BY MICHAEL SHERMAN

Complex Step Derivatives:
How Did I Miss This?

This result is so surprising you have to see it to believe it.

One of the tasks faced by every scientific programmer sooner or later is the need to compute the derivative **f**′(*x*) from code for the original function **f**(*x*). This need arises in design and minimization problems, for example. In practice **f** is often an enormous, messy, "legacy" numerical computation, and *x* is a vector of many arguments. We all know the standard finite difference trick:

$$\mathbf{f}'(x) \approx (\mathbf{f}(x+h)-\mathbf{f}(x)) \,/\, h$$

which converges exactly on **f**′ in the limit $h \rightarrow 0$ ... except *that* can only happen in a calculus textbook! In practice, roundoff error ruins the accuracy of the difference **f**(*x*+*h*)–**f**(*x*) as the arguments get closer together. So we try to balance roundoff error caused by *h* being too small against "truncation" error from *h* too large. Optimal balance is *usually* found near $h=\sqrt{\varepsilon}$ where ε is the precision with which **f** can be calculated, although exceptions abound. On a good day, this yields seven correct digits of **f**′ when **f** has sixteen. Most of us think of that as "about half the accuracy" but a more sober perspective is that we lost *nine orders of magnitude*!

We can improve this somewhat by calculating more terms in the Taylor expansion of **f**′; for example, (**f**(*x*+*h*)–**f**(*x*–*h*))/2*h* (central difference) gives a few more digits at twice the expense. We're still down *six* orders of magnitude (assuming we picked *h* well). William Press expressed it best:

It is disappointing, certainly, that no simple finite-difference formula ... gives accuracy comparable to the machine accuracy.—*Numerical Recipes in* C++ (2003)

```
typedef double        Real;
typedef complex<Real> Complex;
const Complex i(0,1); // sqrt(-1)

Real    f(Real    x) {return sin(3.*x)*log(x);}
Complex fc(Complex x) {return sin(3.*x)*log(x);}

int main() {
    Real h2=1e-7, h3=1e-5, hc=1e-20, x=0.7;
    Real dfdxFinite   = (f(x+h2) - f(x))   / h2,
         dfdxCentral  = (f(x+h3) - f(x-h3))/(2*h3),
         dfdxComplex  = fc(x+hc*i).imag()  / hc,
         dfdxAnalytic = sin(3*x)/x
                        + 3*cos(3*x)*log(x);

    printf("f' finite  = %.16f\n",  dfdxFinite);
    // ...
}
```

But maybe we all did too much science and not enough math. I recently stumbled on a paper[1] reporting an amazing result from complex analysis:

$$\mathbf{f}'(x) \approx \mathrm{Im}[\mathbf{f}(x+h\mathbf{i})] \,/\, h$$

This says to perturb **f** along the *imaginary* axis, and then take the imaginary part of the result. Otherwise it looks deceptively like the usual finite difference formula. But look again—this one contains no subtraction and hence no roundoff error. So we could hope to make *h* smaller to reduce the truncation error as well. Here is the amazing part: you can make *h* as small as you like, and as long as $h<\sqrt{\varepsilon}$ you'll get the derivative to *machine accuracy*. Of course you do have to modify **f** to accept a complex argument. That's easy in languages like C++ and FORTRAN with built-in complex numbers, and some automated tools have also been developed.

This result is so surprising you have to see it to believe it. The inset shows a complete C++ program that differentiates **f**(*x*)=`sin`(3*x*)`log`(*x*) by finite, central, and complex step differencing (in yellow), and analytically to check the answer. Here is

```
f' finite   = 1.7733539392494890
f' central  = 1.7733541058356781
f' complex  = 1.7733541062373444
f' analytic= 1.7733541062373446
```

the output, with correct digits highlighted: Complex step matched to *sixteen* decimal places, full machine precision. I chose 10^{-20} as the complex step size, but 10^{-100} works just as well!

Simbios Center faculty member Michael Levitt recently reported a breakthrough in coarse grained molecular modeling of myosin. He replaced a numerical difference calculation of a large matrix with the complex step method, and can now closely match all-atom normal modes with a simplified model. Perhaps more importantly, he now has a reason to gloat about being a FORTRAN programmer!

There is much more to learn about this fascinating idea, including some practical issues to consider, a deep relationship with automatic differentiation theory, and historical roots in work done in the 1960s by Simbios Scientific Advisor Cleve Moler. For more information, see the referenced paper. Then give it a try yourself and let us know what happens. □

DETAILS

Michael Sherman is Chief Software Architect for the Simbios Center.

FOOTNOTES

1 Martins, J. R., Sturdza, P., and Alonso, J. J. 2003. The complex-step derivative approximation. *ACM Trans. Math. Softw.* 29(3) (2003).