AIAA-2000-0689

AN AUTOMATED METHOD FOR SENSITIVITY ANALYSIS USING COMPLEX VARIABLES

Joaquim R. R. A. Martins * , Ilan M. Kroo † and Juan J. Alonso ‡
Department of Aeronautics and Astronautics
Stanford University, Stanford, CA

Abstract

The complex-step method for calculating sensitivities and its use in numerical algorithms is presented. A general procedure for the implementation of this method is described in detail and a script is developed that automates its implementation. The numerical examples include the automatic conversion of a structural finite element and a two-dimensional computational fluid dynamics code. In both of these examples, the complex-step method is compared with other existing methods, namely finite-differencing, automatic differentiation and an analytic method. The complex-step method is shown to have implementation advantages over automatic differentiation and computational advantages over finite-differencing.

Introduction

Sensitivity analysis has been an important area of engineering research, being particularly useful in design optimization. In choosing a method for computing sensitivities, one is mainly concerned with its accuracy and computational expense. In certain cases it is also important that the method be easily implemented.

One method that is very commonly used is finite-differencing. Although it is not known for being particularly accurate or computationally efficient, this method's biggest advantage resides in the fact that it is extremely easy to implement.

Analytic methods, on the other hand, are typically much more accurate and efficient but they require the derivation and development of a program that is specific to each case.

Yet another technique, called Automatic Differentiation for Fortran (ADIFOR),[1] uses a script that automatically differentiates an existing code by creating a new one that calculates the required sensitivities. This approach has the advantages of the first two methods mentioned above: being both accurate and relatively easy to implement.

The use of complex variables to develop estimates of derivatives originated with the work of Lyness and Moler[2] and Lyness.[3] Their papers introduced several methods that made use of complex variables, including a reliable method for calculating the n^{th} derivative of an analytic function. However, only recently has some of this theory been rediscovered by Squire and Trapp[4] and used to obtain a very simple expression for estimating the first derivative. This estimate is suitable for use in modern numerical computing and has shown to be very accurate, extremely robust and surprisingly easy to implement, while retaining a reasonable computational cost. The potential of this technique is now starting to be recognized and it has been used in CFD sensitivity analysis by Anderson[5] and in a MDO environment by Newman.[6]

The objective of this paper is to shed new light on the theory behind the complex-step derivative approximation, demonstrate its advantages in computing sensitivities and provide a script that can implement it automatically. There will also be a comparison with other methods, with emphasis on finite-differencing and ADIFOR, since these are also in the class of methods that have a relatively straightforward implementation.

*Graduate Student, AIAA Student Member
†Professor, AIAA Associate Fellow
‡Assistant Professor, AIAA Member

Theory

Analyticity

The use of complex variables in the evaluation of function sensitivities is sometimes better understood if one makes the following analogy by Saff and Snider.[7] Consider the set of rational numbers and the fact that no rational number can solve $x^2 = 2$. If we want to extend the set of rational numbers so that are able to solve this equation we can write:

$$w = x + \sqrt{2}y \tag{1}$$

where x and y are rational numbers and a number of this form can now solve $x^2 = 2$.

Now suppose that we already have the set of real numbers. With real numbers alone, we still cannot solve $x^2 = -1$. If we want to solve this equation, we can adopt the same approach as before and extend the set of real numbers by defining a set of new numbers,

$$z = x + iy \tag{2}$$

where x and y are real and $i = \sqrt{-1}$. This defines the set of complex numbers which not only enables us to solve $x^2 = -1$, but in fact any polynomial of degree n.

Now lets consider to our extension of the set of rational numbers represented by w. If a function $f(w)$ is differentiable, its derivative must be the same, whether we differentiate it with respect to the rational part or the irrational one, i.e.,

$$\frac{\partial f}{\partial w} = \frac{\partial f}{\partial x} = \sqrt{2}\frac{\partial f}{\partial y} \tag{3}$$

The same is true for a function of a complex variable that is differentiable in the complex plane and similarly we can write,

$$\frac{\partial f}{\partial y} = i\frac{\partial f}{\partial x} \tag{4}$$

The real and imaginary parts of the function can be separated and we can define,

$$f = u + iv. \tag{5}$$

Comparing the real and imaginary parts of Equation(4) we obtain the familiar Cauchy-Riemann equations,

$$\frac{\partial u}{\partial x} = \frac{\partial v}{\partial y} \tag{6}$$

$$\frac{\partial u}{\partial y} = -\frac{\partial v}{\partial x}. \tag{7}$$

A complex function that satisfies these equations is said to be *analytic*, which is to say that it is differentiable in the complex plane.

The main goal of this analogy between complex numbers and an extension of rational numbers is to remind ourselves that a complex number is really just *one number*. It is not unusual to forget this and think instead of a complex number as two numbers just because we do not have a way of representing it with a single axis or by a single term. This is an important realization that will be useful in explaining some of the implementation issues that are described later.

First Derivative Approximations

Finite-differencing formulas are a common method for estimating derivatives. These formulas can be derived by truncating a Taylor series which has been expanded about a given point x. A common estimate for the first derivative is the forward-difference formula,

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}, \tag{8}$$

where h is finite-difference interval. The truncation error is $O(h)$, and therefore it is a first-order approximation. For a second-order estimate the we can use the central-difference formula,

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}. \tag{9}$$

As with any divided-difference approximation, we are faced in this case with the "step size dilemma", i.e. wanting to choose a small step size that minimizes truncation error while avoiding the use of a step size so small that subtractive cancellation errors become inevitable. Note that this approximation is a discretization of the mathematical definition of the first derivative.

We will now see that an equally simple first derivative estimate for real functions can be obtained using complex calculus. If f is an analytic function, then the Cauchy-Riemann equations apply, establishing the exact relationship between the real and imaginary parts of the function. We can use the definition of a derivative in the right hand side of the first Cauchy-Riemann Equation(6) to obtain,

$$\frac{\partial u}{\partial x} = \lim_{h \to 0} \frac{v(x + i(y+h)) - v(x+iy)}{h}. \tag{10}$$

Since the functions that we are interested in are real functions of a real variable, we restrict ourselves to the real axis and then we know that $y = 0$, $u(x) = f(x)$ and $v(x) = 0$. Equation (10) can then be rewritten as,

$$\frac{\partial f}{\partial x} = \lim_{h \to 0} \frac{\mathrm{Im}\left[f\left(x + ih\right)\right]}{h}. \tag{11}$$

For a small discrete h, this can be approximated by,

$$\frac{\partial f}{\partial x} \approx \frac{\mathrm{Im}\left[f\left(x + ih\right)\right]}{h}, \tag{12}$$

which we will call the *complex-step derivative approximation*. This estimate is not subject to subtractive cancellation error, since it does not involve a difference operation. This constitutes a tremendous advantage over the finite-difference approach expressed in Equation (8).

The property that was used to derive this approximation is a direct consequence of the analyticity of the function and it is therefore necessary that the function f be analytic in the complex plane. In a later section we will discuss to what extent a generic numerical algorithm can be considered to be an analytic function.

In order to determine the error involved in this approximation, we repeat the derivation by Squire and Trapp[4] which is based on a Taylor series expansion. Rather than using a real step h, we use a pure imaginary step, ih. If f is a real function in real variables and it is also analytic, we can expand it in a Taylor series about a real point x as follows,

$$f(x + ih) = f(x) + ihf'(x) - h^2\frac{f''(x)}{2!} - ih^3\frac{f'''(x)}{3!} + \ldots \tag{13}$$

Taking the imaginary parts of both sides of Equation (13) and dividing the equation by h yields

$$f'(x) = \frac{\mathrm{Im}\left[f(x + ih)\right]}{h} + h^2\frac{f'''(x)}{3!} + \ldots \tag{14}$$

Hence the approximations is a $O(h^2)$ estimate of the derivative of f. The reason we achieve a second-order approximation with just one function evaluation has to do with the fact that the imaginary part of the result is an odd function with respect to the imaginary part of the independent variable.

Numerical Example

Because the complex-step approximation does not involve a difference operation, we can choose extremely small steps sizes with no loss of accuracy due to subtractive cancellation.

To illustrate this, consider the following analytic function:

$$f(x) = \frac{e^x}{\sqrt{sin^3 x + cos^3 x}} \tag{15}$$

The exact derivative at $x = 1.5$ was computed analytically to 16 digits and then compared to the results given by the complex-step (12) and the forward and central finite-difference formulas (8,9).

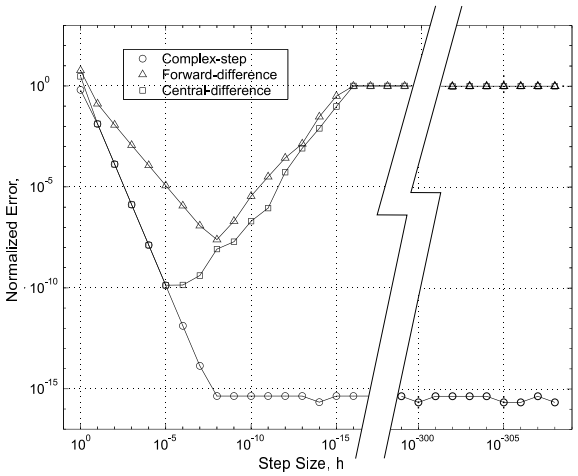

Figure 1: Normalized error in the sensitivity estimates given by finite-difference and the complex-step with the analytic result as the reference; $\varepsilon = \frac{\left|f' - f'_{ref}\right|}{\left|f'_{ref}\right|}$.

The forward-difference estimate initially converges to the exact result at a linear rate since its truncation error is $O(h)$, and the central-difference converges quadratically, as expected. However, as the step is reduced to a value of about 10^{-8} for the forward-difference and 10^{-5} for the central-difference, subtractive cancellation errors become an issue and the estimates become unreliable. When the interval h is so small that no difference exists in the output (for steps smaller than 10^{-16}) the finite-difference estimates eventually yields zero and then $\varepsilon = 1$.

The complex-step estimate converges quadratically with decreasing step size, as predicted by the truncation error estimate. The estimate is practically insensitive to small step sizes and below an h of the order of 10^{-8} it achieves the accuracy of the function evaluation.

Comparing the best accuracy of each of these approaches, we see that by using finite-difference we only achieve a fraction of the accuracy that is obtained by using the complex-step approximation.

As we can see the complex-step size can be made extremely small. However, there is a lower limit on the step size when using finite precision arithmetic. The range of real numbers that can be handled in numerical computing is dependent on the particular compiler that is used. Usually, however, when double-precision complex numbers are used, the smallest non-zero number that can be represented is 10^{-308}. If a number falls below this value, underflow occurs and the number drops to zero. Note that the estimate is still accurate down to a step of the order of 10^{-307}. Below this, underflow occurs and the estimate results in `NaN`. Therefore, the smallest possible h is the one below which underflow occurs somewhere in the algorithm.

When it comes to comparing the relative accuracy of complex and real computations, analysis shows that there is an increased error in basic arithmetic operations when using complex numbers, more specifically when dividing and multiplying.[8]

Higher Derivative Approximations

The derivative of order n of a given analytic function can be calculated by Cauchy's Integral Formula in its general form:[7]

$$f^{(n)}(z) = \frac{n!}{2\pi i} \int_{\Gamma} \frac{f(\xi)}{(\xi - z)^{n+1}} d\xi, \qquad (16)$$

where Γ is a simple closed positively oriented contour that encloses z. This integral can be numerically computed by means of a mid-point trapezoidal rule approximation around a circle of radius r yielding,[9]

$$f^{(n)}(z) \approx \frac{n!}{mr} \sum_{j=0}^{m-1} \frac{f\left(z + re^{i\frac{2\pi j}{m}}\right)}{e^{i\frac{2\pi jn}{m}}}, \qquad (17)$$

where if m is the number of points used in the integration, we can approximate a derivative of order $n = 0, 1, \ldots, m-1$.

When comparing conventional finite-difference methods and the complex integration expressed in Equation (17), we observe that both use formulas of the type $\sum a_i f(x_i)$ where the coefficients have different signs. However, there is a significant difference between the two. In conventional methods the step h has to be decreased in order to reduce the truncation error of the approximation, making it susceptible to subtractive cancellation. If we want to reduce the truncation error of the complex integration method all we need to do is to increase the number of function evaluations, i.e. m in Equation (17). This keeps the subtractive cancellation error constant and it is then possible to calculate a bound on the error involved in the approximation.[2]

The complex-step first derivative approximation (12) can be derived from Equation (17). From complex variable theory, for a real function of the real variable that is analytic,

$$f(x + iy) = u + iv \Rightarrow f(x - iy) = u - iv. \qquad (18)$$

By setting $m = 2$ in Equation (17) and starting the integration at the top of the circle ($z + re^{i\frac{\pi}{2}}$) rather than on the left side ($z + r$) we obtain the second-order approximation that we arrived at previously. This is the only approximation that can be obtained from Equation (17) that does not involve subtraction and it is only valid for functions whose imaginary part is zero on the real axis.

Implementation

Intrinsic Complex Functions and Operators in Fortran

In the derivation of the complex-step derivative approximation (12) for a function f we have assumed that f was an analytic function, i.e. that the Cauchy-Riemann equations apply. It is therefore important to examine to what extent this assumption holds when the value of the function is calculated by a numerical algorithm. In addition it is also useful to explain how we can convert real functions and operators such that they can take complex numbers as arguments. Fortunately, in the case of Fortran, complex numbers are a standard data type and many intrinsic functions are already defined for them.

Any algorithm can be broken down into a sequence of basic operations. Two main types of operations are relevant when converting a real algorithm to a complex one:

- Relational operators
- Arithmetic functions and operators.

Relational logic operators such as "greater than" and "less than" are not defined for complex numbers in Fortran. These operators are usually used in conjunction with `if` statements in order to redirect the execution thread. The original algorithm and its "complexified" version must obviously follow the same execution thread. Therefore, defining these operators to compare only the real parts of the arguments is the correct approach.

Functions that choose one argument such as `max` and `min` are based on relational operators. Therefore, according to our previous discussion, we should once more choose a number based on its real part alone and let the imaginary part "tag along".

Any algorithm that uses conditional statements is likely to be a discontinuous function of its inputs. Either the function value itself is discontinuous or the discontinuity is in the first or higher derivatives. When using a finite-difference method, the derivative estimate will be incorrect if the two function evaluations are within h of the discontinuity location. However, if the complex-step is used, the resulting derivative estimate will be correct right up to the discontinuity. At the discontinuity, a derivative does not exist by definition, but if the function is defined a that point, the approximation will still return a value that will depend on how the function is defined at that point.

Arithmetic functions and operators include addition, multiplication, and trigonometric functions, to name only a few, and most of these have a standard complex definition that is analytic almost everywhere. Many of these definitions are implemented in Fortran. Whether they are or not depends on the compiler and libraries that are used. The user should check the documentation of the particular Fortran compiler being used in order to determine which intrinsic functions need to be redefined. A full description of Fortran's standard intrinsic functions and their complex definitions can be found in Table 5.

Functions of the complex variable are merely extensions of their real counterparts. By requiring that the extended function satisfy the Cauchy-Riemann equations, i.e. analyticity, and that its properties be the same as those of the real function, we can obtain a unique complex function definition.[7] Since these complex functions are analytic, the complex-step approximation is valid and will yield the correct result.

Some of the functions, however, have singularities or branch cuts on which they are not analytic. This does not pose a problem since, as previously observed, the complex-step approximation will return a correct one-sided derivative. As for the case of a function that is not defined at a given point, the algorithm will not return a function value, so a derivative cannot be obtained. However, the derivative estimate will be correct in the neighborhood of the discontinuity.

The only standard complex function definition that is non-analytic is the absolute value function or modulus. When the argument of this function is a complex value, the function returns a positive real number, $|z| = \sqrt{x^2 + y^2}$. This function's definition was not derived by imposing analyticity and therefore it will not yield the correct derivative when using the complex-step estimate. In order to derive an analytic definition of `abs` we start by satisfying the Cauchy-Riemann equations. From the Equation (6), since we know what the value of the derivative must be, we can write,

$$\frac{\partial u}{\partial x} = \frac{\partial v}{\partial y} = \begin{cases} -1 & \Leftarrow x < 0 \\ +1 & \Leftarrow x > 0 \end{cases}. \tag{19}$$

From Equation (7), since $\partial v/\partial x = 0$ on the real axis, we get that $\partial u/\partial y = 0$ on the axis, so the real part of the result must be independent of the imaginary part of the variable. Therefore, the new sign of the imaginary part depends only on the sign of the real part of the complex number, and an analytic "absolute value" function can be defined as:

$$\texttt{abs}(x + iy) = \begin{cases} -x - iy & \Leftarrow x < 0 \\ +x + iy & \Leftarrow x > 0 \end{cases}. \tag{20}$$

Note that this is not analytic at $x = 0$ since a derivative does not exist for the real absolute value. Once again, the complex-step approximation will give the correct value of the first derivative right up to the discontinuity. Later the $x > 0$ condition will be substituted by $x \geq 0$ so that we not only obtain a function value for $x = 0$, but also we are able to calculate the correct right-hand-side derivative at that point.

Automatic Implementation

In order to implement the complex-step method, one must have access to the source code that computes the value of f. The implementation procedure can be summarized as follows:

- Substitute all `real` type variable declarations with `complex` declarations. It is not strictly necessary to declare *all* variables complex, but it is much easier to do so.
- Define all functions and operators that are not defined for complex arguments.
- A complex-step can then be added to the desired x and $\frac{\partial f}{\partial x}$ can be estimated using Equation (12).

The method has been successfully implemented by hand on a Fortran three-dimensional CFD code.[5,6] This was done by writing a set of subroutines with a different name from the intrinsic functions that needed to be defined for complex arguments. Then the code was processed line by line, and some of the intrinsic function names where substituted with the new function names. Also, in some `if` statements the complex arguments in the comparisons must be cast to real. This procedure requires an advanced knowledge of the type of arguments, which is not always a trivial matter.

Fortunately, in Fortran 90, intrinsic functions and operators (including comparison operators) can be overloaded. This means that if a particular function or operator does not take complex arguments, one can extend it by writing another definition that takes this type of arguments. This feature makes it much easier to implement the complex-step method since once we overload the functions and operators, there is no need to change the function calls or conditional statements. The compiler will automatically determine the argument type and choose the correct function or operation.

All the functions that need to be overloaded and their new definitions are listed in Table 5. Adding the extended function definitions is simple, since it has been written as a Fortran 90 module that can be used by any subroutine in a program.

In order to automate the implementation, a script that processes Fortran source files automatically was developed. The script declares the complex functions module in every existing subroutine, substitutes all the real type declarations by complex ones and adds `implicit complex` statements when appropriate. The script was written in Python[10,11] and uses the Perl regular expressions module. The latest versions of both the script and the Fortran 90 module, are available from the first author.[12]

The third step of the implementation, adding the complex-step to the variable of interest and using the complex-step approximation, is left to the user. Another task that must be done manually is the change of file I/O statements, since the user might want to have I/O files containing either full complex numbers or just their real parts.

Implementation in Other Programming Languages

C/C++: Neither C nor C++ perform complex arithmetic by default, although there are complex arithmetic libraries than one can use. In C++, all operators and functions can be overloaded just as in Fortran 90, so automatic implementation seems to be perfectly feasible.

Matlab: As in the case of Fortran, one must redefine functions such as `abs`, `max` and `min`. All differentiable functions are defined for complex variables. Results for the simple example in the previous section were computed using Matlab. The standard transpose operation represented by an apostrophe (`'`) poses a problem as it takes the complex conjugate of the elements of the matrix, so one should use the non-conjugate transpose represented by "dot apostrophe" (`.'`) instead.

Java: Complex arithmetic is not standardized at the moment but there are plans for its implementation. Although function overloading is possible, operator overloading is currently not supported.

Python: When using the Numerical Python module (NumPy), we have access to complex number arithmetic and implementation should be as straightforward as in Fortran.

Results

The complex-step method was implemented automatically in two different analysis codes in order to calculate sensitivities. The first one is a structural finite element solver and the second one is a two-dimensional CFD code. The calculated sensitivities can then be used to perform design optimization in a multidisciplinary environment.

Structural Sensitivities

For purposes of comparison of the complex-step approximation with other sensitivity analysis methods, we used a structural solver based on a finite element code, `fesmeh`, developed by Holden.[13] The sensitivities are calculated in order to perform design optimization in a multidisciplinary environment and the complex-step method was automatically.

In this example, a transonic transport wing is modeled with multiple spars, shear webs, and ribs located at various spanwise stations, together with the skins of the upper and lower surfaces of the wing box.

Two types of finite elements are used: truss and triangular plane-stress plate elements. Both element types have 3 translational degrees of freedom per

node, so the truss has a total of 6 degrees of freedom while the plate has 9 degrees of freedom. Figure 2 shows an expanded view of the finite element model.

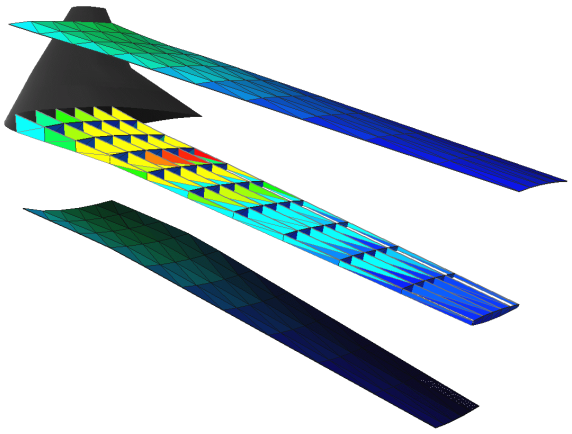

Figure 2: Structural finite element model of wing

In a first study, the sensitivity estimates given by the complex-step and forward-finite-difference methods are compared for a varying step size. The sample sensitivity chosen for this study was the derivative of the stress in a spar cap with respect to its own cross-sectional area. This kind of derivative is very important in structural optimization, where stress constraints are usually required.

Note that in this case, stress is a non-linear function of the cross-sectional area and, therefore, we should be able to observe the rate convergence of the estimates for decreasing step sizes.

Figure 3 shows a comparison of results – analogous to that of Figure 1 – with a reference derivative which is obtained by an analytic method. The analytic method used here is the direct method applied to a discrete linear set of equations as described by Adelman.[14] This method is included here only to provide a benchmark, since it has an implementation that is far more involved and code-specific than the other ones.

As expected, the error of the finite-difference estimate initially decreases at a linear rate. As the step is reduced to a value of about 10^{-6}, subtractive cancellation errors become increasingly significant and the estimate error increases. For even smaller perturbations – of the order of 10^{-17} – no difference exists in the output and the finite-difference estimate eventually goes to zero ($\varepsilon = 1$).

The complex-step estimate converges quadratically with decreasing step size, converging to the code's precision when h is of the order of 10^{-7}. Note

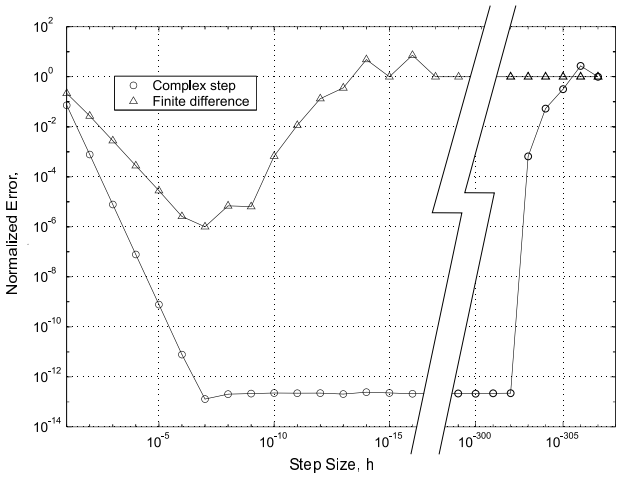

Figure 3: Error of sensitivity estimates given by finite-difference and complex-step with the analytic method result as the reference; $\varepsilon = \frac{|f' - f'_{ref}|}{|f'_{ref}|}$.

that the estimate is still accurate down to a step of the order of 10^{-306}. Below this, underflow starts to occur, the estimate is corrupted, and eventually the result becomes meaningless.

A second study compares the accuracy and computational cost between the three methods mentioned above and an additional method: ADIFOR.[1] This is a package that automatically processes a given Fortran program, producing a new program that in addition to the original computations also calculates the desired sensitivities. A sample of the sensitivity results is shown in Table 1, and a computational comparison is made in Table 2. The cost values are normalized with respect to the computation time and memory usage of the complex-step method.

The computations were performed on a SGI Octane and correspond to the calculation of the sensitivities of the stress in all of the 60 trusses with respect to their cross-sectional areas, i.e., a total of 3600 sensitivities. The sample sensitivity is the same one that was used to produce the results shown previously in Figure 3. When compiled using double precision, the finite element solver has an accuracy of about 13 digits, so the last 4 digits should be ignored.

The finite-difference sensitivity estimate – shown at the bottom of Tables 1 and 2 – is obtained using an optimal step ($h = 10^{-7}$). Even then, the estimate is shown to be only half as accurate as the other ones. Although this method is extremely easy to implement, finding a step that gives reasonably

Method	Sample Sensitivity
Complex	$-39.04976004580\mathit{4646}$
ADIFOR	$-39.04976004580\mathit{9059}$
Analytic	$-39.04976004580\mathit{5281}$
FD	$-39.0497\mathit{24352820375}$

Table 1: Sensitivity estimate accuracy comparison

Method	Time	Memory
Complex	1.00	1.00
ADIFOR	2.33	8.09
Analytic	0.58	2.42
FD	0.88	0.72

Table 2: Relative computational cost comparison for the calculation of the complete Jacobian

accurate estimates is usually a problem and, therefore, the total computation time in practice is much higher than the one that is shown in these results.

The analytic method was accurate, as expected, and by far the fastest. The computation using this method required considerably more memory since a host of new variables were introduced in the algorithm. As mentioned before, its implementation is much more involved than in the other cases and this places this method in a class of its own.

ADIFOR produced very accurate estimates but it was the costliest, with respect to both computation time and memory usage. This has to do with the fact that that ADIFOR produces a code which is is much larger than the original one which contains many more statements and variables. This fact constitutes an implementation disadvantage as it becomes impractical to debug this new code. One has to work with the original source and every time it is changed (or when we want to compute different derivatives) one must first run ADIFOR and then compile the new version.

The complex-step method was also accurate to the code's precision, was reasonably fast, and used less memory than any other method with the exception of the finite-difference method. The results were obtained using $h = 10^{-100}$. In general, the memory requirement will always be greater than in the case of finite-differencing, but never more than twice as much. As opposed to ADIFOR, the new "complexified" code is practically identical to the original one and can therefore be worked on directly. With the help of a few compiler flags one can even produce a single code that can be chosen to be real or complex at compilation.

Finally, note that the relative cost values given in Table 2 may vary for different problems, since they depend heavily on the ratio of the number of outputs we want to differentiate to the number of design variables we want to differentiate with respect to. However, the costs associated with the complex-step method will always be proportional to those of the finite-difference method.

Aerodynamic Sensitivities

To further validate the complex-step method and test the automatic implementation process on a CFD code, we chose to "complexify" FLO82. This is a two-dimensional, cell-centered, finite volume solver for the complete Euler equations. It achieves fast turnaround through the use of several techniques for convergence acceleration, namely multigrid, implicit residual smoothing, enthalpy damping, and local time-stepping. The solution is updated explicitly at every multigrid iteration using a modified Runge-Kutta algorithm which treats the convective and dissipative portions of the residual separately for optimum error damping properties. FLO82 has multiple options for artificial dissipation schemes, ranging from the baseline Jameson-Schmidt-Turkel (JST) algorithm to advanced schemes such as CUSP, ECUSP, and HCUSP.[15] FLO82 was chosen as an example for this work, since it contains the essential numerical schemes present in more complicated programs such as FLO87 and FLO107 that solve three-dimensional Euler and Navier-Stokes flows, and are currently used in our design work.

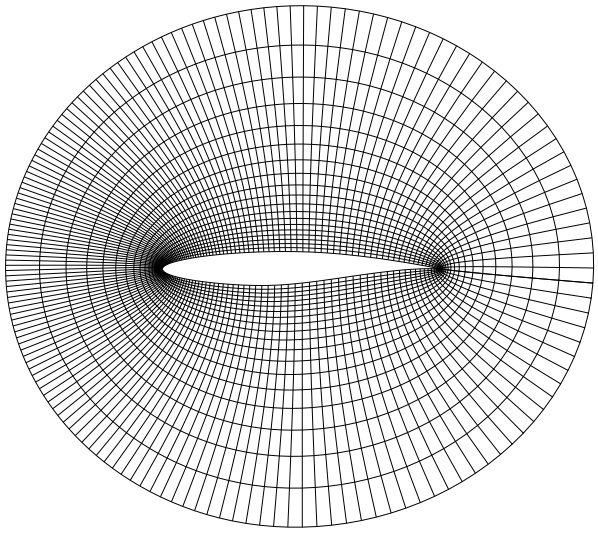

Figure 4: RAE 2822 airfoil and grid geometry.

The implementation process, even in this relatively large program, was carried out very quickly with the use of the "complexifying" script. After less than one hour, the new solver was calculating the correct sensitivities.

The test case was chosen to be the RAE 2822 airfoil shown in Figure 4 at a an angle-of-attack of 3 degrees with a freestream Mach number of 0.7. The resultant pressure coefficient distribution is plotted in Figure 5.

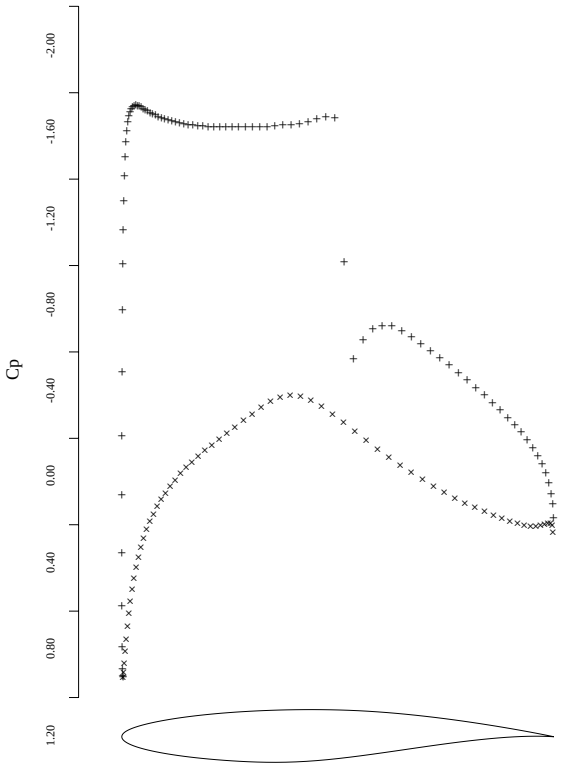

Figure 5: Pressure coefficient distribution, Mach = 0.7, $\alpha = 3.0^{\circ}$.

As we have seen in the previous numerical examples, it is rather useful to study the variation of the sensitivity estimates with step size. For this study, we computed the derivative of drag coefficient with respect to the free stream Mach number, i.e. $\partial C_D/\partial M_\infty$. The results are shown in Figure 6, and the reference estimate used to calculate the error was, in this case, the last estimate calculated by the complex-step method, corresponding to $h = 10^{-38}$. The behavior of the estimates follows once more, the same general pattern: both estimates converge initially, the forward-finite-difference estimate then rebounds to zero, and the complex-step estimate maintains the accuracy of the code until underflow takes place. Again, note that the range of h for which the finite-difference gives a reasonable estimate is very small, and therefore a convergence study would be necessary if we were to use this method in design optimization. Even then, there would be no guarantee that the optimal step would remain the same throughout the design process.

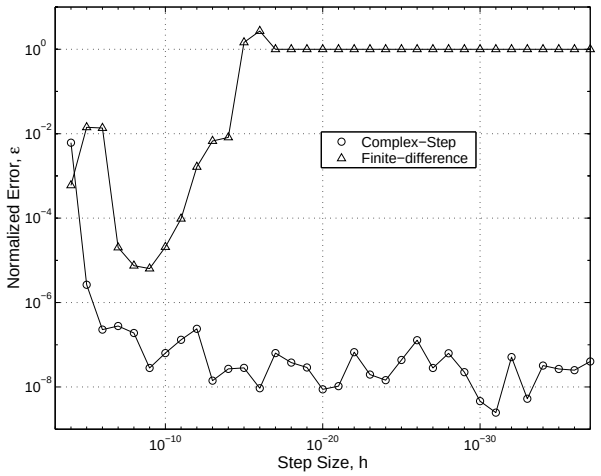

Figure 6: Normalized error of sensitivity estimates of $\partial C_D/\partial M_\infty$ The reference value is the last complex-step estimate; $\varepsilon = \frac{|f' - f'_{ref}|}{|f'_{ref}|}$.

In Table 3 we list the sensitivity matrix of the lift, drag and moment coefficients with respect to angle-of-attack and freestream Mach number. For the complex-step, we used an arbitrary small step size of 10^{-20}, while for the finite-difference, the optimum step sizes were found to be 10^{-8} for α and 10^{-6} for M_∞. The digits that are shown in italics in the table are the ones that have not converged.

Since the solver is iterative, we also analyzed the convergence of the sensitivity estimates and compared it to the convergence of the algorithm. Figure 7 shows the evolution of the sensitivity residual for both the finite-difference and the complex-step estimates of $\partial C_D/\partial M_\infty$ during the process of a typical calculation lasting 150 iterations. The sensitivity residual is defined as the absolute value of the difference between the current estimate of $\partial C_D/\partial M_\infty$ and its value at the previous iteration. To measure the convergence of the algorithm, we used the average residual of the density equation. As expected, the rate of convergence of the sensitivity estimate of the finite-difference method tracks that of the flow solver. The fact that the rate of convergence of the

Sensitivity	Complex	FD
$\partial C_L/\partial\alpha$	12.37054*751691092*	12.370*726665267277*
$\partial C_D/\partial\alpha$	0.8602380*269441042*	0.8602*4234610682058*
$\partial C_M/\partial\alpha$	−0.5026301*652982372*	−0.502631*3670830616*
$\partial C_L/\partial M_\infty$	3.249972*985150532*	3.24994*47577549727*
$\partial C_D/\partial M_\infty$	0.439786*71505102005*	0.43978*998888818932*
$\partial C_M/\partial M_\infty$	−0.990373*88394690016*	−0.99037*47405504145*

Table 3: Lift, drag and moment coefficient sensitivities with respect to angle-of-attack and Mach number.

complex-step estimate is the same as the one for the other two is rather reassuring.

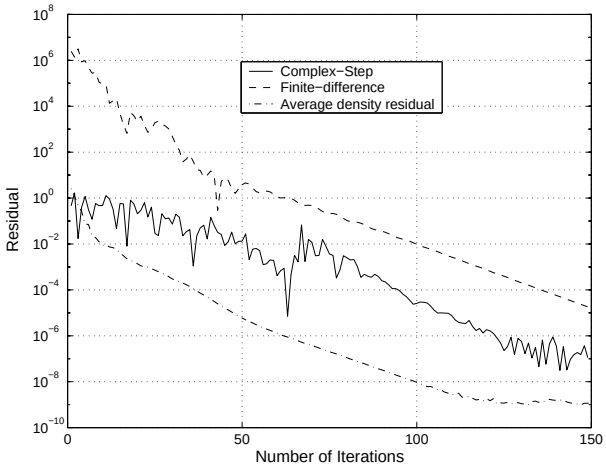

Figure 7: Residual for the estimates of $\partial C_D/\partial M_\infty$ compared to the convergence of the code.

The last set of results presents the calculation of the sensitivity of the drag coefficient with respect to a series of airfoil shape parameters. The use of these sensitivities is commonplace in aerodynamic shape optimization problems. The shape parameters that are used in this study are the amplitudes of a series of Hicks-Henne "bump" functions that are centered at various points along the upper surface of the airfoil, from the leading edge to the 90% chord location. Hicks-Henne "bump" function have been used in previous work[16] and have the desirable property that, when applied to a smooth baseline geometry, they produce a smooth surface. The plot in Figure 8 shows the sensitivities at 55 grid points distributed along the top surface of the airfoil. The values of the sensitivities are interpolated using cubic splines. The complex-step size was set to $h = 10^{-10}$ and for the finite-difference step the optimal value was found to be $h = 10^{-9}$. The plot shows no discernible difference bettween the two sets of results and the maximum difference between the two was calculated

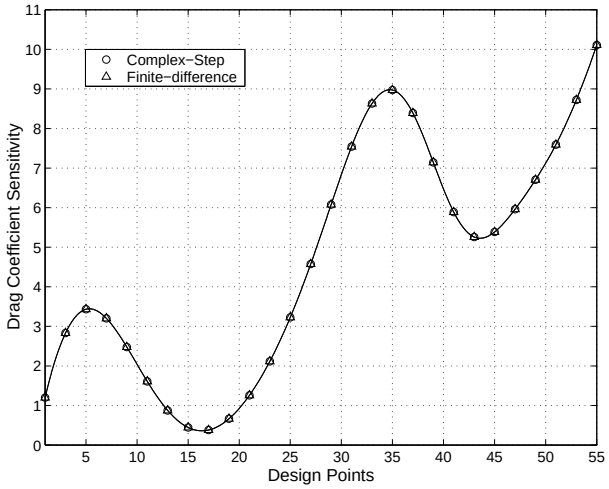

Figure 8: Comparison of the estimates for the shape sensitivities of the drag coeffiecient, $\partial C_D/\partial b_i$.

Method	Time	Memory
Complex	1.00	1.00
FD	0.31	0.55

Table 4: Relative computational cost comparison for the calculation of the complete shape sensitivity vector.

to be 2.1×10^{-4}.

A comparison of the relative computational cost of the two methods was also made for the aerodynamic sensitivities, namely for the calculation of the complete shape sensitivity vector. Table 4 lists these costs, normalized with respect to the complex-step results.

The complex-step exhibits, once more, a higher cost than the finite-difference method. However, in this case the difference is significantly higher than the one observed for the structural solver. In spite of this substantial difference, it is still usually advantageous to use the complex-step method, since no additional computational effort is necessary in

order to find an adequate step.

Conclusions

The theory behind the application of the complex-step method to real-world numerical algorithms was introduced. The Cauchy-Riemann equations and the fact that a complex number is really just *one number*, were vital in the rationalization of the practical implementation of this method.

The implementation process of the complex-step derivative approximation was successfully automated for two large solvers, including an iterative one. The resulting derivative estimates were validated by comparing them to results obtained by other known methods.

The complex-step method, unlike the finite-difference method, has the advantage of being step size insensitive and for small enough steps, the accuracy of the sensitivity estimates is only limited by the numerical precision of the algorithm.

This method is now at least as easy to implement as ADIFOR and once it is implemented it is much easier to maintain the resulting code.

Acknowledgements

The authors would like to thank Professor James Lyness for his invaluable help and support.

References

[1] Bischof, C., A. Carle, G. Corliss, A. Grienwank, P. Hoveland, "ADIFOR: Generating Derivative Codes from Fortran Programs", *Scientific Programming*, Vol. 1, No. 1, 1992, pp. 11-29.

[2] Lyness, J. N., and C. B. Moler,, "Numerical differentiation of analytic functions", *SIAM J. Numer. Anal.*, Vol. 4, 1967, pp. 202-210.

[3] Lyness, J. N., "Numerical algorithms based on the theory of complex variables", *Proc. ACM 22nd Nat. Conf.*, Thompson Book Co., Washington DC, 1967, pp. 124-134.

[4] Squire, W., and G. Trapp, "Using Complex Variables to Estimate Derivatives of Real Functions", *SIAM Review*, Vol. 10, No. 1, March 1998, pp. 100-112.

[5] Anderson, W. K., J. C. Newman, D. L. Whitfield, E. J. Nielsen, "Sensitivity Analysis for the Navier-Stokes Equations on Unstructured Meshes Using Complex Variables", AIAA Paper No. 99-3294, Proceedings of the 17th Applied Aerodynamics Conference, 28 Jun. 1999.

[6] Newman, J. C. , W. K. Anderson, D. L. Whitfield, "Multidisciplinary Sensitivity Derivatives Using Complex Variables", MSSU-COE-ERC-98-08, Jul. 1998.

[7] Saff, E. B., A. D. Snider, *Fundamentals of Complex Analysis*, Prentice Hall, New Jersey, 1976.

[8] Olver, F. W. J., "Error Analysis of Complex Arithmetic", *Computational Aspects of Complex Analysis*, pp. 279-292, 1983.

[9] Lyness, J. N., and G. Sande, "ENTCAF and ENTCRE Evaluation of Normalized Taylor Coefficients of an Analytic Function", *Com. ACM*, Vol. 14, No. 10, 1971, Washington DC, 1967, pp. 124-134.

[10] Lutz, M., *Programming Python*, O'Reilly, New York, Cambridge, 1996.

[11] `http://www.python.org`

[12] `http://aero-comlab.stanford.edu/jmartins`

[13] Holden, M. E., "Aeroelastic Optimization using the Collocation Method", *PhD Thesis*, Stanford, May 1999.

[14] Adelman, H. M., R. T. Haftka, "Sensitivity Analysis of Discrete Structural Systems", *AIAA Journal*, Vol. 24, No. 5, May 1986.

[15] Jameson, A., W. Schmidt and E. Turkel, "Numerical Solution of the Euler Equations by Finite Volume Methods Using Runge-Kutta Time Stepping Schemes", AIAA Paper No. 81-1259, June, 1981.

[16] J. Reuther, J. J. Alonso and A. Jameson, "Constrained Multipoint Aerodynamic Shape Optimization Using an Adjoint Formulation and Parallel Computers: Part I", *Journal of Aircraft*, 36(1):51-60, 1999.

Fortran Function	Mathematical Standard	Definition	Analytic Continuation
`abs`	✘	$abs(z) = \begin{cases} -z & \Leftarrow x < 0 \\ +z & \Leftarrow x \geq 0 \end{cases}$	First derivative discontinuous at $x = 0$.
`exp`	✔	$e^z = e^x(\cos y + i \sin y)$	
`sqrt`	✔	$\sqrt{z} = \sqrt{\lvert z \rvert}\left(\cos\left(\frac{\text{Arg}(z)}{2}\right) + i \sin\left(\frac{\text{Arg}(z)}{2}\right)\right)$	Non-continuous on non-positive real axis ($x \leq 0$, $y = 0$).
`sin`	✔	$\sin(z) = \frac{e^{iz} - e^{-iz}}{2i}$	
`cos`	✔	$\cos(z) = \frac{e^{iz} + e^{-iz}}{2}$	
`tan`	✔	$\tan(z) = \frac{e^{-iz} - e^{iz}}{e^{-iz} + e^{iz}}$	Not defined when $\cos(z) = 0$
`log`	✔	$\log(z) = \log \lvert z \rvert + i \,\text{Arg}(z)$	Not defined for $z = 0$. Non-continuous on non-positive real axis ($x \leq 0$, $y = 0$).
`log10`	✔	$\log_{10}(z) = \frac{\log(z)}{\log(10)}$	Not defined for $z = 0$. Non-continuous on non-positive real axis ($x \leq 0$, $y = 0$).
`asin`	✔	$\arcsin(z) = -i \log[iz + (1 - z^2)^{\frac{1}{2}}]$	
`acos`	✔	$\arccos(z) = -i \log[z + (z^2 - 1)^{\frac{1}{2}}]$	
`atan`	✔	$\arctan(z) = \frac{i}{2} \log\left(\frac{1 - iz}{1 + iz}\right)$	Not defined when $z = \pm i$.
`atan2`	✔	$\arctan2(z_1, z_2) = \arctan(z_2/z_1)$	
`sinh`	✔	$\sinh(z) = \frac{e^z - e^{-z}}{2}$	
`cosh`	✔	$\cosh(z) = \frac{e^z + e^{-z}}{2}$	
`tanh`	✔	$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	
`dim`	✘	$\dim(z_1, z_2) = \begin{cases} z_1 - z_2 & \Leftarrow x_1 > x_2 \\ 0 & \Leftarrow x_1 \leq x_2 \end{cases}$	Non-continuous derivatives when $x_1 = x_2$.
`sign`	✘	$\text{sign}(z_1, z_2) = \begin{cases} +\lvert x_1 \rvert & \Leftarrow x_2 \geq 0 \\ -\lvert x_1 \rvert & \Leftarrow x_2 < 0 \end{cases}$	Non-continuous derivatives when $x_2 = 0$.
`max`	✘	$\max(z_1, z_2) = \begin{cases} z_1 & \Leftarrow x_1 \geq x_2 \\ z_2 & \Leftarrow x_1 < x_2 \end{cases}$	Non-continuous derivatives when $x_1 = x_2$.
`min`	✘	$\min(z_1, z_2) = \begin{cases} z_1 & \Leftarrow x_1 \leq x_2 \\ z_2 & \Leftarrow x_1 > x_2 \end{cases}$	Non-continuous derivatives when $x_1 = x_2$.

Table 5: Fortran intrinsic functions and their complex definitions. ✔ means the presented definition is the standard mathematical one, ✘ mean the definition is new, $z = x + iy$. Note that in the case of `abs`, although it has a standard definition for complex arguments, that definition is not analytic.