**Solution Derivatives Via Complex Variables**

The complex variable approach for generating solution derivatives of numerical schemes is not a new idea, having been proposed in 1967 by Lyness and Lyness and Moler. However, it did not see much use until it was revived by Squire and Trapp in 1998 ("Using Complex Variables to Estimate Derivatives of Real Functions" SIAM Review, Vol 10, No. 1, March 1998, pp.110-112). Kyle Anderson popularized the method at NASA Langley ("Sensitivity Analysis for the Navier-Stokes Equations on Unstructured Meshes Using Complex Variables.", AIAA Paper No. 99-3294, June 1999).

The method is based on the Taylor series expansion of a complex function

f(x+ih) = f(x) +ihf'(x) -0.5h2f''(x) + O(ih3)

so that equating the imaginary parts on the left and right hand sides gives

f'(x) = Im[f(x+ih)]/h + O(h2)

The derivative is subject to a truncation error on the order of the square of the step size, but, quite importantly, is not obtained by differences of two nearly identical numbers as is done in finite differences. Thus, unlike finite differences, the step size h can be made as small as desired without subtractive errors (i.e. numerical precision) ever swamping the truncation error. A step size of 1.e-6 is generally quite sufficient.

Although not available at the time that CFL3D was converted to complex variables, Joaquim Martins at Stanford University has developed a [script](http://aero-comlab.stanford.edu/jmartins/complex-step/index.html) to automate the conversion of a FORTRAN 90 code to complex variables.

A complex version of the parallel code can be generated by typing:

**make cfl3dcmplx\_mpi**

in the build directory. Alternatively, a complex sequential version of the code can be generated by typing:

**make cfl3dcmplx\_seq**

Because the entire code is rendered complex, any real data that comes into the code can potentially be an independent variable with respect to which derivatives of the solution can be obtained. All that is required is to simply give that variable a small complex component. The following variables are already implemented into the code, with the input variable for the appropriate "h" step given in parentheses, and are accessable via [Keyword input](http://cfl3d.larc.nasa.gov/Cfl3dv6/cfl3dv6_new.html#keyword):

* Mach number (xmach\_img)
* unit Reynolds number (reue\_img)
* angle of attack (alpha\_img)
* yaw angle (beta\_img)
* free stream temperature (tinf\_img)
* geometry (geom\_img)
* rotation rate about the x-axis (xrotrate\_img)
* rotation rate about the y-axis (yrotrate\_img)
* rotation rate about the z-axis (xrotrate\_img)

Note that for geometrical derivatives, the value of geom\_img must correspond to the step size used to generate the **complex-valued** grid. To split a complex-valued grid, use splittercmplx (see [Block Splitter](http://cfl3d.larc.nasa.gov/Cfl3dv6/cfl3dv6_splitter.html)). The resulting derivatives of Cl, Cd,Cm, etc are output to the file cfl3d.sd\_res - this is a file similar to the usual CFL3D convergence history file cfl3d.res for the solution convergence. For example, to determine the derivatives with respect to angle of attack, start with a standard CFL3D input file set up for the angle of attack of interest, and then add the [Keyword input](http://cfl3d.larc.nasa.gov/Cfl3dv6/cfl3dv6_new.html#keyword):

>

alpha\_img 1.e-8

<

The complex-valued code will require twice the memory and approximately three times the CPU time of the real-valued code. Derivatives can be obtained by central differences for twice the CPU time. However, finite differences are subject to large truncation error if the step size is too small and large subtractive error if the step size is too small, so factoring in the "trial and error" of step size choice can easily make the complex-variable approach very competetive, CPU-wise. The following figures illustrate this.

The first figure shows the convergence of a 32 block grid for the ONERA M6, using roughly 106 grid points. Convergence is quite acceptable, with the drag varying less than 1 count (0.0001) after 200 coarse level + 200 medium level + 500 fine level iterations. However, the residual does "hang" on the level of roughly 10-9. Evaluating the derivative of drag with respect to angle of attack by calculating two different solutions with small differences in angle of attack and using finite differences with various step sizes leads to wildly different, mostly garbage, results. For comaprison, the derivative computed using the complex code converges as well as the function. Total "cost" of the single complex derivative calculation was 3 times the cost of a standard solution. For the finite derivative result, a total of 6 runs were made, but each was run roughly twice as long on the fine level in order to try and get decent derivatives. This was a "real-world" scenario, in that the initial choice of finite difference step size was chosen as 10-6 since that had worked quite well in other cases.

* [solution convergence](http://cfl3d.larc.nasa.gov/Cfl3dv6/Gifs/m6_soln.gif)
* [derivative convergence](http://cfl3d.larc.nasa.gov/Cfl3dv6/Gifs/m6_complex_vs_fd.gif)

The next figure shows a comparison of the derivative of drag with respect to a geometric design variable (inboard twist) for an inviscid HSCT configuration computed using the complex-variable approach in Version 6 and an earlier parallel version of CFL3D (Version 4.1hp) that had been passed through the ADIFOR automatic differentiation tool. It can be seen that the final derivatives are identical, with quite similar convergence rates.

* [complex-variable vs. ADIFOR convergence](http://cfl3d.larc.nasa.gov/Cfl3dv6/Gifs/cmplx_vs_adifor.gif)

**IMPORTANT NOTE #1: restart files are NOT compatable between the "regular" version of CFL3D and the complex version.**

**IMPORTANT NOTE #2: As of March, 2007, the Intel Version 9 compiler has major problems with complex cases in CFL3D. If you use Intel, consider compiling with a different version.**