Math 56 Compu & Expt Math, Spring 2014: HW2 debrief

April 15, 2014

Please indicate who you work together on homework. Remember the write-up should be in your own words (ie own LaTeX). Careful explaining: "reduce the accuracy" is opposite sense from "reduce the error"—try to be unambiguous.

1. 2+2+3+2+1 = 10 pts Some of you lost points for forgetting that we want *relative* error throughout this question (after all, the naive implementation already has small absolute error)!

   (a) You didn't need full ε analysis here, just to realise how $1 + 10^{-16}$ is rounded down to 1 even though 10^{-16} was stored relatively accurately. Classic catastrophic cancellation. The relative error of something that comes out zero but shouldn't be is always 1.

   (b) Please if you used wikipedia and found Newton's generalized binomial, which is one name for what you're using, then explain *what* the symbols you use mean, eg $\binom{\alpha}{k}$, $(-r)^k$, etc. Best not to use these fancy symbols at all. `Sage` can help you check your answer.

   (c) See Matt or Pawan's nice analysis. However, if you explained which were dominant (the addition and the sqrt caused the two ε's which dominate), fine too. Note $\sqrt{1+\varepsilon} \approx 1 + \varepsilon/2$. Result is $3\varepsilon_{\text{mach}}/x^2$ once the first-order approx to the true answer is used. It you didn't include one of those epsilons, you were underestimating the error by a factor 3/2, so lost a point.

   (d) Pawan had a nice argument to deduce the number of terms is three (x^8 is the remainder term).

   (e) Must quote *relative* difference (abs diff is misleadingly small). Eg see Eli who made a python code that compares and does the switch-over. Relative error is around 2e-12.

2. 3+2+4 = 9 pts

   (a) See eg Michael for showing the three-fold symmetry.

   (b) Again, geometry.

   (c) Effort for coding here. Don't forget to do enough iterations that all pts in the grid converge—this is at least 10 even at low resolution. Also, your code is faster if you *vectorize*, i.e. carry out the Newton step formula on all points in the grid simultaneously. Eg if f and f' are defined funcs that accept vector or array inputs:

   ```
   u = linspace(-2,2,1000);
   [U V] = meshgrid(u,u); Z = U + 1i*V;
   for i=1:20, Z = Z - f(Z)./fp(Z); end
   imagesc(u,u, angle(Z)); axis equal;
   ```

   A neat way to convert the complex z values into a real number indicating which root they were nearest, was to use `angle`, eg see Eli, who also did very high resolution and nice zoom into a point other than the origin.

3. 2+2+2+2 = 8 pts

   (a) Eg see James. Can estimate well by $10^3\sqrt{2}/\pi$ (why? since $\sin^{-1}$ is close to $\pi/2$ and use binomial approx on 0.999999^2).

(b) Surprisingly, κ only large for $x \approx 1$, since as the graph crosses zero, relative errors are v sensitive to x. Large and small x are fine.

(c) As Michael showed, it's best to simplify the formula for roots vs c. Then you think of this as your $f(c)$ to put in the κ formula. So, finding nearby roots of a polynomial is ill-conditioned, and κ is about the inverse of the root separation. This is true for roots of more general functions.

(d) Eg see Michael. This is really now best seen as an application of the error theorem from lecture 6.

4. (Dan's question) 4+1+1+1 = 7 pts. Most of you had good proofs and plots. One or two points were taken off if you did not comment enough on the plot of the error versus h . Full credit was given to students who commented on the slope of the plot for $10^{-4} \leq h \leq 1$ (should be about 2) and gave an explanation for the growth in the error for $h < 10^{-4}$. For a really good discussion of the error plot see Matthew Jin's solution.

   [Alex adds: notice how important your practical skill of estimation of slope of log-log plot is here].

5. 3+2+3 = 8 pts

   (a) Lesson: never compare floating-point numbers to see if they are equal! (possible exception is if they haven't been changed from being set to a definite number like zero).

   Maybe suggestions to fix, but this most accurate is to make integer loop and compute x from it: `for i=0:10, x = i/10; end`

   This is also better since x itself is more accurate each time (no accumulation of round-off errors).

   The following is dodgy and I don't recommend it (sorry Pawan, James, etc):

   `x=0; while abs(1-x)>1e-15, x = x + 0.1; end`

   The reason is it relies on detailed properties of rounding error. Integers are completely robust but this is not (eg if change to 100 steps would have to bump up 1e-15 to 1e-14, yuk).

   (b) Note "error" here just means difference between left and right sides. It's very close! (according to wikipedia, this example of a near-Fermat-breaker was constructed by the show's writer David Cohen!)

   Number of binary digits is `ceil(12*log(1922)/log(2))`, giving 131.

   (c) See Michael for detailed flop count. All got this; basic idea is flops around $4n^2$ vs $2n^3$. Some mentioned less flops causes less arithmetic error - this would be good to research (I don't know!)