Homework 1
APPM/MATH 4650 Fall '20 Numerical Analysis

Due date: Friday, September 4, before 5 PM, via Gradescope. **Instructor**: Prof. Becker
Theme: Introduction to floating point computations; stability; conditioning

Instructions Collaboration with your fellow students is OK and in fact recommended, although direct copying is not allowed. The internet is allowed for basic tasks (e.g., looking up definitions on wikipedia) but it is not permissible to search for proofs or to *post* requests for help on forums such as `http://math.stackexchange.com/` or to look at solution manuals. Please write down the names of the students that you worked with.

An arbitrary subset of these questions will be graded.

Turn in a PDF (either scanned handwritten work, or typed, or a combination of both) to **Gradescope**, using the link to Gradescope from our Canvas page.. Gradescope recommends a few apps for scanning from your phone; see the Gradescope HW submission guide.

We will primarily grade your written work, and computer source code is *not* necessary (and you can use any language you want). You may include it at the end of your homework if you wish (sometimes the graders might look at it, but not always; it will be a bit easier to give partial credit if you include your code).

Problem 1: Consider the polynomial

$$p(x) = (x-2)^9 = x^9 - 18x^8 + 144x^7 - 672x^6 + 2016x^5 - 4032x^4 + 5376x^3 - 4608x^2 + 2304x - 512.$$

Note: you can get these coefficients using Matlab's `poly` or Python's `numpy.poly` and specifying that 2 is a root with multiplicity 9; this saves you having to type them in.

a) Produce a plot that shows the evaluation of p for 100 equispaced points in the range $[1.92, 2.08]$, using the following 4 algorithms for evaluating a polynomial (overlay all 4 results on the same plot):

   i. Make your own algorithm to naively evaluate the polynomial using its coefficients

   ii. Use the compact form $p(x) = (x-2)^9$

   iii. Implement Horner's rule

   iv. Use a software library (e.g., Matlab's `polyval` or Python's `numpy.polyval`).

b) Comment on the similarities and differences, and discuss which algorithm you think is most correct, and what possible sources of numerical error might be.

Problem 2: How would you perform the following calculations to avoid cancellation? Justify your answers.

a) Evaluate $f(x) = \sqrt{x+1} - 1$ for $x \approx 0$.

b) Evaluate $f(x) = \sin(2x) - \sin(2a)$ for $x \approx a$. *Hint*: try a trig identity.

[please see other side]

Problem 3: a) Suppose a sequence has the form

$$x_n = Cn^{-\alpha} \tag{1}$$

for some constants C and α. If you plot n on the x-axis and x_n on the y-axis, this does not form a straight line. On what kind of plot would this form a straight line? (e.g., logarithmic scaling on the x-axis? on the y-axis? on both axes?). Justify your answer.

b) Repeat the question above, but for a sequence of the form

$$x_n = D\rho^n \tag{2}$$

for some constants D and $\rho < 1$.

c) Suppose you are given the first 10 terms of a sequence (x_n):

`5.6000 4.4800 3.5840 2.8672 2.2938 1.8350 1.4680 1.1744 0.9395 0.7516`

Estimate (you cannot *prove* anything, since you've only seen the first 10 terms) whether this sequence converges superlinearly, linearly, or quadratically; and explain why you think so. If the sequence fits the form of Eq. (1) or Eq. (2), find the values of the parameters (either (C, α) or (D, ρ)).

d) Repeat the question above, but for the following sequence

`3.0000 1.0607 0.5774 0.3750 0.2683 0.2041 0.1620 0.1326 0.1111 0.0949`

Problem 4: At what rate (i.e., use big-O notation) does $\frac{1}{1-h} - h - 1$ converge to 0 as $h \to 0$?

Problem 5: Let $f(x) = e^x - 1$

a) What is the relative condition number $\kappa_f(x)$? Are there any values of x for which this is ill-conditioned?

b) Consider computing $f(x)$ via the following algorithm:

1: $y \leftarrow e^x$
2: **return** $y - 1$

Is this algorithm stable? Justify your answer

c) Let x have the value $9.999999995000000 \times 10^{-10}$, in which case $f(x)$ is equal to 10^{-9} up to 16 decimal places. How many correct digits does the algorithm listed above give you? Is this expected?

d) Find a polynomial approximation of $f(x)$ that is accurate to 16 digits for all $|x| \leq 10^{-9}$ and prove your answer. *Hint*: use Taylor series, and remember that 16 digits of accuracy is a *relative* error, not an *absolute* one.

e) Evaluate your polynomial approximation at the same value of x as before. How many digits of precision do you have?

f) [Optional] How many digits of precision do you have if you do a simpler Taylor series?

g) [Fact; no work required] Matlab provides `expm1` and Python provides `numpy.expm1` which are special-purpose algorithms to compute $e^x - 1$ for $x \approx 0$. You could compare your Taylor series approximation with `expm1`.