Math 56 Compu & Expt Math, Spring 2013: HW1 debrief

April 6, 2014

Some of you will notice that I give bonuses for going beyond the usual in any question, not just the stated bonus ones. Efforts rewarded.

Please tell me who you worked with; eg see Matt's Attribution.

Matlab notes:

• To input base-ten exponent notation use e.g. `5e16` or `2.2e-16`.

• You will not see full accuracy printed unless you do something like `format long g`

• function filenames should match the function name, and useful documentation should come up when we type `help` and the function name (place this in comments at the top of the file).

1. 2+3 = 5 pts

   (a) Since adding 10 to the denominator decreases the overall quantity, the proof is easy here. See e.g. Matthew Jin. But what if the sign on 10 were changed?

   (b) Firstly, which is bigger as $N \to \infty$? N^2 grows faster than $O(N \log N)$, so the latter is better for large N. The transition point (which is needed only roughly in practice) can be solved by setting the two equal, giving a transcendental equation. You can use your later bisection code, or:

   ```
   fzero( @(x) x^2 - 1e6*x*log(x), 1e7)
   ```

   Note log is natural log by default. A couple of you found another solution at $N \approx 1$, but keep in mind for algorithms, large N is all that matters.

2. 3+1+1+1 = 6pts. [Dan Cianci graded].

   Most of you had good data and plots. One or two points were taken off depending on how much detail you could give for the relationship between the maximum eigenvalue and n. Almost everyone observed a relationship of $\lambda_{max} = O(n^{\frac{1}{2}})$, but only a few of you used the log-log plot to show that $\lambda_{max} = 2.8n^{\frac{1}{2}}$ (roughly). Also some of you did not compute an average over several random matrices of fixed dimension n. For an exemplary solution, see Jon King.

3. 3+1+3+2 = 9 pts.

   (a) Almost all of you did good data and plots. Some measured a strange slope: estimate it by eye so you think about what you get by rise/run on the log of the data. The slope is approximate, so only give it to 3 digits, say.

   (b) A few did a linear axes plot to see what happens; e.g. see Max's nice comparison. It "crushes" data against the two axes, and tells you nothing about the size of small errors.

   (c) A proof is asked for, not a reiteration or claim based on *experimental* observations from (a) ! Several of you forgot to flip to result to give $n = O(\varepsilon^{-1/3})$.

   (d) You discover (unless you use sage's sum command which is fancy) that forwards only gives 13 digits, backwards 16 digits. Eg see Michael who demonstrated this very cleanly.

4. 4 for working code (with documentation) +1+3 = 8 pts.

   Documentation best if includes the calling command and description of inputs & outputs, and pops up when you do `help` for your function. James had good documentation. Pawan had a model test

script: it checks if the root accuracy is at least as good as requested. This is important. Eli's was all in python (you're welcome to use). Recursion is not necessary; a simple "while loop" is enough.

Question: should input tolerance on the root be interpreted as absolute or relative? (most of you did absolute)

(a) Some of your codes just went ahead even when $f(a) = 0$; as requested it should exit in this case (returning a maybe!) since it can't guarantee that the root is on the correct side. Detecting bad signs should be done using the *sign* of f at a, b, c, not merely that f is less than some tolerance. The tolerance is on the x of the root, not on the size of f. We don't know the "typical size" of the user's supplied f. In other words, scaling f by a small (or large) number should have no effect on relative accuracy of f and hence ability to find roots. The following should run in the same way:

```
bisection(@(x) sin(x),3,4,1e-15)
```

```
bisection(@(x) 1e-20*sin(x),3,4,1e-15)
```

(b) Demanding tolerance of `1e-20` for the root π crashed or made most of your codes run infinitely long; Jon has a good explanation of this. This is generally bad, and you need to think about how to detect such issues if you are to release software.

BONUS Advantages:

i) doesn't need derivatives of f

ii) guaranteed to work if there's a root (whereas Newton can head off somewhere bad if $|f'|$ too small). Thanks Max and Pawan for this 2nd one.

5. 3+2+2+2 = 9 pts

(a) All good.

(b) see many of your codes. Don't be scared of increasing the resolution.

(c) Don't forget elementwise operation, e.g.

```
f = @(x) 1./(1+x.^2)
```

otherwise you'll get garbage (some weird matrix-matrix producting).

By *phase* I mean the angle of a complex number (see lecture). You can extract this via `angle(z)` in matlab. At each pole the phase climbs smoothly by 2π once per clockwise revolution. Eg see Matthew's plot. Note it appears to "jump" from π to $-\pi$, but this is just an artifact of the angle returned being single-valued. Really there is no jump (just like an analog clock face doesn't "jump" from 12 back down to 1 ... angle changes smoothly with time). Michael explains this.

(d) Caused trouble. Dist of nearest sing from expansion pt is $R = \sqrt{2}$, and dist of x from expansion pt is 0.7. Rate is therefore their ratio $r = 0.7/\sqrt{2} \approx 0.49 < 1$, so exponential convergence.

6. 4 pts. Arguing using the 1-page summary of floating point, in general it's $\beta^{t+1} + 1$, since all smaller integers are "hit" exactly by a member of F.

`(2^53 + 1) - 2^53` gives zero, showing that the +1 is lost in rounding the bracketed expression to a floating-point number. But `(2^53 - 1) - 2^53` gives -1, showing that $2^{53} - 1$ is correctly represented. Notice that a complete answer really should demonstrate that the next smaller integer is correctly represented (eg Aron did this).

7. 3 pts. The power $1/2^{60}$ is so ridiculously small that it reduces most numbers to a number which rounds to 1 in floating-point, and the returned answer is 1 not x. As Michael and Eli explain, something can survive for x bigger than around 1.52×10^{112}, in which case it rounds to $1 + 2^{-52} = 1 + 2\varepsilon_{\text{mach}}$. If you never found these large x for which the returned answer is not 1, you got 2/3.