Assignment #2

Due October 11, 2015, 2:15 p.m
(after that day assignments should be put into the mbox of Shuai SHAO)

(1) The original demonstration of Newton’s method was done by Newton almost 350 years ago. He used it for finding roots of cubic polynomials. Since you aspire to be “Newton of the 21st century”, you try to do the same (albeit, a few years later). For that, choose a cubic polynomial that has a root (i.e., a zero) in the interval \((k, k + 1)\), with \(k\) the last digit in your UW ID. The cubic polynomial must involve at least three terms, i.e., do not choose something like \(x^3 - 2 = 0\). Run Newton’s algorithm on your problem. Then, find that same root by using the bisection method, and by using the secant method (use \texttt{matlab}; FYI, Newton neither used \texttt{matlab} during his demo, nor had a UW ID). Derive from the output the rate of convergence of each method (to the extent that this is possible).

(2) You finally found at home the famous \textit{Kaput} calculator that grandpa used to have in his heyday. It turns out that the calculator can really do \(+/-/*\) (up to 16 significant digits). Alas, it does not perform division so well: it computes only the first three significant digits in the result. Since by now you are a true fan of Newton, you decide to use his method to improve Kaput’s division accuracy. To this end, you use the function

\[
f(x) = 1 - \frac{1}{ax}
\]

for the computation of the number \(1/a\).

(a) Explain how this method is implemented; your explanation should cover at least the following points: (i) how does the Newton method, when applied to the above function, compute the number \(1/a\). (ii) What is the “trick” here; precisely, how are we able in this way to divide 1 by \(a\) better than Kaput can do, using only Kaput for the calculations. To understand the process better, do the first iteration for \(a = 17\). Take your initial seed for \(1/17\) to be Kaput’s approximation of \(1/17\). Show that no division is used in the iteration (which is necessary if we agree that we do all the computations on Kaput)!

(b) Use the expression for the error in Newton’s method and derive the number of iterations that are needed for 16 (significant) digit accuracy. An important remark: the number of iterations crucially depends on the accuracy of your initial seed. Your initial seed, naturally, is the division output of Kaput.

Note that we did not ask you to actually write any code, only to estimate \textit{a priori} the requisite number of iterations.

(3) About 300 years after Newton, the famous numerical analyst Wilkinson showed that roots of polynomials are highly sensitive to small alterations in their coefficients. He used the polynomial

\[
p(t) = (t - 1)(t - 2)(t - 3) \ldots (t - 20) - 10^{-8}t^{19}.
\]
(a) Prove that \( p \) has a root in \([20, 22]\), and then use the secant method (you already have a code from (1)...) for finding that root. Your code may contain a \texttt{while} loop for the iterations, but you should try to avoid a loop for evaluating \( p \) (the \texttt{matlab} command \texttt{prod(v)} computes the product of the entries of the row/column vector \( v \)).

(b) In order to appreciate the power of superlinear convergence, find (experimentally) the number of iterations needed in (a) to get to the root within an error \(< 10^{-3}\), find also the number of iterations for accuracy \(10^{-13}\), and compute the ratio of these two numbers. What would have been that ratio had we used bisection instead of secant? (As an estimate for the error in the secant case, take the difference between consecutive approximations. As an estimate for the error in the bisection case, take half the size of your current interval).

(c) What, in your opinion, is the reason for choosing secant here (and not Newton or bisection)? (Give all reasons you may think about.)

(d) Explain how this example helped Wilkinson in making his point. Why, in your opinion, is this an important observation? (Hint: suppose that we try to find polynomial roots by running some program on some machine.)

(4) Find the polynomial of degree 10 that interpolates the function \( \text{arctan} \) at the 11 equally-spaced points \( \text{linspace}(0,5,11)' \) (you may use any interpolation process including those that I used in \texttt{inter} demo. You are not allowed, however, to use the \texttt{matlab} library m-file \texttt{polyfit.m}). Then compute the error over the interval \([-1,6]\) (i.e., evaluate the error on a fine mesh on the interval \([-1,6]\). The \texttt{matlab} command \texttt{linspace(-1,6)} generates such mesh). Try then to see whether similar results are obtained if you increase the number of interpolation points and/or change their distribution. Turn in your code, your output, and a brief account of the conclusions you drew from this experiment.