Worksheet 6 (April 7th, 2021)

1 Error in Polynomial Interpolation

In this worksheet you will explore a bit how good polynomial interpolation is *before* you watch the pre-recorded lecture. This will give you not just practical skills doing polynomial interpolation but also some insights into the theory. Note that parts of the code you develop here will be reused in later worksheets, so make sure you exchange codes and polish them and save them for future reuse.

You will try to approximate with a polynomial the following two functions:

1. (Start with this one) An "easy" function

$$f_1(x) = \exp(x) \quad \text{on} \quad x \in [-2, 2].$$

2. The "hard" Runge function

$$f_2(x) = \frac{1}{1+x^2} \quad \text{on} \quad x \in [-5, 5].$$

1.1 Polynomial interpolation code

In this part you will write Matlab code to do the polynomial interpolation and then plot the function $f(x)$, the $n+1$ interpolation nodes, and the interpolating polynomial $p_n(x)$. In a separate figure, plot the error $|f(x) - p_n(x)|$ with a logarithmic y axis, and report the maximum error over the interval. Make sure that this code is written so that it works for *any* function $f(x)$, choice of the polynomial degree n, the interpolation nodes x_k, and interval $x \in [a, b]$, even though for testing the code you can use equispaced nodes. That is, do *not hard-wire* these choices into the code so you can easily change them in the rest of this worksheet.

To plot the function/polynomial and to estimate the (maximum) error in the polynomial approximation, use a fine grid with $\sim$1000 equi-spaced nodes in $[a, b]$.

To construct the polynomial, it is OK to use the $polyfit$ function, but note that this will solve a linear system with the Vandermonde matrix and Matlab may complain about ill-conditioning for $n > 14$ or so. Therefore, in principle we should not trust the polynomial interpolant blindly. If you have time and interest, try using pseudo-inverse to solve the linear system instead to help (to some extent) with the ill-conditioning (see the previous worksheet). To evaluate the polynomial you can use $polyval$.

Test your code with $n = 12$ for both functions f_1 and f_2. [*Hint: The interpolating polynomial must pass through the interpolation points, this is why it is important to plot them as well.*] Is the polynomial interpolant a good approximation to $f(x)$ for the "easy" and "hard" functions, respectively?

1.2 Improved nodes

As we will learn in lectures, it is much better to use a non-uniform grid of nodes. Usually, coordinates x_k for the $n+1$ "good" nodes are computed/provided on the interval $x \in [-1,1]$. By rescaling and shifting the x coordinate, i.e., by doing a linear transformation of x that maps $[-1,1]$ to $[a,b]$, it is easy to make these points be on any interval $[a,b]$ of interest. Figure out how to do that on your own.

For example, a good grid of nodes to use are the Chebyshev nodes, given by (on $[-1,1]$)

$$x_k = \cos\left(\frac{2k+1}{2(n+1)}\pi\right), \quad k = 0,\ldots,n,$$

which does not include the endpoints. Equally as good is to use

$$x_k = \cos\left(\frac{k}{n}\pi\right), \quad k = 0,\ldots,n,$$

which includes the endpoints. Choose one or try both of these choices.

Try interpolating using the Chebyshev nodes for $n = 12$ for the Runge function and compare to what you got for equi-spaced nodes before. [*If your code was written well trying out new nodes should be very easy.*] Now try $n = 32$ and report how good the polynomial interpolation is for the Chebyshev nodes, versus for equi-spaced nodes. What do you conclude from this?

1.3 Improved numerics

Instead of using $polyfit$ and $polyval$, you can use the Barycentric interpolation formula to avoid solving ill-conditioned linear systems. If time permits, try to implement your own code for evaluating the barycentric formula and compare to using $polyval$ for say Runge's function evaluated at $x = 0.1$ for $n = 12$; this will test your barycentric code. It is great if you get this far in this worksheet, but try to finish the rest on your own.

Once your code is tested, plot the barycentric polynomial interpolant (say, with a dashed line) on the same plot as the Vandermonde one (plotted with, say, a solid line), and compare the two for $n = 32$ visually. Now try $n = 52$ and see whether using the barycentric formula helped or not.

Note that the barycentric formula, while numerically the best one we have, still suffers from division by zero when $x = x_k$ is an interpolation point. To avoid division by zero $y_k = x - x_k$, you can use the Matlab code (recall that *eps* is on the order of 10^{-16})
y(abs(y)<eps) = eps;
which sets any value close to zero to $\sim 10^{-16}$.