September 25, 2019

CS412, Fall 19

Prof. Ron

Assignment #2

Due October 09, 2019, 3:45 p.m

(after that day assignments should be put into the mbox of Rocil Machado)

(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 `matlab`; FYI, Newton neither used `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 *Kaput* computer that grandma used to have in her hayday. It turns out that Kaput can handle flawlessly basic `matlab` commands, and it performs perfectly $+/-/*/:$ (up to 8 significant digits). Alas, it does not compute cubic roots correctly: it computes only the first two significant digits in the result. You decide to write a short `matlab` program that will improve *Kaput*'s accuracy vis-a-vis cubic roots: your routine, when run on Kaput, will compute cubic roots at eight decimal accuracy. You decide to use Newton's method to this end.

(a) Choose the correct equation (the number a whose cubic root you seek must be a parameter here), and write the details of the iterations you need to perform. What is your initial seed x_0?

(b) Note that you may assume that the number a whose root you seek satisfies $1 \leq a < 1000$ (why?). Based on that, on the known accuracy of your initial seed x_0, and on the error analysis of the Newton method from class/lecture notes, find the number of iterations you need to run in order to get 8 digit accuracy (it does not matter here whether we talk on "significant digits" or decimal digits, since $a^{1/3}$ lies between 1 and 10.)

Note that we did not ask you to actually write the routine, only to estimate *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

$$p(t) = (t-1)(t-2)(t-3)\dots(t-20) - 10^{-8}t^{19}.$$

(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 `while` loop for the iterations, but you should try to avoid a loop for evaluating p (the `matlab` command `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 7 that interpolates the function `arctan` at the 10 equally-spaced points `[1:.5:4]` (you may use any interpolation process including those that I used in `3inter` demo. You are not allowed, however, to use the `matlab` library m-file `polyfit.m`). Then compute the error over the interval $[-1, 7]$ (i.e., evaluate the error on a fine mesh on the interval $[-1, 7]$. The `matlab` command `linspace(-1,7)` generates such mesh). Try then to see whether similar results are obtained if you increase the number of interpolation points and/or change the function you interpolate. Turn in your code, your output, and a brief account of the conclusions you drew from this experiment.