(2.3) (Group project) Building your own Sine.

A standard technique in computation is to approximate expensive functions by fits to pre-computed values or approximate functional forms. For example, Christopher Myers (a colleague in computational sciences here) 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 \( \sin(x) \) between \(-\pi\) and \(\pi\) which was much faster than that provided by the system, but which was accurate to fewer decimal places.

In Numerical Recipes, there are many different methods for approximating functions.\(^1\) I propose that we split up into groups, and see which methods produce the fastest code for a given accuracy. Groups with stronger backgrounds in scientific computing should choose methods farther down along the list...

1. Taylor series
2. Linear interpolation
3. Splines
4. Polynomial interpolation
5. Least squares fit polynomial (use Legendre functions, not covered in NR)
6. Chebyshev polynomials
7. Padé approximants
8. Rational function interpolation
9. Barycentric rational interpolation

This is the first part of a project which will consume much of the first part of the semester. For the first two weeks, use your preferred working environment to get your method of choice to work, and estimate the maximum error as a function of the order of the approximation (number of interpolation points, degree of the function, etc.) I recommend individually reading the relevant sections of NR and/or the Web, then meeting as a group for a couple of hours of intense programming.

\(^1\)They also mention in the introduction that modern CPU's are very fast at computing trigonometric functions, so we may not gain much.