CS 322: Introduction to Scientific Computing
Spring 2003
**Problem Set 1**

Handed out: Wed., Jan. 29.

Due: Wed., Feb. 5 in lecture.

The policies for this (and other problem sets) are as follows:

- You should hand in your on-time problem set at the beginning or end of lecture on the day it is due in the box at the front of the room. Problem sets handed in elsewhere (TA's office, Upson 303, etc.) will be considered late. See the next bullet.
- Late papers may be handed in up to 24 hours late. For instance, this problem set may be handed in up to 11:00 on Feb. 6. You can hand in a late paper in Upson 303. Late papers get an automatic deduction of 10%. The full late penalty is applied even if you turn in part of the solution on time.
- Problem sets may be done individually or in teams of two. Put your name or names on the front page. Re-read the academic integrity statement on the web for the policy concerning working in larger groups.
- Problem sets count for 20% of the final course grade. The lowest scoring problem set will be dropped.
- You need Matlab for some of the questions. Matlab is available in the following CIT labs: Upson and Carpenter.
- If you need clarification for a homework question, please either ask your question in section, lecture, or office hours or else post it to the newsgroup cornell.class.cs322. The professor reads this newsgroup and will post an answer.
- Write your names and section number (like this: "Section 2") at the top of the front page of your paper. This is the section where your graded paper will be returned. As a reminder, Section 1 is Th 12:20, Section 2 is Th 3:35, Section 3 is F 2:30, Section 4 is F 3:35.

1. A standard form polynomial $p(x) = a_d x^d + a_{d-1}x^{d-1} + \cdots + a_0$ is said to be "centered at 0", whereas a polynomial centered at a real number $c$ would be written $h_d(x - c)^d + h_{d-1}(x - c)^{d-1} + \cdots + h_1(x - c) + h_0$. Write (on paper) a Matlab fragment that takes as input the coefficients $a_0, \ldots, a_d$ of a standard-form polynomial stored in an array `a` indexed `1:d+1` and a scalar `c`, and produces as output the coefficients $h_0, \ldots, h_d$ of the same polynomial with its center shifted to $c$. For example, if the input is `a(1:3)=[4,5,6]` and `c=-2`, then the output of the fragment is `h=[18,-19,6]` since $6x^2+5x+4 = 6(x+2)^2 - 19(x+2) + 18$. Hint: use a temporary array `b` that stores the coefficients such that $x^k = b_k(x-c)^k + b_{k-1}(x-c)^{k-1} + \cdots + b_0$. [**NOTE: The formula in the previous sentence was corrected on 1/30/03.**] On each iteration, add a multiple of $b$ to the vector $h$, and then update $b$ for the next exponent $k$.

2. Inequality (2.2) in the text was presented in lecture for the accuracy of an interpolant $p$ to a continuous function $f$. Determine by experimenting whether the bound increases or decreases as the number of interpolation points increases for the Runge function $f(x) = 1/(1 + 25x^2)$. Hint: in order to carry this out, you need high derivatives of $f$. Evaluate the derivatives at 0. It is tedious to compute derivatives by hand, but Matlab's symbolic toolbox can do them for you. Follow the example of the script `exampldiff.m` on the course home page that evaluates derivatives of the function $1/(x + 1)$. Hand in relevant Matlab printouts.

3. Consider interpolating the Runge function on p. 91 of the text. It turns out that you can find a much better interpolant over $[-1, 1]$ by choosing x-coordinates that are not evenly spaced. Come up with a set of 11 x-coordinates such that the interpolant matches the function much better than the interpolant depicted in Figure 2.4. Make a plot analogous to Figure 2.4, that is, a plot showing the Runge function and your improved interpolant on the same axis, with asterisks for the interpolation points.

   Hand in: a plot and a listing of the script that you used to generate the plot.

   Some hints: You can download the script `RungeEg.m` from the book's website and modify it to answer the question. If you prefer, you can use the `polyfit` function (built into Matlab) for interpolation, and the `polyval` function for evaluating polynomials rather than the book's functions `InterpN` and `HornerN`. The `help` command gives help on both. Note: Use `polyfit` in the interpolation sense, not the least-squares sense. This means: when interpolating a set of $n$ datapoints with `polyfit`, the third argument should be equal to $n-1$. Ignore the second return argument from `polyfit`. It is suggested that you try crowding the x-coordinates more near the interval endpoints.