From *SIAM News*, Volume 44, Number 9, November 2011

ICIAM 2011

Chebfun: Get Out from Under the Hood, and Into the Fast Car

By Nilima Nigam

The Chebfun car with Chebfun authors, clockwise from lower left, Toby Driscoll, Nick Trefethen, Rodrigo Platte, and Ricardo Pachón. The keys to this car are readily available, and it's fun to drive.

A numerical analyst and a car mechanic have several traits in common. Both like to invent, analyze, tweak, tinker with the goal of optimizing performance; both are happy to admire the handiwork of colleagues, and can gladly spend hours discussing the finer points of the inner workings of objects of interest. Both may have collections of favorite projects that are in development and not quite ready for broad consumption.

But sometimes, just sometimes, it's fun to climb into that shiny, fast, highly optimized car whose keys someone left in the ignition, and take it for a spin. This is an extremely personalized account of how well-developed, robust, well-documented, and easy-to-use high-performance software can simplify the life of an applied mathematician, even when employed outside the "recommended usage."

In the early years of this millennium, my colleague Sherwin Maslowe (at McGill University) told me about a fascinating problem with a rich history: *How do helical perturbations to a columnar vortex propagate?* Think about the counter-rotating vortices shed from the wing tips of aircraft, especially during takeoff or landing. Smaller aircraft must avoid these vortices, and indeed such considerations determine the timing between landings and takeoffs at airports. Studying such vortices and their stability, therefore, has occupied many scientists and engineers.

Lord Kelvin first studied this problem in the specific case of a fluid in rigid rotation confined in a cylinder. In the situation of interest to us, the perturbations are superimposed on an unbounded vortex with a continuous velocity profile $(0,V(r),0)$ (in cylindrical coordinates).

Sipp and Jacquin suggested, based on a linear viscous stability analysis, that critical-layer Kelvin modes would be damped. My colleague had thought about examining the nonlinear effects in the critical layer and wondered whether these modes could actually be neutral. Viscous effects also play a role, but we focused first on the effects of nonlinearity in the critical layer. Deriving the appropriate equations to study these questions represented a large effort in itself.

Once we had the relevant equations, we still had two important questions to resolve: *Where is the critical layer? What happens to the perturbations inside the critical layer?* The first of these questions requires the solution of a highly nonlinear eigenvalue problem for a second-order differential operator. The second requires the solution of a system of four nonlinear coupled PDEs in two independent variables. The system is degenerate and not strictly hyperbolic. We were able to use the method of characteristics to convert it into a system of coupled differential–algebraic equations, but they were still nonlinear. We found colliding characteristics numerically, and then had to derive conditions within the region of closed characteristics. With its critical interplay between modeling, analysis, and numerics, this was a genuine "applied mathematics" problem.

In the end, we tried a combination of asymptotics, symbolic computation, computations by hand, built-in Matlab solvers, and a variety of spectral collocation strategies until we found approaches that worked. We had to locate discretization parameters in each situation, we had to check whether methods converged, we had to revisit the discretizations as soon as one of the model parameters changed. In short, this process was something we enjoyed tremendously, but it was neither central nor efficient as a means of answering the specific questions at hand. After nearly a year's worth of work, however, we had nice results, and we sent off our paper.

Fast-forward two years. In the summer of 2009, a colleague at Simon Fraser University (where I had moved) pointed me to "this cool software called Chebfun." I got a copy from him and started playing around. Chebfun, an open-source system in Matlab, overloads Matlab commands for functions. So, for example, locating the minimum of a continuous function can be done using the `min` command with a function as an argument. Looking "under the hood," one finds familiar ideas like Chebyshev expansions, automatic differentiation, edge detection, and barycentric interpolation. Cleverly combining these ideas, the authors have built a platform that quite robustly makes it possible to handle even piecewise smooth functions with high accuracy, *and with ease*. It is handy if one wants to `sum` a function, and get its integral without worrying about describing quadrature points and tolerances.

We were told that Chebfun was neither ready to be used for 2D problems, nor always appropriate for highly nonlinear eigenvalue problems. However, I figured that maybe I could use Chebfun to validate the numerics we had done a few years ago. The actual coding would be easy; I wouldn't have to worry about selecting discretization parameters, and I could use the familiar Matlab commands, albeit overloaded for functions. Maybe it would work, maybe it wouldn't, but I'd find out fast.

To my very pleasant surprise, Chebfun proved remarkably easy to install, learn, and use. It took me some time to think about how to use a code specifically designed for 1D problems in our 2D setting, but the special structure of our problem (and our prior experience) made this possible. I started these calculations again, and within two days had computational results arrived at previously only after months of experimentation. I was even able to get higher accuracy for the PDE part of the calculation than was possible before.

This episode has provoked a shift in my thinking. While I still like to tinker with my own codes and prototype algorithms myself, for more complex applications I see how well-curated and -maintained software can help focus attention on the problem itself. Most of us already use existing libraries, such as ARPACK. I believe the new generation of open-source high-performance software packages, such as Chebfun, deal.II, and Clawpack, will enable us to deploy powerful approximation strategies, with ever more ease, on increasingly complex problems. Why not rely solely on commercial software, then? The answer is simple: We can't open the hood, jiggle the nuts and bolts, and replace standard parts with newer parts we've made. And that, as we all know, takes all the fun out of both cars and numerical analysis.

Nilima Nigam is an associate professor of mathematics at Simon Fraser University.