-point operations.

In the case of hyper-dual numbers, one hyper-dual addition translates to 4 floating-point operations and one multiplication translates to 9 floating-point operations. Thus the total operation count for a hyper-dual tridiagonal solver using the Thomas algorithm is $4 \cdot 3(N-1) + 9 \cdot (5(N-1)+2) = 57N - 39$ which is approximately seven times the operation count of a real-number solve.

REFERENCES

[1] Bewley, T. R., *Numerical Methods in Science and Engineering*. Lecture Notes, University of California, San Diego, 2006.

[2] Fike, J. A. and Alonso, J. A., *The Development of Hyper-Dual Numbers for Exact Second-Derivative Calculations*. 49th AIAA Aerospace Sciences Meeting, 4 January 2011.

[3] Fike, J. A. and Alonso, J. A., *Automatic Differentiation Through the Use of Hyper-Dual Number for Second Derivatives*. Lecture Notes in Computational Science and Engineering (87): Recent Advances in Algorithmic Differentiation, 2012.

[4] Fike, J. A., *Numerically Exact Derivative Calculations Using Fake Numbers*. Unpublished report, Stanford University, 9 April 2008.

[5] Fike, J. A., Jongsma, S., Alonso, J. A. and van der Weide, E. *Optimization with Gradient and Hessian Information Calculated Using Hyper-Dual Numbers*. 29th AIAA Applied Aerodynamics Conference, 27-30 June 2011.

[6] Fike, J. A., *Numerically Exact Derivative Calculations Using Hyper-Dual Numbers*. 3rd Annural Student Joint Workshop in Simulation-Based Engineering and Design, 18 June 2009.

[7] Giles, M. B. and Pierce, N. A., *Adjoint equations in CFD: duality, boundary conditions and solution behaviour*. AIAA Paper: 97-1850, 1997.

[8] Johnson, S. G., *Notes on FFT-based differentiation*. MIT Lecture Notes, 4 May 2011.

[9] Jongen, H., Meer, K. and Triesch, E., *Optimization Theory*. Kluwer Academic Publishers Group, 2004.

[10] Lai, K., Crassidis, J. L. and Cheng, Y., *New Complex-Step Derivative Approximations with Application to Second-Order Kalman Filtering*. AIAA Guidance, Navigation and Control Conference, San Francisco, California, August 2005, AIAA-2005-5944.

[11] Martinelli, M., Dervieux, A. and Hascoet, L., *Strategies for Computing Second-Order Derivatives in CFD Design Problems*. West East High Speed Flow Field Conference (Moscow), 20 November 2007.

[12] Nadarajah, S., and Jameson, A. *Studies of the Continuous and Discrete Adjoint Approaches to Viscous Automatic Aerodynamic Shape Optimization*. 15^{th} AIAA Compuational Fluid Dynamics Conference, June 2001.

[13] O'Leary, D. P. and Yeremin, A., *The Linear Algebra of Block Quasi-Newton Algorithms*. Linear Algebra and its Applications, Elsevier Science, 212/213:153-168, 1994.

[14] Papadimitriou, D. I. and Giannakoglou, K. C., *Direct, adjoint and mixed approaches for the computation of Hessian in airfoil design problems*. International Journal for Numerical Methods in Fluids, 56: 1929-1943, 2008.

[15] Scolnik, H. D. and Gambini, M. J., *A New Method to Compute Second Derivatives*. Unpublished report, University of Buenos Aires, 24 July 2001.