Numerical Methods – HW2 2.1 – 2.5 Due Friday Jan 25

2.1 Errors: Finite Precision Algebra

5 pts Consider a "FAKE" computer that rounds any real number into the 3-significant figure, floating-point decimal format ± d.dd x 10^{d-5}, where each d is a decimal from 0-9, but there are *no* other restrictions (no "infinity", no "forcing a number to be denormal if the d in the exponent is 0").

(a) What is the largest number (*realmax*) you could store in this FAKE computer?

(b) What is the smallest positive number (*realmin*) you could store in this FAKE computer that is not denormal (*i.e.* maintains all 3 significant figures)?

(c) What is the value of machine precision (ε)?

Now let's say you want to add the following four numbers with this FAKE computer:

S = 1.43 + 173 + 41.3 + 6.41 (The *real*, exact sum is clearly 222.14)

(d) What is the final **value** of S in the fake computer if you did the sum in the left-to-right order written above? What is the **percent error** in the value of S (compared to the *real* sum)?

(e) Demonstrate the **ideal order** you should add the four numbers to provide the most accurate value of S. What is that final **value** of S, and what is its **percent error**?

AS ALWAYS: SHOW ALL YOUR WORK !!! JUSTIFY YOUR RESULTS !!!

I care less about you having the right answer than you **showing** me the right process. That means you have to show me all your steps in how you came up with (say) ε, or how and why you ordered your numbers a certain way in part (e).

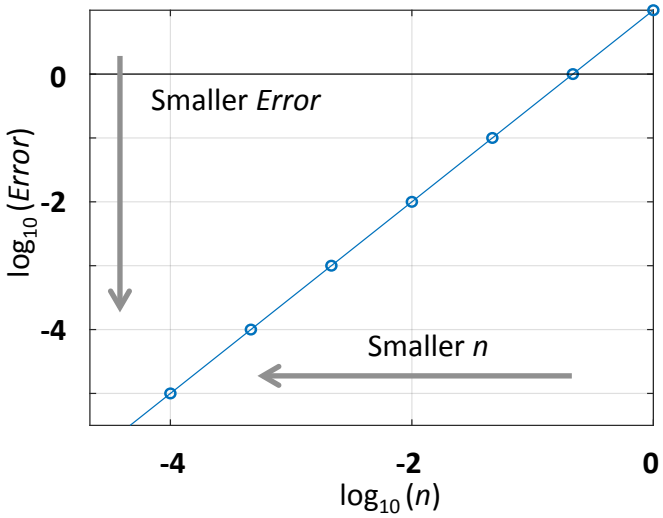
2.2 Analyzing Error plots (Fundamentals of *Logarithms*)

5 pts *This problem has nothing to do with MATLAB. It's to review your high-school "rules" of how logarithms work. Go see the wikipedia page on logarithms if you forgot how to calculate* $log_{10}(0.0001)$*, or expand* $log_{10}(xy)$ *and* $log_{10}(x^y)$.

In many numerical processes, the *Error* varies exponentially with some parameter *n* according to $Error = k\,n^p$ for constants *k* and *p*. We then call ***p*** the "**order**" of the *Error* with respect to *n*. This semester we'll often plot $\log_{10}(Error)$ on the y-axis against $\log_{10}(n)$ on the x-axis to visualize how quickly *Error* goes down as *n* gets smaller.

a) Take the $\log_{10}$ of the equation $\boldsymbol{Error = k\,n^p}$ and determine <u>in general</u> how the constants *k* and *p* relate to the *slope* and *y-intercept* of the straight line $\log_{10}(Error)$ vs. $\log_{10}(n)$ plot you'd get.

I now plotted an example base-10 *log of Error* vs. base-10 *log of n* at right. It shows that *Error* goes down for smaller *n*, and $\log_{10}(Error)$ vs. $\log_{10}(n)$ looks like a straight line.



b) Use the plot to read off the values of ***Error*** for both $n = 1$ and $n = 0.0001$.

c) Apply the general relation you developed in part (a) to determine the **value** of the error *order* ***p*** for the data in this plot.

For everything – ***show your work***! Don't just write down an answer, show me *how* you got it.

Numerical Methods – HW2 2.1 – 2.5 Due Friday Jan 25

2.3 Fundamentals: Converting between Binary and Decimal ***Show your work!!***

a) Convert the following base 2 (binary) representations of numbers to base 10 (decimal):

2 pts

i) $(\mathbf{10110101.010111})_2$ --> "fixed" point binary (many bits to the left of the "point")

ii) $(\mathbf{1}.001011)_2 \times 2^{(\mathbf{10110})_2}$ --> "floating" point binary (like how MATLAB stores it)

b) Convert the following decimal numbers to binary. Please do NOT write it in "floating-point binary" form like (a)(ii) above; just write it in the simpler "fixed" point form like (a)(i).

3 pts

i) 470.125

ii) 12.85 (if this ends up repeating in base 2, just give answer to first 7 fractional "bits" to the right of the "point", *i.e. something*.**bbbbbbb**)

2.4 Recall from class that MATLAB uses standard (IEEE) **double**-precision floating point notation:

$$\text{Any Number} = +/- \overbrace{(1.bbb\ldots bbb)_2}^{52\text{ bits}} \times 2^{\overbrace{(bbb\ldots bbb_2 - 1023_{10})}^{11\text{ bits}}}$$

where each bit *b* represents the digit 0 or 1.

That is, the mantissa is always assumed to start with a 1, with 52 bits afterwards, and the exponent is an eleven bit integer (from 000…001 to 111…110) *biased* by subtracting 1023.

Well, in "my college days" the standard was **single**-precision floating point notation in **32**-bit words:

$$\text{Any Number} = +/- \overbrace{(1.bbb\ldots bbb)_2}^{23\text{ bits}} \times 2^{\overbrace{(bbbbbbbb_2 - 127_{10})}^{8\text{ bits}}}$$

That is, the mantissa is always assumed to start with a 1, with 23 bits afterwards, and the exponent is an eight bit integer (from 00000001 to 11111110) biased by subtracting 127 to allow for an almost equal range of positive, zero, and negative exponents. And it still reserved all exponent bits identically equal to 00000000 for the number 0 (and "denormal" numbers), and 11111111 for ∞.

a) Evaluate *REALMAX* (the largest possible positive number that is not infinity) for my (1990s) **single**-precision computer.

3 pts

i. Express your value as a base 10 floating-point numbers with 3 sig. figs (e.g. 3.45×10^{25}), and **show your work**! (What did you start with in binary, and how did you get that to decimal?)

ii. Compare your value to today's double-precision computers (just type `realmax` in MATLAB)

b) I told you in class that machine precision in MATLAB, which is defined as the *difference between 1 and the next largest storable number*, is approximately 2.2204×10^{-16} in today's double-precision computers (just type `eps` in MATLAB to confirm).

3 pts

i. Evaluate machine precision for the **single**-precision computers from "my day", and express your answer as a base 10 floating-point number with 3 sig. figs (e.g. 3.45×10^{-10}). **Show your work!** (What did you start with in binary, and how did you get that to decimal?)

ii. What's the ratio of machine precision *now* to machine precision *then*? (*i.e.* how much more accurately can we store numbers since 64-bit processors arrived in the early 2000s?)

EVERYTHING FOR 2.1 – 2.4 IS BEING HANDED IN TO CLASS ON PAPER (not online).

Numerical Methods – HW2 2.1 – 2.5 Due Friday Jan 25

2.5 12 pts

From "real" calculus, you should know the derivative of $f(x) = x^2$ is $f'(x) = 2x$. So at $x = 300$, the *exact* value of the derivative $f'(300)$ is obviously 600.

However, computers *approximate* the derivative by taking the limit $f'(x) = \lim_{a \to 0} \frac{f(x+a) - f(x)}{a}$. So applying this to the $f(x) = x^2$ example above, I would hope that $f'(300)$ would be well-approximated by the limit $f'(300) = 600 = \lim_{a \to 0} \frac{(300+a)^2 - (300)^2}{a}$

Let's test this out using MATLAB for smaller-and-smaller values *a*, and look at the errors! Specifically, we'll look at the right-hand-side of the limit using a range of *a* from 1 to 10^{-18}, and compare that *approximate* value of the derivative to the *exact* value of $f'(300) = 600$.

(A) First, make the exact MATLAB function **deriv.m** below to evaluate the approximate limit for any *a*:

```
function fprime = deriv(a)
    x = 300;
    fprime = ((x+a).^2 - x^2) ./ a;
end
```

Do **not** simplify or change the formula for `deriv(a)` in any way! The equation is important so the mathematical operations are done the same way for everyone in the course. If you change it, you may not see the numerical phenomena below, of which your explanation is being graded.

(B) Make your own script, called **HW2_5.m**, that uses deriv.m to calculate vectors of the following four parameters using the 18 values of *a* in the vector [1, 0.1, 10^{-2}, 10^{-3}, … , 10^{-18}] :

a, *deriv*(*a*), absolute value of "error", absolute value of "% relative error"

For example, you might call the four vectors `a`, `fprime`, `Error` and `PctError`.

- Define "error" here as the difference between *deriv*(*a*) and the exact value of 600.
- Your error *must* be only positive, because soon you'll take it's logarithm. Try using the built-in command `abs( )` to take the absolute value of elements in a vector.

(C) Add to the end of your script, the following code fragment to make two plots in your figure window: one of the approximate derivative, and one of the $\log_{10}$ of the percent relative error, both as a function of the *base-10* logarithm of *a*:

```
subplot(211); plot(log10(a), fprime, '-o')
  xlabel('log_1_0(a)'); ylabel('Approx Derivative')

subplot(212); plot(log10(n), log10(PctError),'-o')
  xlabel('log_1_0(a)'); ylabel('log_1_0(|% Error|)')
```

Are any of these plotting commands new for you?

- `subplot(211)` and `subplot(212)` allow you to put *two* plots in the same figure.
- `log10(x)` returns the base-10 logarithm of *x*. Don't use `log(x)`, that's used instead for the *natural* log (*i.e. ln*(*x*)).
- The *underscores* ('_') in the x- and y- labels tell MATLAB to write the next letter as a subscript font, so your label looks very professional as $\log_{10}(a)$.

2.5 continued …

(D) Save your figure (with both plots in it) as a single pdf file called **PLOT2_5.pdf**. I'd use the command `print('-dpdf','PLOT2_5.pdf')` but do whatever works for you.

That's it! You're done the MATLAB work. Now you're ready to *interpret* the results.

You would hope that your approximation deriv(*a*) would just keep getting closer-and-closer to the exact value of 600 as *a* gets smaller. That's what "real" calculus would say is true. What you *should* be seeing is this trend working initially (good!), but then for small-enough *a* the approximation starts getting worse (bad!), and then once *a* hits a critical value of 10^{-14} things go *really* bad and you suddenly get a 100% error ($\log_{10}$(% error) = 2) !!! Please see the TA if you don't see that trend. You need to make sure you did your MATLAB work correctly so we're all starting from the same values and plot to answer these questions.

Problem (a): What is the smallest value of ***percent error***, and the corresponding ("optimum") value of ***a***? Enter these values directly in the *comment box* on Carmen.

So smaller *a* sometimes helps, but not always. Does this make sense? Recall that <u>total</u> numerical error comes from <u>both</u> *Finite Precision Algebra* and *Series Truncation* (algorithm) errors. Work through the remaining questions to understand and explain the error trend better:

Problem (b): (A-type "error" in the approximate derivative due to just *Finite Precision Algebra*)

(i) Describe how you expect only this source of error to behave as *a* gets smaller. Be sure to use **quantitative reasoning** (*i.e.* don't just guess) to justify/explain your answer.

(ii) Which region of the plot (as a range of *a*) do you think the total error is being dominated by *this* (finite precision) source of error?

Problem (c): (Why does "error" suddenly jump to 100% for *a* at and below 10^{-14}?)

(i) Here's the most important part of the homework: think carefully and explain to me ("quantitatively justify") exactly why it turned out to be *that* value of *a* specifically. In other words, based on the equation we're using for the derivative approximation, and your understanding of how MATLAB calculates it, how could you have predicted the error would "blow-up" at <u>*that*</u> specific value of $a = 10^{-14}$?

Problem (d): (B-type "error" in the approximate derivative due to just *Series Truncation*)

(i) Describe how you expect only this source of error to behave as *a* gets smaller. Be sure to use **quantitative reasoning** to justify/explain your answer.

(ii) Which region of the plot (range of *a*) do you think is being dominated by *this* source of error?

(iii) Look at the slope of the $\log_{10}$(*Error*) vs. $\log_{10}$(*a*) plot in the region you identified in (d)(ii). Use that slope to compute the *order* of the error (*i.e. "p"*, as defined in problem 2.2).

(iv) Finally: using just the equations in *deriv*(*a*) *show* me how you could have analytically predicted <u>*that*</u> value of error order *p* for this problem, before even making the plot. Hint: similar to part (c)(i), think about how MATLAB is calculating *deriv*(*a*) as *a* gets smaller

Numerical Methods – HW2 2.1 – 2.5 Due Friday Jan 25

2.5 continued …

Why I love this homework: half is you trying a new process and making *observations* about the "error", and the other half is you *analytically justifying* why those error trends make sense for this particular case.

Please submit the following:

- ONLINE: **4** things! Your plot **`PLOT2_5.pdf`**, documented script **`HW2_5.m`**, and two numbers in the comment section: the minimum **`Error`** and corresponding value of **`a`** from problem **(a)**.
- ON PAPER: Answers, discussions and justifications/proofs from problems **(b)**, **(c)** and **(d)**.

Here's the catch: The ON PAPER part **MUST USE A SEPARATE PIECE OF PAPER** from HW 2.1 – 2.4. I'll collect your work in class for 2.5 in a separate pile from your stapled 2.1 – 2.4 (goes to a different grader).