Math 56 Compu & Expt Math, Spring 2013: HW5 Debriefing

May 5, 2014

1. (Dan) 2+3 = 5 pts

   (a) Most of you got this one. Something to keep in mind here: if you can express $\cos(6x)$ as a sum exponential functions then that sum *must* be the Fourier series since the expression is unique. Otherwise you have to use the formula for the coefficients.

   (b) Aliasing occurs. See Matthew's solution for an explanation.

2. 3+3 = 6 pts

   (a) Notice the power of what you've done: from only 30 samples of the function, you've approximated its derivative for all x to machine precision! This is like a super finite-difference formula ("high-order method") that is much better than a standard finite-difference formula. Your theorem from HW4 #1e proves error is super-algebraic. In fact as Max observes, it's super-exponential: $f(x) = e^{\sin x}$ is entire, so f' is too, but it's beyond this course to explain the connection.

   (b) The observed convergence in error is 2nd-order algebraic, see eg Matt. The analysis is a bit more subtle. I gave points for clear thinking and applying what we know from class. $f(x) = |\sin^3 x| \in C^2$ but $\notin C^3$. However f''' exists and is bounded, so the theorem from HW4 #1e says that $\hat{f}_n = o(n^{-3})$. However we're really interpolating $g = f'$, so $\hat{g}_n = in\hat{f}_n = o(n^{-2})$. However, the error due to interpolation depends on the *sum* of the tail of the Fourier series: $E_N \leq 2\sum_{|n|\geq N/2} |\hat{g}_n|^2$. (This wasn't done in class until Thurs 5/1/14, but you had all the ideas present, as Pawan realised.) So in our case you can prove $E_N = o(n^{-1})$, which is indeed consistent with (but not as good as) the observed. It turns out the theorem from HW4 is not "sharp": it can be improved to say that if $f^{(k)}$ has "bounded variation" (similar to being bounded), then $\hat{f}_n = O(1/|n|^{k+1})$, which is a whole order better than we proved by elementary means.

3. 7 pts.

   The points here are for getting the code to work (small error norm, should be at least 10^{-11}), and timing it. I tested each of your codes on a *generic, complex* vector `y = randn(1,N) + 1i*randn(1,N);` If you merely check on a real or simpler vector, you don't know it's robust.

   Pawan ran codes 10 times to reduce variations in timing. One tricky thing (that eg Michael rants in CAPS about) is that since Matlab arrays are 1-indexed, extracting the even elements looks superficially like the odd ones! Another: Matlab `'` is conjugate tranpose; to get plain transpose use `.'` . As you all found, FFTW is around 1000 times faster—a great reason to program in C, and think very hard about memory movement, cache, CPU design, etc. See wikipedia for Cooley–Tukey algorithm.

4. 4+2 = 6 pts. (Dan)

   (a) $\hat{f}_{654}$ is the complex conjugate of $\hat{f}_{-654}$, which must hold since the signal vector $\mathbf{f}$ is real-valued. (We did this in lecture—prove it as an exercise).

   (b) See the `audiofft.m` example from lecture. If m is the index of the peak coefficient, and $T = N/f_{\text{sam}}$ is the total sample time, where $N = 2^{16}$ is the sample vector length and $f_{\text{sam}} = 44100$ is the sampling freq. So the peak freq is $m/T = 440$ Hz (this is A above "middle C").

BONUS You probably cannot hear any tone above the noise (at least I cannot!) To compute amplitude, `abs(ft(654+1))/N` returns the amplitude of *one* complex exponential mode, but you need to double it since there is the negative freq mode present too (they combine to give a real-valued sine wave), to give amplitude of 0.0129. Meanwhile the mean square overall amplitude is `sqrt(mean(abs(f).^2))` which is 0.226. Thus the signal is amplitude about 5% of the overall.

5. 2+2+2+2+2 = 10 pts

   (a) Eg see James who wrote out each element's computation. Max shows the "blurring" structure.

   (b) Here the last entry is wrapped around and added to the first.

   (c) As you all discussed, $N_1 + N_2 - 1$ is sufficient.

   (d) Eli proves this by change of variable; Michael via the DFT.

   (e) See Aron for probably the best-expressed proof.

6. 6 pts.

   This was fun and easy. You needed to zero-pad out to length $N_1 + N_2 - 1$ as above. If your `wavread` gave a stereo signal, you needed to use just one of the columns of the output. If the sampling freq was not 44100, you needed to use whatever it was in the `wavwrite` command.

   Unfortunately if `wavwrite` is given data outside the range $[-1, 1]$ it will "clip" it to these maximum values, which sounds horrible. Ues `y = y/max(abs(y));` before writing out. Many of you forgot the `abs` here.

7. 2+3+2 = 7 pts

   (a) Just reading in here. Don't forget `axis equal` to view correctly.

   (b) I was not worried about zero-padding here (in fact I used the periodic convolution to blur it in the first place). If your deconvolved picture was upside-down, this is due to using `ifft2` instead of `fft2`, since recall that (in the 1D case) F^2 is N times a permutation matrix that reverses the order of the elements.

   (c) iid Gaussian white noise with unit standard deviation given by `randn(512,512)`. I chose the noise level 3×10^{-5} so that the noise nearly swamps the image after dividing by this aperture's FFT. Your explanations that dividing by small numbers amplifies noise were good.

BONUS One way to improve this (as eg Pawan realized) is just kill (set to zero) the Fourier coefficients that get divided by really small coefficients of the aperture. E.g. cut-off below 10^{-4} in the aperture coefficient magnitudes (although this is a subjective choice based on the observed noise level). The few missing data don't affect the image quality too much, and the noise is much reduced. This is called *Tikhonov regularization*, very useful in *inverse problems* like this one.