Math 56 Compu & Expt Math, Spring 2014: HW6 Debriefing

1. [Dan] 5+4 = 9 pts

   (a) You just computed a number with over one million digits in a second or two! The joy (and utility!) of FFTs.

   A crucial step is to `round` the real-valued result from the FFT to integers before the carrying is done, otherwise when the test of whether to carry is done, eg 9.9999999 is treated as 9 not as 10. See Michael's solution for a nice implementation.

   (b) For the time estimate of the naive method (mult by 2, repeat 2^{22} times), each mult is $O(D)$ not $O(D \log D)$ since it's mult by a small $O(1)$ number, where D = number of digits in answer. Overall is $O(D^2) = 2^{44} \approx 10^{13}$ flops, about half an hour of computer run time.

   BONUS See Eli's solution for proof of the periodicity of the last two digits (and also a nice use of strong induction).

2. 4+2+2+3 = 11 pts. Some of the early points for getting into python.

   (a) As several found, if the answer is x, initial guesses must be in $(0, 2x)$ otherwise the iteration blows up to infinity. Pawan proves this.

   (b) The repeating string is 96 long:
   010309278350515463917525773195876288659793814432989690721649484536082474226804123711340206185567
   Hanh explains this in terms of Fermat's Little Theorem. Also see
   `http://en.wikipedia.org/wiki/Repeating_decimal`

   (c) $O(N \log^2 N)$, see eg John.

   (d) As you found, the errors hardly differ; both are of order $\varepsilon_{\text{mach}}$. I get that the first way has $3\varepsilon_{\text{mach}}$ relative error and the second way $2\varepsilon_{\text{mach}}$. None of you quite got this second case, although you all got similar $O(\varepsilon_{\text{mach}})$. There is certainly no difference where one is $O(\varepsilon_{\text{mach}})$ but the other $O(\varepsilon_{\text{mach}}^{1/2})$, as wikipedia suggests. Here wikipedia is wrong! (Please, someone correct it; bring it up on discussion page first.)

3. 3+4+2+4 = 13 pts.

   Throughout this question it was important to *check convergence* to the required accuracy! One way to guarantee this is a `while` loop that only stops when the answer doesn't change to the required precision. Another (harder) is to precompute the number of terms using a convergence rate or estimate.

   (a) The definition of the number of "terms used" is a bit ambiguous, since it could mean "what is the highest poiwer n of x used in the Taylor series?" By that definition, as Kunyi calculates, if $\tan^{-1} 1/5$ is the largest number to approximate by a Taylor series, around $n = 14500$ is needed. But this involves only $n/2$ nonzero terms, i.e. around 7300 "terms." Either definition I treated as correct.

   Speed: if you compute the power $y = x^{2k+1}$ from scratch in each term, it will be slow. It's *much* faster to precompute $z = x^2$ and $y = x$ then update $y = zy$ each time around the loop. Those of you coming to office hours (eg Eli) learned this.

   (b) As many of you discovered, online tools allow you to check digits of pi, or, better, sage or python/mpmath can do it efficiently via `mp.dps = 10000; print str(pi)[-10:]` which prints digits 9991 to 10000. Note the python array indexing (equivalent to `(end-9:end)` in Matlab). We believe mpmath uses Brent–Salamin. If you rounded the 10000th digit from 7 to 8, this was fine. Careful with defining modules with names like `pi`, etc. This overwrites the constant `pi` that mpmath has!

(c) Difference of two squares; review of quadratic convergence proofs. Hanh shows that since the iterations x_n, y_n always lie between the original values, you can use $\min(x_0, y_0)$ to bound the const. Proving the convergence of α_n would be extra; see Salamin's original 1976 paper.

(d) You all found Brent–Salamin around 10^2 to 10^4 times faster than Taylor series even at $N = 10^4$ digits. Imagine how much faster it is at $N = 10^6$. Your algorithms were close to mpmath's in speed for evaluating π, ie one million digits in a couple of seconds! See eg Aron or Jon for the digits. If you started at 999999th or 999998th that was fine (remember python arrays are 0-indexed)