Interpolation of Base-2 Logarithm
Due Thursday, Oct 4, 2012

This is a major assignment that may seem simple at first glance. Do not be fooled – the difficulty is in the details.

You are to analyze and compare the efficiency of using four different piecewise polynomial interpolation schemes for approximating the base-2 logarithm function on the interval $[1,2]$ (I will explain later why this interval is important.)

The four different schemes are:

1. Equispaced piecewise linear interpolation
2. Variably spaced piecewise linear interpolation
3. Equispaced cubic Hermite interpolation
4. Variably spaced cubic Hermite interpolation

In all cases the maximum error is to be no more than $1 \cdot 10^{-7}$ over the interval $[1,2]$.

1. First, for each of the four methods determine the nodes $x_1, x_2, ..., x_n$. Consider carefully how to use the "excess error": if you determine that n nodes are required what is the best way to distribute them? One way is to have a shorter interval at an end. Another way is to reduce the error below $1 \cdot 10^{-7}$ without increasing the number of points. The second way is more efficient so you should use it. Assuming $x_1 = 1$, think of the location of x_n as being a function of the tolerance ε. You may at first determine that for $\varepsilon = 1 \cdot 10^{-7}$, you could have x_n greater than 2. But you are asked for the approximation only on $[1,2]$. So you try to determine ε so that the associated $x_n = 2.$ This is fairly easy to do with the equispaced cases. It will take some work (perhaps programming) for the variably spaced cases.

Remember your solutions will be the four values of n and the associated four set of nodes $x_1, x_2, ..., x_n$.

2. Now analyze each of the four situations by constructing algorithms. You may express the algorithms in any mathematical notation or pseudo-code that is clear. For each of the four cases construct two algorithms: one that is **time efficient** and one that is **space efficient**. To make things simpler, for time efficiency consider only the real arithmetic operations and comparisons. Furthermore, assume the time required for each arithmetic operation is the same (e.g., an algorithm with three multiplications, four additions and two comparisons has equal time efficiency to one with three subtractions, three divisions, and three comparisons). For space efficiency ignore the space required for code and count only that for array storage (e.g., an algorithm with three arrays of length ten has equal space efficiency to one with a single array of length thirty). *Consider that you may do pre-computation to gain efficiency.* Also consider how to do interval searches with equispaced nodes.

Your solutions will have the eight algorithms and the time and space requirements of each.