29th AIAA Applied Aerodynamics Conference
27 - 30 June 2011, Honolulu, Hawaii

AIAA 2011-3807

Optimization with Gradient and Hessian Information Calculated Using Hyper-Dual Numbers

Jeffrey A. Fike * Sietse Jongsma † Juan J. Alonso ‡ Edwin van der Weide §

Hyper-dual numbers can be used to compute exact first and second derivatives in order to form gradients and Hessians for optimization methods. There is however an increased computational cost associated with the mathematics of hyper-dual numbers. A strategy for increasing the computational efficiency of this technique is presented for situations involving iterative procedures. This strategy involves first converging the iterative procedure using real numbers, then computing the derivatives starting from this converged solution. The benefit this strategy offers is demonstrated in the optimization of a supersonic business jet using low-fidelity, conceptual-design-level tools. In this case, the computational cost of using hyper-dual numbers is reduced to below that of using finite difference formulas. This strategy is then applied to computational fluid dynamics codes. The effectiveness of the strategy depends on the use of the exact Jacobian of the residual. Optimization of an inviscid, transonic airfoil is presented using IPOPT with gradients computed using the adjoint equation method and the Hessian matrix computed using several methods including hyper-dual numbers.

Nomenclature

f	generic function, Objective Function
f'	first derivative of f
f''	second derivative of f
x, y	generic scalar value; not necessarily real numbers
$\mathbf{x}$	vector of scalar values, Design Variables
$\mathbf{e}_i$, $\mathbf{e}_j$, $\mathbf{e}_k$	unit vector with indicated element equal to one and other elements equal to zero
n	dimension of $\mathbf{x}$
$\nabla f(\mathbf{x})$	gradient of $f(\mathbf{x})$
$\nabla^2 f(\mathbf{x})$	Hessian of $f(\mathbf{x})$
h, h_1, h_2	step sizes
$\epsilon, \epsilon_1, \epsilon_2, \epsilon_1\epsilon_2$	hyper-dual number terms
x_1, x_2, x_3, x_4	real-valued elements of hyper-dual number x
$\mathbf{q}$	flow variables in a computational fluid dynamics code
$\mathbf{b}$	vector of residuals in a computational fluid dynamics code
$\mathbf{A}$	Jacobian matrix in a computational fluid dynamics code
c_l	aerodynamic lift coefficient
c_d	aerodynamic drag coefficient
M	free stream Mach number
α, AOA	free stream angle of attack

*Graduate Student, Department of Aeronautics and Astronautics, Stanford University, Stanford CA, AIAA Student Member
†Graduate Student, Department of Engineering Technology, University of Twente, the Netherlands, AIAA Student Member
‡Associate Professor, Department of Aeronautics and Astronautics, Stanford University, Stanford CA, AIAA Member
§Assistant Professor, Department of Engineering Technology, University of Twente, the Netherlands, AIAA Member

I. Introduction

Numerical optimization methods systematically vary the inputs to a function in order to find the maximum or minimum value of the function. This typically requires very many evaluations of the objective function. Optimization methods that make use of gradient information typically converge to the optimal solution in fewer iterations than methods that only rely on the function values. This of course depends on the smoothness of the function and the ability to compute derivatives. Further benefit may be obtained if the Hessian matrix containing second derivative information is also used.

Newton's method uses both the gradient and Hessian of the objective function,

$$\nabla^2 f(\mathbf{x})\Delta\mathbf{x} = -\nabla f(\mathbf{x}). \tag{1}$$

For a well-behaved function, Newton's method converges to the optimal solution quadratically. However, the number of iterations required for convergence is not the only concern. The effort required to compute the gradient and Hessian must also be taken into account. The steepest descent method converges linearly, but only requires calculation of the gradient. Quasi-Newton methods use the gradient vector at each iteration to construct an approximation to the Hessian matrix, and are capable of superlinear convergence. Some of the benefit of using second-derivative information is retained, while avoiding the cost of explicitly computing the Hessian.

Aside from the issue of computational cost, calculating the Hessian for a complicated objective function is often avoided because of issues with accurately computing second derivatives. Hyper-dual numbers are a number system created to produce exact second derivatives.[1] This is not the most efficient approach to computing derivatives, however it is easy to implement and produces first and second derivatives that are exact to numerical precision. There is an inherent increase in computational effort associated with the mathematics of hyper-dual numbers.

Approaches for reducing the computational cost of using hyper-dual numbers are possible for certain types of functions. To use hyper-dual numbers to calculate derivatives, a real valued analysis code must be modified to use hyper-dual numbers. The naive approach to using hyper-dual numbers is to modify every function to operate on hyper-dual numbers. A more efficient approach is to analyze the code and determine which functions need to be modified. For functions involving iterative procedures, the computational cost can be greatly reduced, increasing the appeal of using hyper-dual numbers compared to finite difference formulas. The strategy for using hyper-dual numbers with an iterative procedure is to first converge the solution using real numbers. Once the solution has converged, one iteration is performed using hyper-dual numbers to compute the derivatives.

This approach is demonstrated on two optimization problems: a low-fidelity supersonic business jet design(SSBJ) and a transonic, inviscid airfoil shape optimization using a Computational Fluid Dynamics(CFD) code. For the low-fidelity SSBJ case, the computational cost is reduced by a factor of 8 over the naive approach. This reduces the computational cost to below that of using finite differences.

The transonic, inviscid airfoil design is carried out using a 2D Euler method. For this problem, the exact Jacobian matrix is computed from the real-valued converged solution, for which only the LU decomposition is stored. This LU decomposition is subsequently used to obtain a converged flow solution for the hyper-dual numbers. In addition to using hyper-dual numbers, two other methods are employed to determine the entries of the Hessian matrix; and the performance is compared.

II. Hyper-Dual Numbers

Hyper-dual numbers are a number system created to produce exact second derivatives.[1] Using hyper-dual numbers to compute second derivatives is similar to using complex numbers to compute first derivatives via the complex-step derivative approximation(CSDA).[2] The use of these techniques involves converting a real-valued function evaluation to operate on these alternative types of numbers. Derivatives are computed by adding a perturbation to the non-real parts of an input and evaluating the modified function. Every operation involved in the real-valued function evaluation is modified to operate on these alternative types of numbers. The mathematics of these alternative types of numbers are such that when operations are carried out on the real part of the number, derivative information for those operations is formed and stored in the non-real parts of the number. At every stage of the function evaluation the non-real parts of the number contain derivative information with respect to the input.[3] This process must be repeated for every input

variable combination for which derivative information is desired. These techniques possess advantages over computing derivatives using finite-difference formulas.

Finite-difference formulas such as the forward-difference(FD) approximation,

$$\frac{\partial f(\mathbf{x})}{\partial x_j} = \frac{f(\mathbf{x} + h\mathbf{e}_j) - f(\mathbf{x})}{h} + O(h), \tag{2}$$

and the central-difference(CD) approximation,

$$\frac{\partial f(\mathbf{x})}{\partial x_j} = \frac{f(\mathbf{x} + h\mathbf{e}_j) - f(\mathbf{x} - h\mathbf{e}_j)}{2h} + O(h^2), \tag{3}$$

are subject to both truncation error and subtractive cancellation error. Truncation error is associated with the higher order terms that are ignored when forming the approximation. Subtractive cancellation error is a result of performing these calculations on a computer with finite precision. The first-derivative complex-step approximation,

$$\frac{\partial f(\mathbf{x})}{\partial x_j} = \frac{\mathrm{Im}\left[f(\mathbf{x} + ih\mathbf{e}_j)\right]}{h} + O(h^2), \tag{4}$$

produces first derivatives that are subject to truncation error but are not subject to subtractive cancellation error. There is no subtractive cancellation error in the complex-step approximation because the first derivative information is the leading term of the imaginary part. As such, an approximation can be formed by simply examining the imaginary part of the complex-valued function evaluation. Due to the fact that no subtractive cancellation error is present, the imaginary disturbance can be chosen arbitrarily small, effectively leading to a zero truncation error.

Hyper-dual numbers are a higher dimensional extension of dual numbers[4,5] in the same way that quaternions are a higher-dimensional extension of ordinary complex numbers. Dual numbers and ordinary complex numbers are types of generalized-complex numbers.[6] Dual numbers are based on the non-real term $\epsilon^2 = 0$ where $\epsilon \neq 0$,[7,8] while ordinary complex numbers are based on $i^2 = -1$. A hyper-dual number is of the form

$$x = x_1 + x_2\epsilon_1 + x_3\epsilon_2 + x_4\epsilon_1\epsilon_2, \tag{5}$$

with one real part and three non-real parts. These non-real parts have the properties $\epsilon_1^2 = \epsilon_2^2 = 0$ where $\epsilon_1 \neq \epsilon_2 \neq 0$ and $\epsilon_1\epsilon_2 = \epsilon_2\epsilon_1 \neq 0$. The definition of ϵ_1 and ϵ_2 means that the Taylor series for a scalar function with a hyper-dual step truncates exactly at the second-derivative term,

$$f(x + h_1\epsilon_1 + h_2\epsilon_2 + 0\epsilon_1\epsilon_2) = f(x) + h_1 f'(x)\epsilon_1 + h_2 f'(x)\epsilon_2 + h_1 h_2 f"(x)\epsilon_1\epsilon_2. \tag{6}$$

The higher order terms are all zero by the definition of $\epsilon_1^2 = \epsilon_2^2 = 0$, so there is no truncation error. The first and second derivatives are the leading terms of the non-real parts, meaning that these values can simply be found by examining the non-real parts of the number, so the derivative calculations are not subject to subtractive cancellation error. Therefore, the use of hyper-dual numbers results in first and second derivative calculations that are exact, regardless of the step size. The real part is also exactly the same as the function evaluated for a real number, x. Information flows in only one direction, from the real part, to the ϵ_1 and ϵ_2 parts, and on to the $\epsilon_1\epsilon_2$ part. It is mathematically impossible for the derivative calculations to affect the real part.

The calculation of first and second derivatives using hyper-dual numbers is similar to the use of the complex-step approximation for computing first derivatives, Eq. (4). The function is evaluated with a hyper-dual number input, as in Eq. (6), and the derivative information is extracted from the non-real terms. First derivatives are found using

$$\frac{\partial f(\mathbf{x})}{\partial x_i} = \frac{\epsilon_1 part\left[f(\mathbf{x} + h_1\epsilon_1\mathbf{e}_i + h_2\epsilon_2\mathbf{e}_j + \mathbf{0}\epsilon_1\epsilon_2)\right]}{h_1} \tag{7}$$

and

$$\frac{\partial f(\mathbf{x})}{\partial x_j} = \frac{\epsilon_2 part\left[f(\mathbf{x} + h_1\epsilon_1\mathbf{e}_i + h_2\epsilon_2\mathbf{e}_j + \mathbf{0}\epsilon_1\epsilon_2)\right]}{h_2}. \tag{8}$$

And second derivatives are found using

$$\frac{\partial^2 f(\mathbf{x})}{\partial x_i \partial x_j} = \frac{\epsilon_1\epsilon_2 part\left[f(\mathbf{x} + h_1\epsilon_1\mathbf{e}_i + h_2\epsilon_2\mathbf{e}_j + \mathbf{0}\epsilon_1\epsilon_2)\right]}{h_1 h_2}. \tag{9}$$

Figure 1 shows the accuracy of the derivative calculation methods discussed above as the step size is decreased for a simple analytic function. As the step size is initially decreased, the error decreases according to the order of the truncation error. As the step size decreases beyond a certain point, the error starts to increase for those methods that are subject to subtractive cancellation. The optimal step size for these methods requires a trade off between the truncation and subtractive cancellation errors. This optimal step size is not known a priori, and may require substantial effort and knowledge of the true derivative value. Unfortunately, the optimal step size is not a constant value. It will change depending on the function, the point where the derivative is desired, and the variable with which the derivative is calculated. Methods such as hyper-dual numbers and the first-derivative complex-step approximation do not require an optimal step size to be found.

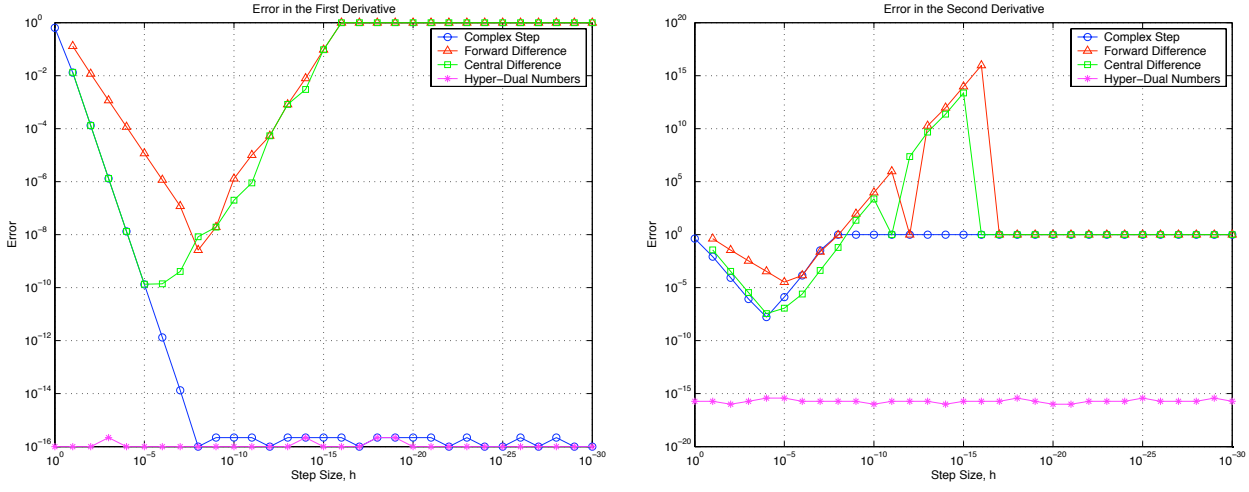
Hyper-dual numbers have been implemented as a class, using operator overloading, in C++ and MATLAB. These implementations are available at Ref. 9. The implementations include definitions for the standard algebraic operations, logical comparison operations, and other more general functions such as the exponential, sine, and absolute value. The use of operator overloading allows a real-valued analysis routine to be easily converted to operate on hyper-dual numbers by essentially just changing the variable type declarations. The structure and details of the code remain unchanged.

II.A. Computational Cost

Forming the gradient of $f(\mathbf{x})$, for $\mathbf{x} \in \mathbb{R}^n$, requires the calculation of n first derivatives. Using the forward-difference approximation requires $n+1$ function evaluations. The central-difference approximation requires $2n$ function evaluations. The complex-step approximation requires only n complex function evaluations. However, each complex function evaluation involves more work. Adding two complex numbers is equivalent to 2 real additions, while multiplying two complex numbers is equivalent to 4 real multiplications and 3 real additions. Therefore a complex function evaluation should take about 2 to 4 times the runtime of a real function evaluation.

Forming the gradient using hyper-dual numbers requires n hyper-dual function evaluations. As with complex numbers, working with hyper-dual numbers requires additional computational work. Adding hyper-dual numbers is equivalent to 4 real additions, and multiplying two hyper-dual numbers is equivalent to 9 real multiplications and 5 real additions. Therefore a hyper-dual function evaluation should take about 4 to 14 times the runtime of a real function evaluation.

Forming the Hessian of $f(\mathbf{x})$, for $\mathbf{x} \in \mathbb{R}^n$, requires $\frac{n(n+1)}{2}$ second derivative calculations. The forward difference approximation requires $(n+1)^2$ function evaluations. The central difference approximation requires $2n(n+1)$ function evaluations. The use of hyper-dual numbers requires $\frac{n(n+1)}{2}$ hyper-dual function evalua-



(a) First-Derivative Accuracy.

(b) Second-Derivative Accuracy.

Figure 1. The accuracies of several derivative calculation methods are presented as a function of step size for the function $f(x) = \frac{e^x}{\sqrt{sin(x)^3+cos(x)^3}}$.

tions, which as discussed above is expected to be the equivalent of up to $7n(n+1)$ real function evaluations. So the use of hyper-dual numbers is expected to be around 3.5 times as expensive as the central-difference approximation, which is itself about twice as expensive as the forward-difference approximation.

In many situations, such as numerical optimization using Newton's method, both the gradient and Hessian need to be computed. This does not change the results for the use of hyper-dual numbers or the forward difference approximation. This is because the required function evaluations are used for both the gradient and Hessian calculations. However, the central-difference approximation now requires $2n(n+2)$ function evaluations.

The aforementioned naive approach to using hyper-dual numbers is to modify every function to operate on hyper-dual numbers. Following this approach will lead to the performance estimates discussed above. A more efficient approach is to analyze the code and determine which functions need to be modified. In particular, when the code involves an iterative procedure the computational cost of using hyper-dual numbers can be greatly reduced. The strategy for using hyper-dual numbers with an iterative procedure is to first converge the solution using real numbers. Once the solution has converged, one iteration is performed using hyper-dual numbers to compute the derivatives.

III. Supersonic Business Jet Design Optimization

A supersonic business jet(SSBJ) design is optimized using Newton's method. The objective function for this design is a weighted combination of aircraft range and the sonic boom strength at the ground. This optimization was performed using three methods for computing the gradient and Hessian: the forward difference and central difference approximations, and hyper-dual numbers.

III.A. Objective Function Descriptions

III.A.1. Aircraft Range

The range of the SSBJ is calculated using tools that are appropriate for a conceptual design level analysis. These routines were provided by the authors of the multi-disciplinary optimization algorithm BLISS(Bi-Level Integrated System Synthesis).[10] These are low fidelity analysis routines, but they capture the trends and can be run very quickly. For the present work, these coupled disciplinary analysis routines were combined into a single monolithic function. This requires adding two target variables in order to allow the individual routines to be executed sequentially.

The monolithic routine begins by running the power analysis routine. This routine uses the throttle setting and the drag target of the aircraft to size the engines. The engine sizing is done using a precomputed surface fit. The power routine outputs the engine weight, specific fuel consumption, and engine scale factor. There are inequality constraints on the engine weight, engine scale factor, and engine temperature. This information is then fed into the weight analysis routine. This routine uses the lift target, the geometry, and structural thicknesses, and the computed engine weight to compute the total weight, fuel weight fraction, and wing twist. Inequality constraints are computed on the stresses in the structural members and the twist. The information computed so far is then fed into the aerodynamic analysis routine. This routine computes the lift and drag of the aircraft by assuming elliptical loading and a parabolic drag polar, and imposes a trim constraint. Equality constraints are needed during the optimization in order to ensure the consistency of the design. These equality constraints enforce that the lift and drag, computed at the end of the analysis, match the target values used to start the analysis. The range is then calculated using the Breguet range equation,

$$R = Ma\left(\frac{L}{D}\right)\left(\frac{1}{SFC}\right)\left(-log\left(1-\frac{W_f}{W_t}\right)\right), \tag{10}$$

where M is the Mach number, a is the speed of sound, $\frac{L}{D}$ is the lift to drag ratio, SFC is the specific fuel consumption, W_f is the fuel weight, and W_t is the total weight of the aircraft.

III.A.2. Sonic Boom

The sonic boom computation consists of several steps. An aircraft shape factor[11] is found based on the coarse 33 variable description of the design. This method involves creating an effective area distribution, $A_e(x)$, by combining the cross sectional area distribution and an equivalent area distribution due to lift. The

cross sectional area distribution is determined from cross sections normal to the longitudinal axis. This is in contrast to other methods that require slices at the Mach angle. The equivalent area due to lift is determined at every point along the length of the aircraft using

$$B(x) = \frac{W_t\sqrt{M^2-1}}{\gamma p S_{ref} M^2}\int_0^x b(x)dx, \tag{11}$$

where p is the pressure at altitude, W_t is the weight of the aircraft, and $b(x)$ is the span at a given location.

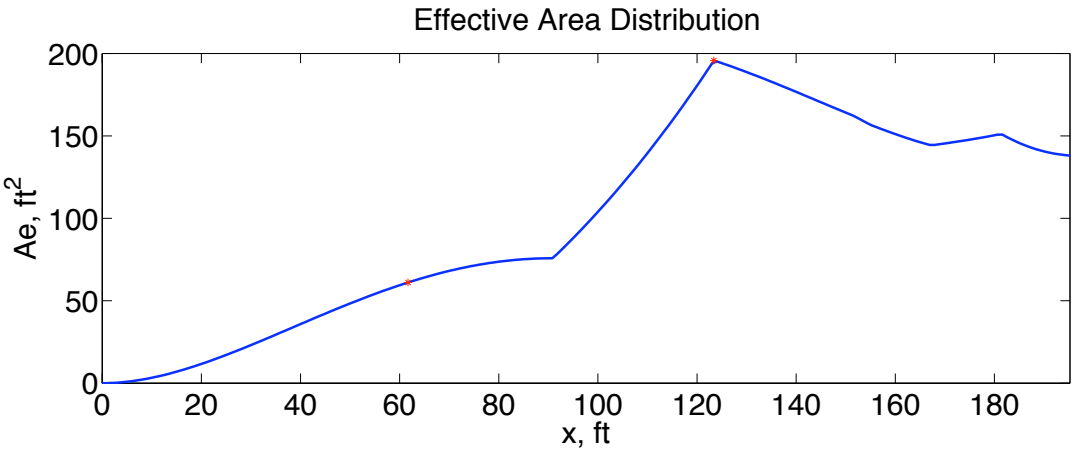


Figure 2. Effective area distribution for the optimal SSBJ design.

The aircraft shape factor, K_S, requires determining the maximum value of the equivalent area distribution, $A_e(l_e)$, where l_e is the location of the maximum equivalent area. A typical effective area curve is shown in Fig. 2 with the values of $A_e(l_e)$ and $A_e(0.5l_e)$ highlighted. Using this information, the shape factor is calculated using

$$K_S = Y_{SF}\frac{\sqrt{A_e(l_e)}}{l^{\frac{3}{4}}l_e^{\frac{1}{4}}}, \tag{12}$$

where l is the length of the aircraft and Y_{SF} is a coefficient found using $A_e(l_e)/A_e(0.5l_e)$. This shape factor is then related to an initial pressure signature at cruise altitude. The initial pressure signature is an N-wave of length l and magnitude

$$\frac{\Delta p}{p} = 3.46\sqrt{l}K_S^2. \tag{13}$$

This initial pressure signature is then propagated to the ground using the Waveform Parameter Method.[12] This method is based on ideas from geometric acoustics. This sonic boom propagation method involves determining ray tube paths from the cruise altitude, through a non-uniform atmosphere, to the ground. The waveform parameter method discretizes a general pressure signal into a number of straight line segments. These segments are specified by parameters defining the slope and duration of each segment, and the jump in pressure between the segments. Differential equations relate the evolution of these parameters as the waveform propagates along the ray path. These equations account for attenuation, steepening, stretching, and coalescence of the segments. This method handles arbitrary pressure signatures, however, since the initial pressure signature is an N-wave, the waveform will only experience stretching and attenuation.

III.B. Optimization Results

Newton's method is used to solve this optimization problem. This optimization problem involves 33 design variables and 45 design constraints. Of the 45 design constraints, 2 are equality constraints and 43 are inequality constraints. There are also lower and upper bound constraints on the 33 design variables. The constraints are handled using quadratic penalty functions, which means that the constraints may be slightly violated.

The 33 design variables provide a coarse description of the aircraft geometry, operating conditions, and interior structure. The design variables and their optimal values are given in Table 1. These include the

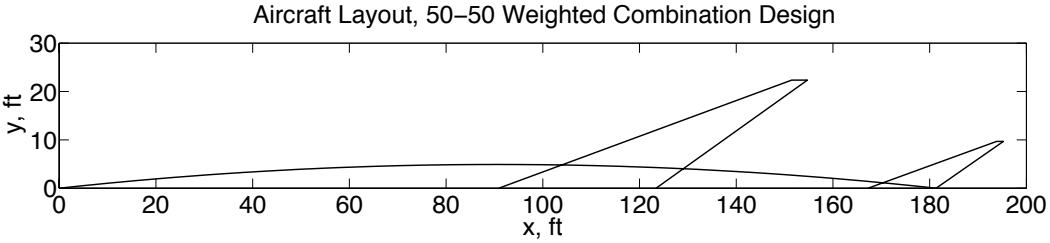


Figure 3. Optimal supersonic business jet configuration for a 50-50 weighted combination of the two objectives, maximizing range and minimizing the sonic boom at the ground.

location, area, aspect ratio, and sweep for both the wing and horizontal tail, as well as the thickness to chord ratio and taper ratio. There are 18 variables that specify the thicknesses of the structural elements of the wing. The altitude, Mach number, and throttle setting determine the operating conditions. There are also two variables, a drag target and a lift target, which are used to ensure consistency of the design. The optimal design is shown in Fig. 3, and the performance of the design is given in Table 2.

III.C. Hyper-Dual Calculations

Two different versions of these analysis tools were created: a version using real numbers and a version using hyper-dual numbers. The SSBJ was then optimized using Newton's method with the gradients and Hessians calculated using both finite difference formulas and hyper-dual numbers. Using hyper-dual numbers, this optimization initially took 7 times as long as using forward differences and 3.6 times as long as using central differences. This is consistent with the discussion in Section II.A, and was expected. This original hyper-dual version was created following the naive approach of overloading every function.

A more efficient version can be created by modifying a small part of one routine. During the sonic boom calculation, the location of the maximum effective area is determined. The maximum value is found using a golden-section line search, however this value could just as easily be found by many different techniques. The procedure used to find the maximum value should therefore not affect the derivative calculations. Only the value itself matters, not the path taken to find it. The location of the maximum effective area can therefore be found using real numbers. Once the value is found, a single calculation can be run using hyper-dual numbers to determine the derivatives.

This change decreases the runtime of the hyper-dual version of the code by a factor of 8. This means that the runtime for the optimization using hyper-dual numbers was reduced to 0.9 times the forward-difference runtime and 0.46 times the central-difference runtime. For this problem, using hyper-dual numbers is faster than using finite differences *and* the gradient and Hessian calculations are exact.

IV. Application to a Computational Fluid Dynamics Code

This section describes the use of hyper-dual numbers to compute the derivatives of a CFD code. This discussion is general in nature, but the approach is applied to JOE,[13] a parallel, unstructured, 3-D, unsteady Reynolds-Averaged Navier-Stokes code. This code is written in C++, which enables the straightforward conversion to hyper-dual numbers.

The conversion of a real valued analysis code involves modifying or overloading each function to operate on hyper-dual numbers. This requires modifications to the source code. However, CFD codes, such as JOE, often make use of external libraries for which the source code is not available. For instance JOE can make use of the generalized minimal residual method(GMRES) from the Portable, Extensible Toolkit for Scientific Computation(PETSc).[14] This routine can still be used, without overloading it, by looking at how the routine is called and defining a hyper-dual version of this function call.

IV.A. Differentiation of the Solution of a Linear System

At the core of an implicit CFD code is the solution of a linear system to solve for the updates of the flow variables. In a steady RANS solver the goal is to drive the residuals to zero, $\mathbf{b}(\mathbf{q}, \mathbf{x}) = \mathbf{0}$, where $\mathbf{q}$ are the

Variable	Lower Bound	Optimal Design	Upper Bound
Thickness to Chord Ratio, t/c	0.0500	0.1200	0.1200
Cruise Altitude, h (ft.)	45,000	60,005	60,000
Cruise Mach Number, M	1.4000	1.3999	1.8000
Wing Aspect Ratio, AR_W	2.5000	2.4999	5.0000
Wing Sweep, Λ_W (°)	40.0000	69.7390	70.0000
Wing Area, S_W (ft^2)	200.0000	800.0074	800.0000
Horizontal Tail Area, S_{HT} (ft^2)	50.0000	149.9999	150.0000
Horizontal Tail Aspect Ratio, AR_{HT}	2.5000	2.5000	8.5000
Wing Box Face Thickness, t_1	0.1000	0.5369	0.9500
Wing Box Face Thickness, t_2	0.1000	0.4723	0.9500
Wing Box Face Thickness, t_3	0.1000	0.3789	0.9500
Wing Box Face Thickness, t_4	0.1000	0.9500	0.9500
Wing Box Face Thickness, t_5	0.1000	0.9500	0.9500
Wing Box Face Thickness, t_6	0.1000	0.9500	0.9500
Wing Box Face Thickness, t_7	0.1000	0.5369	0.9500
Wing Box Face Thickness, t_8	0.1000	0.4723	0.9500
Wing Box Face Thickness, t_9	0.1000	0.3789	0.9500
Wing Box Caliper Thickness, ts_1	0.1000	1.6374	9.0000
Wing Box Caliper Thickness, ts_2	0.1000	1.1649	9.0000
Wing Box Caliper Thickness, ts_3	0.1000	0.6115	9.0000
Wing Box Caliper Thickness, ts_4	0.1000	9.0000	9.0000
Wing Box Caliper Thickness, ts_5	0.1000	3.9419	9.0000
Wing Box Caliper Thickness, ts_6	0.1000	0.9363	9.0000
Wing Box Caliper Thickness, ts_7	0.1000	1.6374	9.0000
Wing Box Caliper Thickness, ts_8	0.1000	1.1649	9.0000
Wing Box Caliper Thickness, ts_9	0.1000	0.6115	9.0000
Taper Ratio, λ	0.1000	0.1000	0.4000
Horizontal Tail Sweep, Λ_{HT} (°)	40.0000	69.9995	70.0000
Wing Location, L_W (% MAC)	0.0100	0.0100	0.2000
Horizontal Tail Location, L_{HT} (% MAC)	1.0000	3.5000	3.5000
Throttle (%)	0.1000	0.1562	1.0000
Drag Target (lb.)	100	7,577	70,000
Lift Target (lb.)	5,000	58,346	100,000

Table 1. The design variables for the supersonic business jet design, their optimal values, and the upper and lower bounds. Many of the bound constraints are active. The constraints may be slightly violated since they are enforced using quadratic penalty functions.

Aircraft Range (nm)	Maximum Ground Pressure (psf)
4,055.6	0.7891

Table 2. Values of the two objective functions for the optimal SSBJ design.

flow variables and $\mathbf{x}$ are the design variables. For an implicit solver this is accomplished by solving

$$\mathbf{A}(\mathbf{x})\mathbf{dq}(\mathbf{x}) = \mathbf{b}(\mathbf{x}), \tag{14}$$

where $\mathbf{b}$ is the vector of residuals, and $\mathbf{A}$ is the Jacobian matrix.

Differentiating both sides with respect to the i^{th} component of $\mathbf{x}$ gives

$$\frac{\partial \mathbf{A}(\mathbf{x})}{\partial x_i}\mathbf{dq}(\mathbf{x}) + \mathbf{A}(\mathbf{x})\frac{\partial \mathbf{dq}(\mathbf{x})}{\partial x_i} = \frac{\partial \mathbf{b}(\mathbf{x})}{\partial x_i}. \tag{15}$$

Differentiating this result with respect to the j^{th} component of $\mathbf{x}$ gives

$$\frac{\partial^2 \mathbf{A}(\mathbf{x})}{\partial x_j \partial x_i}\mathbf{dq}(\mathbf{x}) + \frac{\partial \mathbf{A}(\mathbf{x})}{\partial x_i}\frac{\partial \mathbf{dq}(\mathbf{x})}{\partial x_j} + \frac{\partial \mathbf{A}(\mathbf{x})}{\partial x_j}\frac{\partial \mathbf{dq}(\mathbf{x})}{\partial x_i} + \mathbf{A}(\mathbf{x})\frac{\partial^2 \mathbf{dq}(\mathbf{x})}{\partial x_j \partial x_i} = \frac{\partial^2 \mathbf{b}(\mathbf{x})}{\partial x_j \partial x_i}. \tag{16}$$

This can be solved as

$$\begin{bmatrix} \mathbf{A}(\mathbf{x}) & 0 & 0 & 0 \\ \frac{\partial \mathbf{A}(\mathbf{x})}{\partial x_i} & \mathbf{A}(\mathbf{x}) & 0 & 0 \\ \frac{\partial \mathbf{A}(\mathbf{x})}{\partial x_j} & 0 & \mathbf{A}(\mathbf{x}) & 0 \\ \frac{\partial^2 \mathbf{A}(\mathbf{x})}{\partial x_j \partial x_i} & \frac{\partial \mathbf{A}(\mathbf{x})}{\partial x_j} & \frac{\partial \mathbf{A}(\mathbf{x})}{\partial x_i} & \mathbf{A}(\mathbf{x}) \end{bmatrix} \begin{Bmatrix} \mathbf{dq}(\mathbf{x}) \\ \frac{\partial \mathbf{dq}(\mathbf{x})}{\partial x_i} \\ \frac{\partial \mathbf{dq}(\mathbf{x})}{\partial x_j} \\ \frac{\partial^2 \mathbf{dq}(\mathbf{x})}{\partial x_j \partial x_i} \end{Bmatrix} = \begin{Bmatrix} \mathbf{b}(\mathbf{x}) \\ \frac{\partial \mathbf{b}(\mathbf{x})}{\partial x_i} \\ \frac{\partial \mathbf{b}(\mathbf{x})}{\partial x_j} \\ \frac{\partial^2 \mathbf{b}(\mathbf{x})}{\partial x_j \partial x_i} \end{Bmatrix}. \tag{17}$$

When using hyper-dual numbers, the formation of the matrix $\mathbf{A}$ produces

$$\mathbf{A} = \mathbf{A}(\mathbf{x}) + \frac{\partial \mathbf{A}(\mathbf{x})}{\partial x_i}\epsilon_1 + \frac{\partial \mathbf{A}(\mathbf{x})}{\partial x_j}\epsilon_2 + \frac{\partial^2 \mathbf{A}(\mathbf{x})}{\partial x_j \partial x_i}\epsilon_1\epsilon_2, \tag{18}$$

and the formation of $\mathbf{b}$ produces

$$\mathbf{b} = \mathbf{b}(\mathbf{x}) + \frac{\partial \mathbf{b}(\mathbf{x})}{\partial x_i}\epsilon_1 + \frac{\partial \mathbf{b}(\mathbf{x})}{\partial x_j}\epsilon_2 + \frac{\partial^2 \mathbf{b}(\mathbf{x})}{\partial x_j \partial x_i}\epsilon_1\epsilon_2. \tag{19}$$

The new larger matrix and right-hand side defined in Eq. (17) can simply be formed by extracting the information from the hyper-dual versions of $\mathbf{A}$ and $\mathbf{b}$. Once this equation is solved, the elements of the new larger $\mathbf{dq}$ vector can be used to form the hyper-dual version of $\mathbf{q}$.

There is an alternative way of solving this. Rather than solving one large matrix equation, a more efficient approach is to solve a series of smaller problems. The first step is to solve

$$\mathbf{A}(\mathbf{x})\mathbf{dq}(\mathbf{x}) = \mathbf{b}(\mathbf{x}). \tag{20}$$

Then the first derivatives can be computed using

$$\mathbf{A}(\mathbf{x})\frac{\partial \mathbf{dq}(\mathbf{x})}{\partial x_i} = \frac{\partial \mathbf{b}(\mathbf{x})}{\partial x_i} - \frac{\partial \mathbf{A}(\mathbf{x})}{\partial x_i}\mathbf{dq}(\mathbf{x}) \tag{21}$$

and

$$\mathbf{A}(\mathbf{x})\frac{\partial \mathbf{dq}(\mathbf{x})}{\partial x_j} = \frac{\partial \mathbf{b}(\mathbf{x})}{\partial x_j} - \frac{\partial \mathbf{A}(\mathbf{x})}{\partial x_j}\mathbf{dq}(\mathbf{x}). \tag{22}$$

Finally, the second derivatives can be calculated using

$$\mathbf{A}(\mathbf{x})\frac{\partial^2 \mathbf{dq}(\mathbf{x})}{\partial x_j \partial x_i} = \frac{\partial^2 \mathbf{b}(\mathbf{x})}{\partial x_j \partial x_i} - \frac{\partial^2 \mathbf{A}(\mathbf{x})}{\partial x_j \partial x_i}\mathbf{dq}(\mathbf{x}) - \frac{\partial \mathbf{A}(\mathbf{x})}{\partial x_i}\frac{\partial \mathbf{dq}(\mathbf{x})}{\partial x_j} - \frac{\partial \mathbf{A}(\mathbf{x})}{\partial x_j}\frac{\partial \mathbf{dq}(\mathbf{x})}{\partial x_i}. \tag{23}$$

Solving these four matrix equations is more efficient than solving the large matrix equation given in Eq. (17). The computational effort required to solve a matrix equation is typically quoted as being on the order of the size of the matrix raised to some power a ($a > 1$), where the value of the exponent depends on the algorithm used. The cost of solving the four smaller matrix equations can be written as $O(4(N^a))$, which is expected to be smaller than $O((4N)^a)$. There are also reduced memory requirements when solving the four smaller matrix equations. The large system does not need to be stored, and temporary variables

can be reused when forming the right hand sides. This second approach is how PETSc GMRES is used in the hyper-dual version of JOE.

In practice, a hyper-dual JOE run using this approach takes roughly 10 times that of a real-valued JOE run. Computing the Hessian using hyper-dual numbers requires 5 times the time required to use forward differences, and 2.5 times the time required to use central differences. This is reasonable based on the discussion above. The actual value depends greatly on the compiler and optimization flags that are used. It has been observed to be as low as 4 or as high as 2000, but the average is around 10. This makes computing each second derivative on the order of 10 times as expensive as running the flow solution

IV.B. Approach for Iterative Procedures

CFD codes are inherently iterative procedures. If we assume that we start with a converged real-valued solution, the above procedure can be simplified to reduce the computational effort. When starting with a converged solution, the real part is held fixed and only the derivative terms are updated.

The first step in the procedure described above is to solve

$$\mathbf{A}(\mathbf{x})\mathbf{dq}(\mathbf{x}) = \mathbf{b}(\mathbf{x}). \tag{24}$$

However, for a converged solution, $\mathbf{dq}(\mathbf{x}) \equiv \mathbf{0}$, so this matrix equation does not need to be solved. The next step would be to solve

$$\mathbf{A}(\mathbf{x})\frac{\partial \mathbf{dq}(\mathbf{x})}{\partial x_i} = \frac{\partial \mathbf{b}(\mathbf{x})}{\partial x_i} - \frac{\partial \mathbf{A}(\mathbf{x})}{\partial x_i}\mathbf{dq}(\mathbf{x}). \tag{25}$$

However, by following similar logic, this reduces to

$$\mathbf{A}(\mathbf{x})\frac{\partial \mathbf{dq}(\mathbf{x})}{\partial x_i} = \frac{\partial \mathbf{b}(\mathbf{x})}{\partial x_i}, \tag{26}$$

and similarly,

$$\mathbf{A}(\mathbf{x})\frac{\partial \mathbf{dq}(\mathbf{x})}{\partial x_j} = \frac{\partial \mathbf{b}(\mathbf{x})}{\partial x_j}. \tag{27}$$

The second derivative calculation becomes

$$\mathbf{A}(\mathbf{x})\frac{\partial^2 \mathbf{dq}(\mathbf{x})}{\partial x_j \partial x_i} = \frac{\partial^2 \mathbf{b}(\mathbf{x})}{\partial x_j \partial x_i} - \frac{\partial \mathbf{A}(\mathbf{x})}{\partial x_i}\frac{\partial \mathbf{dq}(\mathbf{x})}{\partial x_j} - \frac{\partial \mathbf{A}(\mathbf{x})}{\partial x_j}\frac{\partial \mathbf{dq}(\mathbf{x})}{\partial x_i}. \tag{28}$$

If we now assume that we have converged the first derivative terms, then this reduces to

$$\mathbf{A}(\mathbf{x})\frac{\partial^2 \mathbf{dq}(\mathbf{x})}{\partial x_j \partial x_i} = \frac{\partial^2 \mathbf{b}(\mathbf{x})}{\partial x_j \partial x_i}. \tag{29}$$

Following this approach, it should be possible to start from the converged solution of the real-valued code and run one Newton iteration per design variable to compute the gradient.[15,16] Then using the converged first derivative values, each second derivative should also only require one Newton iteration.

IV.C. Comparison of Derivative Calculation Strategies

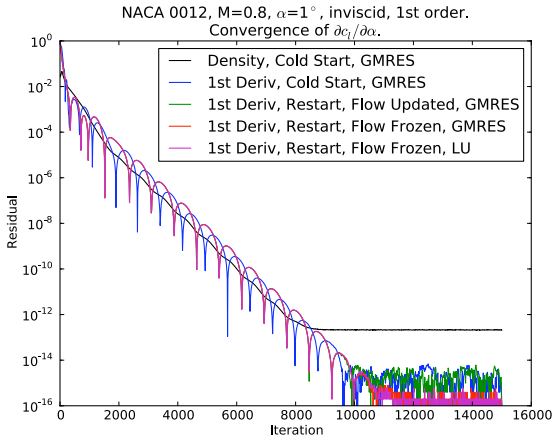
Three strategies for computing the derivative parts are compared:

1. Running the entire flow solver using hyper-dual numbers, converging both the flow solution and the derivatives at the same time.

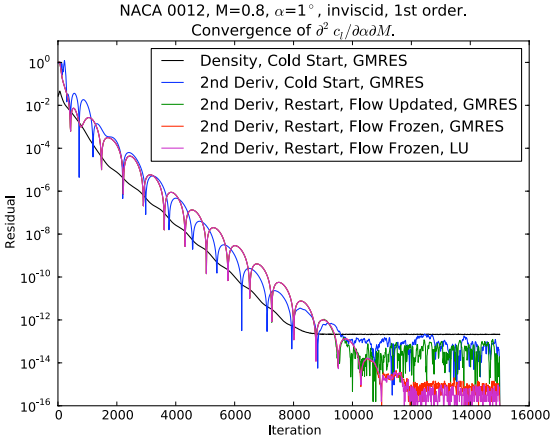
2. First running the flow solver using real numbers until the solution has converged. Then, using this restart file, running the flow solver using hyper-dual numbers to calculate the derivatives, while still allowing the flow to change.

3. First running the flow solver using real numbers until the solution has converged. Then, using this restart file, running the flow solver using hyper-dual numbers to calculate the derivatives, while keeping the flow fixed.

The results using JOE are shown in Fig. 4. In these plots the density residual from the real number solution is shown to indicate how the convergence of the derivatives compares to the convergence of the flow solver itself. As can be seen in these figures, there is little apparent benefit to starting with a converged solution. Strategy 3 may be more efficient computationally, but this requires changes to the code. Strategies 1 and 2 can be run using the same code. The effect of using an iterative solver like GMRES versus a direct LU solve is also explored. This involves running Strategy 3, but changing the PETSc options to perform LU decomposition. This greatly increases the runtime required, but there is no observable difference in convergence rate.



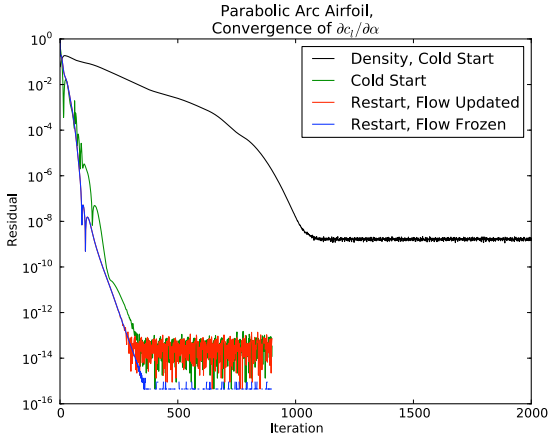
(a) Convergence of $\frac{\partial c_l}{\partial \alpha}$.



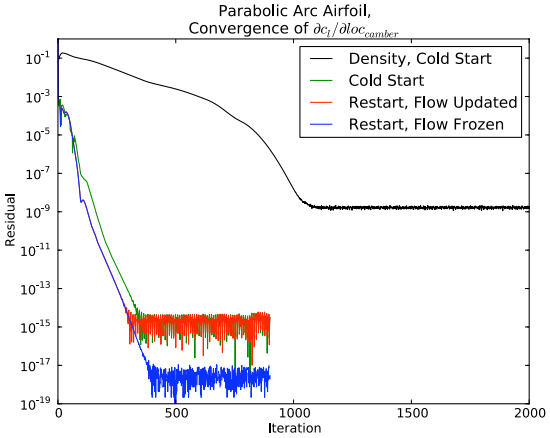
(b) Convergence of $\frac{\partial^2 c_l}{\partial \alpha \partial M}$.

Figure 4. Convergence of first and second derivatives for an inviscid NACA 0012 airfoil using different strategies for computing the derivatives. Also shown is the convergence of the density residual. For this case, the derivatives converge at the same rate as the flow solution, regardless of the strategy for computing the derivatives.

One possible explanation for the slow convergence of the derivatives is that if the hyper-dual perturbation is applied to the free stream conditions then that disturbance will take time to propagate through the flow field. A test was run to see if derivatives with respect to shape variables converge faster than derivatives with respect to free stream conditions, when starting with a converged solution. The idea is that the derivatives of quantities like c_l and c_d, which are computed around a body, may converge faster if we perturb the shape as opposed to the free stream conditions since the disturbances do not have to propagate as far to influence the quantity of interest. However, as shown in Fig. 5, there does not seem to be a difference between shape variables and free stream variables.



(a) Convergence of the first derivative of c_l with respect to camber location.



(b) Convergence of the first derivative of c_l with respect to the angle of attack.

Figure 5. The convergence rate of first derivatives with respect to shape variables compared to free stream variables. In this case, the derivatives converge much faster than the flow solution, but there is still not a significant benefit to starting with a converged solution.

The above results do not show a benefit to starting with a converged solution. The suspected reason for this is that the Jacobian used to solve the linear system is the same approximate Jacobian used by the flow solver rather than the exact Jacobian. When the exact Jacobian is used for a 2D Euler code, the derivatives are observed to converge in one or two iterations, depending on the implementation.

V. Transonic inviscid airfoil optimization

The general definition of an optimization problem is given by

$$\begin{array}{rll} \text{minimize:} & f(\mathbf{x}) & \mathbf{x} \in \mathbb{R}^n \\ \text{subject to:} & C_i^{\text{eq}}(\mathbf{x}) = \alpha_i & i = 1 \dots s \\ & \beta_i^{\text{low}} \leq C_i^{\text{ineq}}(\mathbf{x}) \leq \beta_i^{\text{upp}} & i = 1 \dots t \end{array} \tag{30}$$

where f is the objective function to be minimized, $\mathbf{x}$ the vector of n design variables, $C_i^{\text{eq}}(\mathbf{x})$ the s equality constraints and $C_i^{\text{ineq}}(\mathbf{x})$ the t inequality constraints. For the results presented in section V.B, f is the drag coefficient c_d, $\mathbf{x}$ are the angle of attack and the geometric design variables (see section V.A), and the constraints are the desired lift coefficient c_l as well as geometric constraints. The entire approach is shown in Fig. 6. The actual optimization step is carried out by IPOPT,[17] which is used as a black box. In order

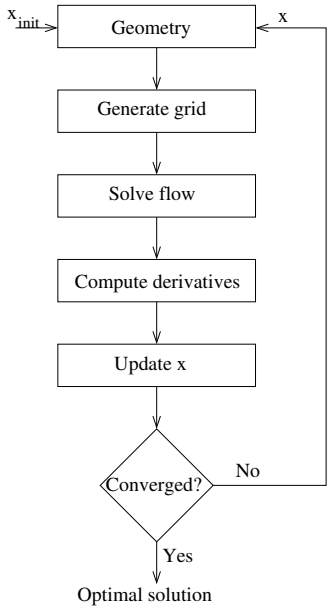


Figure 6. Global overview of the employed optimization algorithm

to perform an optimization, IPOPT requires the value of the objective and constraint function as well as their sensitivities. In addition to this, IPOPT has the option to provide the Hessian matrix, to determine the search directions for the optimization problem. When no Hessian matrix is provided by the user, it will be estimated by IPOPT employing a limited memory BFGS-method.

In order to investigate the potential of the hyper-dual number technique in the scope of aerodynamic shape optimization, it is used to compute the Hessian matrix provided to IPOPT. The performance of the optimizer when specifying the Hessian matrix computed with hyper-dual numbers is compared to the performance when an estimate of the Hessian matrix is employed. Furthermore, two other means to determine the Hessian matrix will be investigated and their performance compared. The first method is an ordinary finite-difference approximation of the second derivative for which the following central difference stencils will be used:

$$\frac{\partial^2 f(\mathbf{x})}{\partial x_j^2} = \frac{-f(\mathbf{x} - 2h\mathbf{e}_j) + 16f(\mathbf{x} - h\mathbf{e}_j) + 30f(\mathbf{x}) + 16f(\mathbf{x} + h\mathbf{e}_j) - f(\mathbf{x} + 2h\mathbf{e}_j)}{12h^2} + \mathcal{O}\left(h^4\right) \tag{31}$$

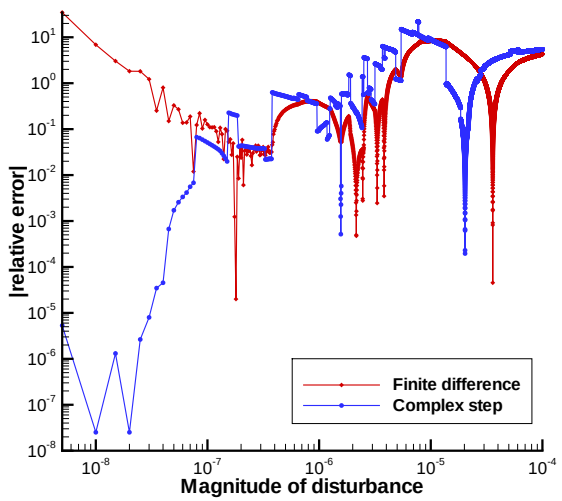
for the matrix entries on the diagonal of the Hessian matrix, and

$$\frac{\partial^2 f(\mathbf{x})}{\partial x_j \partial x_k} = \frac{f(\mathbf{x} + h[\mathbf{e}_j + \mathbf{e}_k]) - f(\mathbf{x} - h[\mathbf{e}_j - \mathbf{e}_k])}{4h^2} - \frac{f(\mathbf{x} + h[\mathbf{e}_j - \mathbf{e}_k]) - f(\mathbf{x} - h[\mathbf{e}_j + \mathbf{e}_k])}{4h^2} + \mathcal{O}(h^2) \quad (32)$$

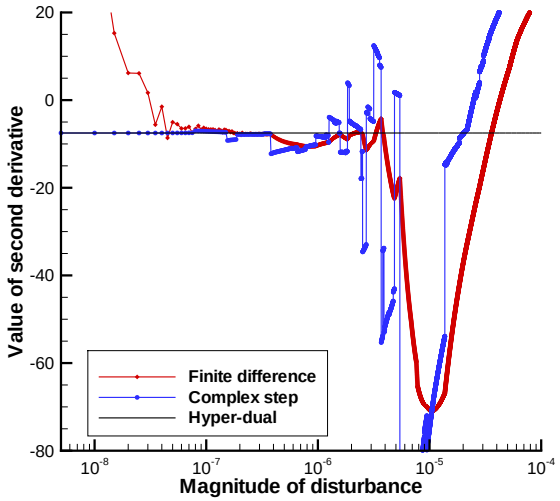
for the off-diagonal entries. The second method uses a combination between an ordinary finite-difference method and the complex-step method. The complex-step method is used to determine the first derivative, while ordinary finite differencing is used for the second derivative. The derivative approximation for this method can be expressed as

$$\frac{\partial^2 f(\mathbf{x})}{\partial x_j \partial x_k} = \frac{\mathrm{Im}[f(\mathbf{x} + ih_1\mathbf{e}_j - 2h_2\mathbf{e}_k)] - \mathrm{Im}[f(\mathbf{x} + ih_1\mathbf{e}_j + 2h_2\mathbf{e}_k)]}{12h_1h_2} + 2\left(\frac{\mathrm{Im}[f(\mathbf{x} + ih_1\mathbf{e}_j + h_2\mathbf{e}_k)] - \mathrm{Im}[f(\mathbf{x} + ih_1\mathbf{e}_j - h_2\mathbf{e}_k)]}{3h_1h_2}\right) + \mathcal{O}(h_1^2 + h_2^4). \quad (33)$$

The magnitude of the imaginary disturbance h_1 is typically chosen of the order 10^{-30} or even smaller. This is possible due to the absence of subtractive cancellation error. On the other hand, for the real valued disturbance h_2 the choice is more critical. Some preliminary tests to determine a suitable value for the disturbance showed a strong dependence of the result on the disturbance applied to the angle of attack. The same behavior was observed for the real valued finite-difference method. Therefore, a more rigorous investigation was performed to settle on a suitable value for the magnitude of the disturbance. The results of this investigation are presented in Fig. 7. The graph on the left shows the absolute value of the relative error of $\frac{\partial^2 c_l}{\partial \alpha^2}$ with respect to the value obtained using the hyper-dual method, i.e. -7.51 for this particular case, for both the real valued and complex-step finite-difference methods. Figure 7(b) compares the actual value of the second derivative for both finite-difference methods.



(a) Absolute relative error of finite-difference approximations with respect to the result obtained with the hyper-dual method, i.e. -7.50583.



(b) Value of $\frac{\partial^2 c_l}{\partial \alpha^2}$ for both finite-difference approximations compared to the value obtained with the hyper-dual method.

Figure 7. Results of the investigation of the sensitivity of the step size on the value of the second derivative of c_l with respect to the angle of attack.

There are multiple combinations of h and h_2 which produce the same erroneous value of the second derivative, as can be observed in Fig. 7(b). Therefore, relying on these approximations to find a suitable value for the magnitude of the disturbance would likely yield the wrong conclusion. Knowledge of the actual

value is required, such as that produced by hyper-dual numbers. Using the result of this investigation, the choice was made to use a disturbance of magnitude $1.0 \cdot 10^{-7}$ for h when the angle of attack is disturbed and $1.0 \cdot 10^{-6}$ otherwise. For the complex-step method, $h_2 = 1.0 \cdot 10^{-8}$ appears suitable for perturbing α. For the other design variables a value of $1.0 \cdot 10^{-7}$ is more suited. The latter choice is based on a less thorough investigation than the one performed for the angle of attack. Hence it is possible that for some of the design variables a more suitable choice exists.

Aside from the dependence of the resulting value of the second derivative, the magnitude of the disturbance can also influence the robustness of the optimization algorithm. A disturbance of relatively large magnitude can result in convergence problems for the full Newton method that is used to obtain the flow solution for the disturbed geometry. Moreover, it should be noted that except for the hyper-dual method to compute the Hessian matrix, the entries of the matrix are subject to truncation error. Hence, the hyper-dual method will render a Hessian matrix that is accurate up to machine accuracy, while the other two methods will not.

V.A. Geometric design variables and constraints

The shape of the airfoil is parametrized using a fifth order (with rational basis functions of degree four) NURBS curve[18] with 11 control points, see Fig. 8. Note that the trailing edge is both the beginning and

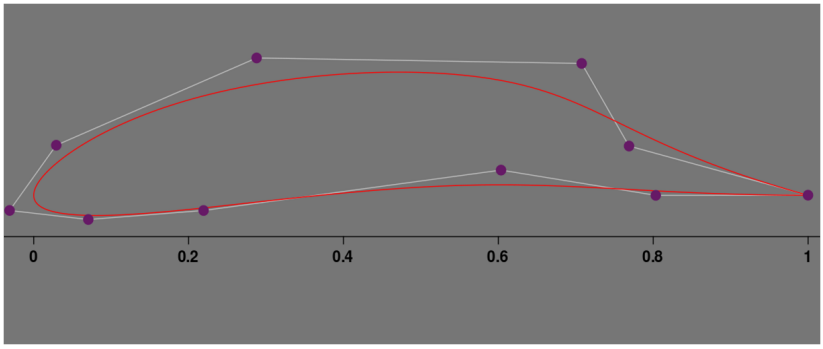


Figure 8. Parametrization of the airfoil via a NURBS curve.

end point of the curve, hence two control points are located at this position. As the location of the trailing edge is fixed at $(x, y) = (1, 0)$, only the position of the remaining 9 control points and their corresponding weights can be altered, leading to 27 design variables. Combined with the angle of attack, this results in a total of 28 design variables to be used in the optimization process.

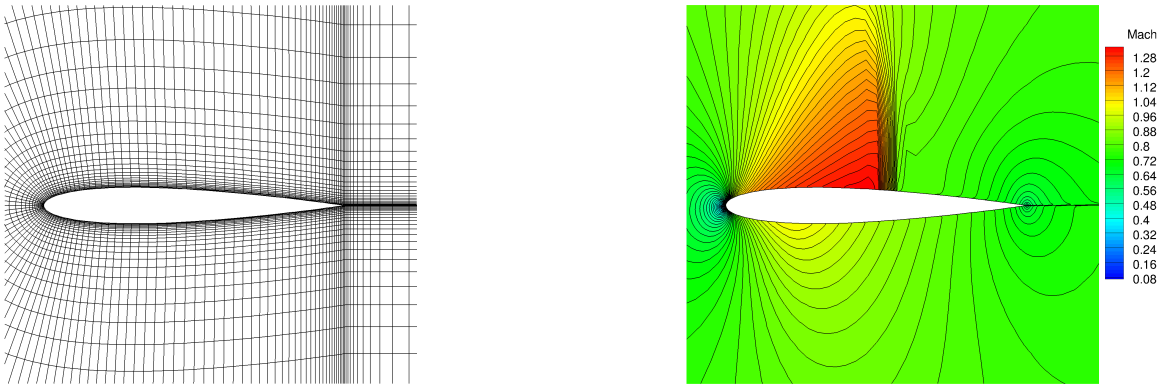
In order to obtain a realistic airfoil some geometric constraints must be imposed. These are:

- Location of the leading edge at $(x, y) = (0, 0)$
- Maximum curvature must be smaller than a user-prescribed value
- Maximum thickness must be larger than a user-prescribed value
- Trailing edge angle must be larger than a user-prescribed value
- Regularity constraints on the location of the control points

Note that this may not be the most efficient way of parametrizing the airfoil. Other methods may be more efficient in terms of the number of design variables required for the parametrization and the resulting number of design iterations that need to be performed before a converged result for the optimization procedure is obtained. Nevertheless, the parametrization method at hand provides a good means to investigate the hyper-dual number technique in an optimization method.

V.B. Results

The optimization process described above is applied to drag minimization at transonic conditions using the Euler equations. More specifically, the free stream Mach number is $M = 0.78$ and the initial geometry is



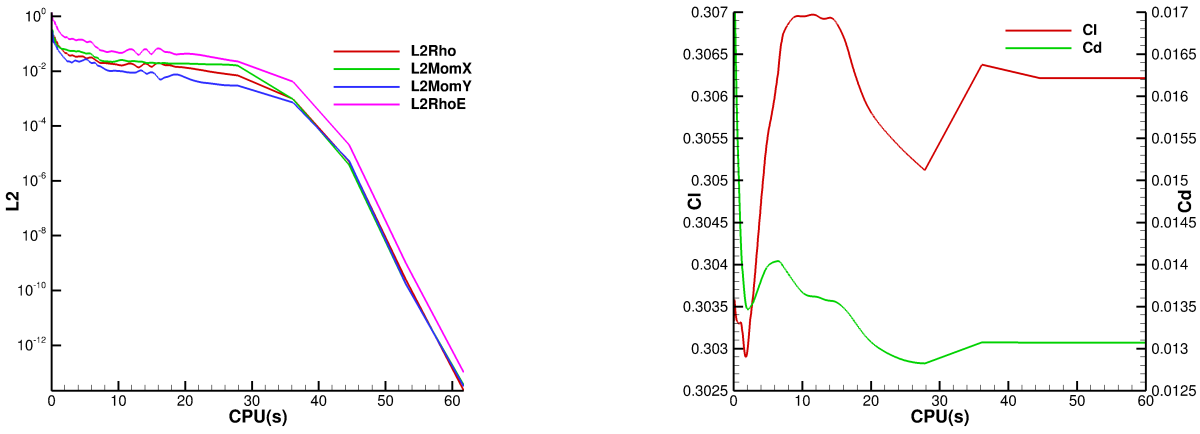
(a) Detail of the 192×48 computational grid.

(b) Mach number isolines.

Figure 9. NACA-0012, $M = 0.78$, $\alpha = 1.2^\circ$.

a NACA-0012 airfoil. The computational grid and flow field are shown in Figs. 9(a) and 9(b), respectively. The shock on the suction side is clearly visible, leading to a $c_d = 1.307 \cdot 10^{-2}$. The convergence of the flow solver (cell centered finite volume discretization, using Roe's approximate Riemann solver and MUSCL reconstruction via the Van Albada limiter) is shown in Figs. 10(a) and 10(b), which show the convergence of the L2 norms of the residuals and the force coefficients as a function of the CPU time, respectively. For the last couple of iterations a full Newton algorithm is used, for which the Jacobian matrix is obtained via the automatic differentiation tool Tapenade.[19] This is the same Jacobian matrix, which occurs in Eqs. (26), (27) and (29). As only one iteration is needed to solve these equations, the cost of obtaining a derivative is identical to the cost of one Newton iteration of the flow field. For this particular case it is even less, because a direct solver is used for which the LU decomposition is stored. In this case, the derivative information is obtained for a fraction of the cost of a Newton iteration. However, this is in general not the case when an iterative solver must be used.

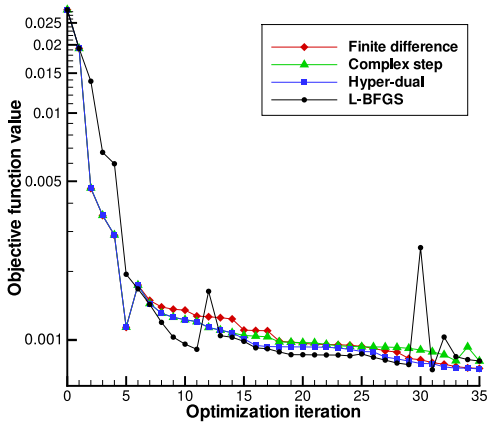
Starting from a NACA-0012, the profile is modified by the optimization algorithm, such that the c_d value is minimized, while the lift is fixed at $c_l = 0.5$. The result of the optimization is depicted in Fig. 11(b) by



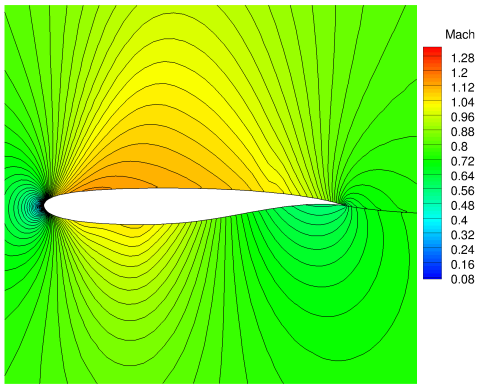
(a) Residuals.

(b) Force coefficients.

Figure 10. Convergence for the NACA-0012.



(a) Convergence of the c_d, for different method to determine the Hessian matrix.



(b) Mach number isolines of one of the optimized airfoils.

Figure 11. Results of IPOPT optimizations for $M = 0.78$.

means of a Mach number isoline contour plot, for which the same scale is used as in Fig. 9(b), so that a comparison can be made. It must be noted that the result shown in Fig. 11(b) is not unique. In Fig. 12 the result of using a different optimization software, SNOPT,[20] is shown. Although the geometries are different for both optimized airfoils, the shock has completely disappeared and the resulting drag is solely caused by the discretization error.

Figure 11(a) shows a comparison of the convergence of the optimization procedure for optimizations performed with IPOPT. The optimizations are performed on a C-grid of 192×48 vertices, similar to the one depicted in Fig. 9(a), where the latter dimension is the direction perpendicular to the airfoil. The graph contains four different curves, one for each method that is used to determine the Hessian matrix. These results show that similar convergence behavior is observed, although there are some slight differences. For the three methods that compute the entries of the Hessian matrix explicitly, the lines coincide for the first 6 optimization iterations. Then the curve for the finite difference method starts to deviate. After 14 design iterations the hyper-dual method and the complex-step method start to part ways. The reason for this different behavior is the slight differences in the Hessian matrix caused by the occurrence of truncation and cancellation errors.

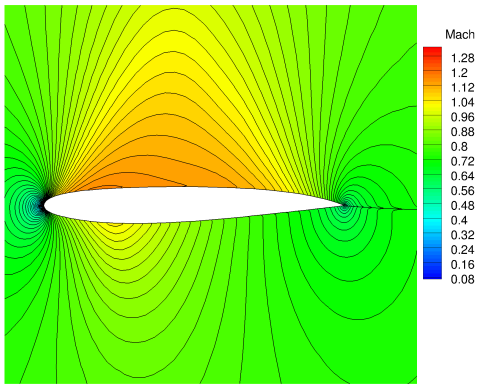


Figure 12. Other optimization result for $M = 0.78$, using the optimizer SNOPT.

When the convergence behavior of the L-BFGS method is considered, occasional erratic behavior is observed, while the methods using an 'exact' Hessian show much more smooth behavior; except for the increase in the objective function value at design iteration 6 for all three methods. Table 3 gives the duration of one optimization iteration, averaged over 100 iterations, normalized by the average duration of an IPOPT iteration which uses an approximation of the Hessian matrix. The results in Table 3 show that

Method of Hessian matrix computation	Normalized duration	Percentage of time for Hessian
L-BFGS approximation	1.00	-
Hyper-dual numbers	1.37	0.59
Finite difference approximation	1.18	0.44
Combination complex-step and finite-difference	1.95	1.20

Table 3. Timing results for different methods to determine the Hessian matrix. Second column gives the time of a design iteration relative to the one using a Hessian approximation; third column shows the time required to compute the Hessian matrix relative to one design iteration of the L-BFGS approximation.

an optimization iteration with IPOPT takes the least amount of time when an approximation of the Hessian matrix is used. This is because no entries of the Hessian matrix are computed explicitly, only the gradient is required.

For the other three cases where the Hessian matrix is computed explicitly, the finite-difference method appears to be the most efficient. The finite-difference method requires four flow solutions to determine the off-diagonal entries of the Hessian matrix and five for the terms on the diagonal of the matrix. Furthermore, each of these requires three Newton iterations to obtain a converged flow solution. On the other hand, the method using hyper-dual numbers to compute the Hessian matrix requires only one additional flow solution for each entry of the Hessian matrix. Moreover, for such a hyper-dual flow solution two Newton iterations suffice.

Nevertheless, the finite-difference method is faster, because of the increased number of mathematical operations that need to be performed when hyper-dual numbers are used. On the other hand, the complex-step method requires significantly more time than the other two methods to determine the Hessian matrix. This is because it requires three iterations to get a converged flow solution. Moreover, four flow solutions are required to evaluate the complex-step finite-difference stencil of Eq. (33). Furthermore, the application of complex arithmetic requires more mathematical operations, as with hyper-dual numbers, and therefore requires more time to complete than the real valued finite-difference method, which also requires (at least) four flow solutions to evaluate the stencil.

V.C. Discussion

The good correspondence of the convergence behavior for the first few design iterations depicted in Fig. 11(a) for the methods that explicitly compute the entries of the Hessian matrix indicates that the three methods presented can render a reasonably accurate representation of the Hessian matrix. However, all except for the hyper-dual number method require a considerable amount of tuning in order to achieve this. Moreover, without knowledge of the correct value of the second derivative, this tuning appeared virtually impossible.

The use of the 'exact' Hessian matrix results in a more favorable convergence behavior of the design iterations. When the costs are considered, the increase in computational time is quite reasonable. However, it should be noted that for the test case presented, the number of design variables is very limited. Moreover, the ability to reuse the LU decomposition of the Jacobian matrix to determine the flow solution for the disturbed geometry will not be an option for larger problems where the use of an iterative solution method for the linear system is inevitable. In that case the LU decomposition will not be available, which makes obtaining the flow solution much more costly.

Although the finite-difference method to determine the Hessian matrix is the cheapest, the hyper-dual number method is the preferred choice, because it does not need tuning in order to provide correct derivatives. Furthermore, it lacks truncation errors and therefore provides the exact Hessian, up to machine accuracy. Another advantage of the use of the hyper-dual number is that it does not influence the robustness of the optimization procedure, because the hyper-dual disturbance has no influence on the flow solution itself.

VI. Conclusions

It is possible to decrease the computational cost of forming the gradient and Hessian for some objective functions when using hyper-dual numbers. In order to see this benefit, care must be taken when modifying the objective function to operate on hyper-dual numbers. The naive approach is to modify every routine. However, to be computationally efficient, certain routines, such as iterative procedures, should not be modified. When the objective function involves an iterative procedure, an efficient strategy is to converge the procedure using real numbers, and then perform one iteration using hyper-dual numbers to compute the derivatives.

This approach has been demonstrated on two optimization problems using Newton's method. For the optimization of a supersonic business jet design, this approach decreases the computational cost by a factor of 8 over the naive approach. This reduces the cost of using hyper-dual numbers to below that of using finite differences. Making hyper-dual numbers both more accurate and less expensive to use.

The second application of this approach is to computational fluid dynamics codes, which at their hearts are iterative procedures. Initial testing indicated that starting from a converged solution did not provide any benefit. This initial testing was done using the same approximate Jacobian used for the flow solution. When this approach was applied to an Euler code with the *exact* Jacobian, the derivatives were seen to converge in one or two Newton iterations, leading to the conclusion that for an efficient use of hyper-dual numbers in CFD codes the usage of the exact Jacobian is a necessity. Transonic, airfoil shape optimization was performed using this 2D Euler code with Hessians calculated using hyper-dual numbers and two other less accurate approaches. The accuracy of the Hessian had little impact on the convergence of the optimization, however, the cost of using hyper-dual numbers was not unreasonable. Weighing the overall effort, hyper-dual numbers may be the preferred approach because the other approaches required considerable effort to find reasonable step sizes.

References

[1]Fike, J. A. and Alonso, J. J., "The Development of Hyper-Dual Numbers for Exact Second-Derivative Calculations," *AIAA paper 2011-886, 49th AIAA Aerospace Sciences Meeting*, 2011.

[2]Martins, J. R. R. A., Sturdza, P., and Alonso, J. J., "The complex-step derivative approximation," *ACM Trans. Math. Softw.*, Vol. 29, No. 3, 2003, pp. 245–262.

[3]Martins, J. R. R. A., Sturdza, P., and Alonso, J. J., "The Connection Between The Complex-Step Derivative Approximation And Algorithmic Differentiation," *AIAA paper 2001-0921, 39th Aerospace Sciences Meeting*, 2001.

[4]Clifford, W. K., "Preliminary Sketch of Biquaternions," *Proc. London Math. Soc.*, Vol. s1-4, No. 1, 1871, pp. 381–395.

[5]Hudson, R. W. H. T., "Review: Geometrie der Dynamen. Von E. Study," *The Mathematical Gazette*, Vol. 3, No. 44, 1904, pp. 15–16.

[6]Kantor, I. and Solodovnikov, A., *Hypercomplex Numbers: An Elementary Introduction to Algebras*, Springer-Verlag, New York, 1989.

[7]Deakin, M. A. B., "Functions of a Dual or Duo Variable," *Mathematics Magazine*, Vol. 39, No. 4, 1966, pp. 215–219.

[8]Eastham, M. S. P., "2968. On the Definition of Dual Numbers," *The Mathematical Gazette*, Vol. 45, No. 353, 1961, pp. 232–233.

[9]`http://adl.stanford.edu/`.

[10]Sobieszczanski-Sobieski, J., Altus, T. D., Phillips, M., and Sandusky, R., "Bi-level Integrated System Synthesis (BLISS) for Concurrent and Distributed Processing," *AIAA paper 2002-5409, 9th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Oprimization*, 2002.

[11]Carlson, H. W., "Simplified Sonic-Boom Prediction," NASA-TP-1122, March 1978.

[12]Thomas, C. L., "Extrapolation of Sonic Boom Pressure Signatures by the Waveform Parameter Method," NASA-TN-D-6832, June 1972.

[13]Pecnik, R., Terrapon, V. E., Ham, F., and Iaccarino, G., "Full system scramjet simulation," *Annual Research Briefs, Center for Turbulence Research, Stanford University*, 2009.

[14]Balay, S., Brown, J., , Buschelman, K., Eijkhout, V., Gropp, W. D., Kaushik, D., Knepley, M. G., McInnes, L. C., Smith, B. F., and Zhang, H., "PETSc Users Manual," Tech. Rep. ANL-95/11 - Revision 3.1, Argonne National Laboratory, 2010.

[15]Griewank, A. and Walther, A., *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2nd ed., 2008.

[16]Bartholomew-Biggs, M., "Using Forward Accumulation for Automatic Differentiation of Implicitly-Defined Functions," *Computational Optimization and Applications*, Vol. 9, 1998, pp. 65–84, 10.1023/A:1018382103801.

[17]Wachter, A. and Biegler, L. T., "On the Implementation of a Primal-Dual Interior Point Filter Line Search Algorithm for Large-Scale Nonlinear Programming," *Mathematical Programming*, Vol. 106, No. 1, 2006, pp. 25–57.

[18]Piegl, L. and Tiller, W., *The NURBS book (2nd ed.)*, Springer-Verlag New York, Inc., New York, NY, USA, 1997.

[19]Martinelli, M., Dervieux, A., Hascoët, L., Pascual, V., and Belme, A., "AD-based perturbation methods for uncertainties and errors," *Int. J. Engineering Systems Modelling and Simulation*, Vol. 2, No. 1/2, 2010, pp. 65–74.

[20]Gill, P. E., Murray, W., and Saunders, M. A., "SNOPT: An SQP Algorithm for Large-Scale Constrained Optimization," *Sigest article, SIAM Review*, Vol. 47, No. 1, 2005, pp. 99–131.