Computer Lab I: Expanding and interpolating $\sin(x)$
Computational Physics 4480/7680, Astro 7690

James Sethna

Last modified at January 22, 2013, 8:33 pm

How do compiler writers evaluate mathematical functions like $\sin(x)$? How can we generate efficient implementations of our own special functions, or even write faster (but less accurate) versions of standard functions?[1]

A standard technique in computation is to approximate expensive functions by fits to pre-computed values (interpolation) or by generating approximate functional forms (expansion). Interpolation can be linear, polynomial, spline, rational polynomial, barycentric rational polynomial, etc. Expansion methods include Taylor series and Padé approximates; least-squares fits and Chebyshev polynomials combine aspects of each. The group projects for the first half of the semester will focus on first implementing a variety of these methods, then testing them against one another for speed and accuracy, and finally generating ten-minute presentations on various issues, methods, and techniques we addressed during this exercise.

In this first computer lab, we will launch each group into implementing one method of each type: Taylor series and linear interpolation.

Taylor: *Using the computational environment of your choice, write a routine that evaluates the Taylor approximation to* $\sin(x)$ *with* N *terms. Test its accuracy (root-mean-square error) and speed for a random distribution of* 10^6 *points between zero and* 2π*, for various* N*.*

Linear interpolation: *Using the computational environment of your choice, store the values of* $\sin(x)$ *at equally-spaced points* $x_n = n\Delta$*. Write a routine that linearly interpolates between these evaluated points. Test its accuracy (root-mean-square error) and speed for a random distribution of* 10^6 *points between zero and* 2π*, for various* Δ*.*

[1] For example, Prof. Christopher Myers and I many years ago worked on a problem where we needed to calculate $\sin(x)$ an enormous number of times, for randomly distributed points x. To speed things up, we traded accuracy for speed; Myers implemented a spline fit to $\sin(x)$ which was much faster than that provided by the system, but which was accurate to fewer decimal places.