18.336 Pset 1 Solutions

Problem 1: Trig. interp. poly.

(a) When aliasing is included, the general form for the interpolated function is:

$$f(x) = \sum_{k=0}^{N-1} c_k e^{i(k+\ell_k N)x},$$

and thus the mean-square slope is:

$$\frac{1}{2\pi}\int_0^{2\pi} |f'(x)|^2 dx = \sum_{k=0}^{N-1} (k+\ell_k N)^2 |c_k|^2$$

since the integration kills all of the cross terms in the squared summation, by the usual orthogonality. Each of these terms is non-negative, and so the mean-square slope is minimized when each coefficient $(k+\ell_k N)^2$ is minimized independently. If $N = 2M+1$ (odd), then for $k \le M$ this is minimized for $\ell_k = 0$, and for $k > M$ it is minimized for $\ell_k = -1$ (since in that case $|k-N| \le M < k$). Q.E.D. (since $M = (N-1)/2$ and $M+1 = (N+1)/2$).

(b) The symmetric form from (a) can equivalently be written as

$$f(x) = \sum_{k=-M}^{M} c_k e^{ikx},$$

where $c_{-k} \equiv c_{N-k}$ according to aliasing. Since

$$c_k = \frac{1}{N}\sum_{n=0}^{N-1} f_n \omega_N^{nk},$$

then for real f_n ($f_n = f_n^*$) it immediately follows that $c_{-k} = c_k^*$, and thus

$$f(x) = c_0 + \sum_{k=1}^{M} (c_k e^{ikx} + c_k^* e^{-ikx}).$$

This is manifestly real, since $c_0 = c_0^*$ is real and each term in the sum is $2\Re[c_k e^{ikx}]$.

This is *not* a *unique* real interpolating polynomial, however. In particular, we could also choose *any* shifts ℓ_k as long as $\ell_{-k} \equiv \ell_{N-k} + 1 = -\ell_k$, as this would preserve the property that we have a sum of terms and their complex conjugates.

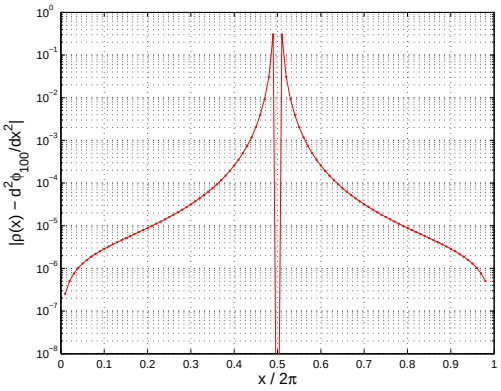

Figure 1: The absolute error $|\rho(x) - \phi_N''(x)|$ vs. $x/2\pi$, using a finite-difference approximation for the derivative ϕ_N''.

Problem 2: Solving Poisson's eq.

(a) To evaluate $\phi_N''(x)$ via finite differences in Matlab, we simply use `diff(diff(phi))/(2*pi/100)^2`. This is equivalent to:

$$\phi_N''(x) \cong \frac{\phi_N(x+\Delta x) - 2\phi_N(x) + \phi_N(x-\Delta x)}{\Delta x^2}$$

which (as we shall explore in more detail in lecture) is a second-order approximation with $O(\Delta x^2)$ error. In figure 1 we plot $|\rho(x) - \phi_N''(x)|$ versus x using this approximation. As we noted in class, there is a happy "accident" in this case—often, the peak error would be at the point of discontinuity, but in this case the error is exactly zero there, and the error elsewhere is lower than we might expect from pessimistic upper bounds. (Similarly, in Fourier space the $c_0 = 0$ coefficient is computed exactly.) Still, the peak error is $O(1)$ due to Gibb's phenomena.

(b) In figure 2, we show the approximate L_2 error Δ_N vs. N for three functions $\rho(x)$. First, the original discontinuous "sawtooth" $\rho_1(x)$. Second, $\rho_2(x) = |\sin x| - 2/\pi$ which is continuous with discontinuous slope. Third, we consider $\rho_3(x) = 2x/\pi - 1 + \cos x$ for $x < \pi$ and $\rho_3(x) = 2x/\pi - 3 - \cos x$ for $x > \pi$, which is continuous with continuous first derivative but discontinuous second derivative. After some initial bumpiness (see below), we see that *both* ρ_1 and ρ_2 give $O(1/N^2)$ convergence. The simple upper bound

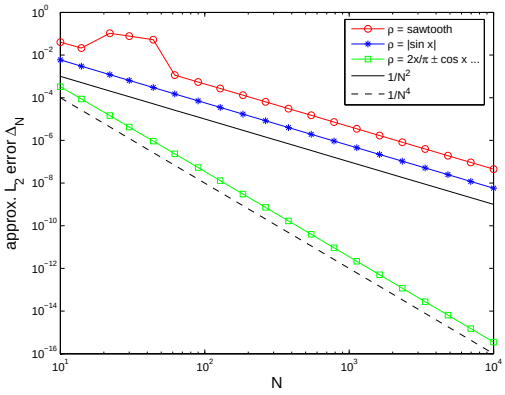

Figure 2: Approximate L_2 error Δ_N vs. N for three $\rho(x)$ with delta-function first, second, and third derivatives respectively. Also shown are $1/N^2$ and $1/N^4$ lines, for reference.

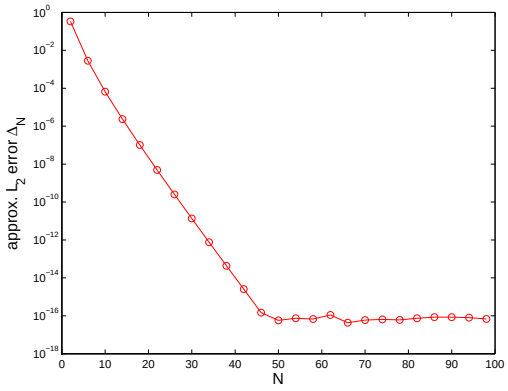

Figure 3: Approximate L_2 error Δ_N vs. N for smooth $\rho(x) = f(x) - \frac{1}{2\pi}\int f(x)dx$, where $f(x) = 1/(2+\cos x)$.

from class would be $O(1/N)$ for discontinuous ρ and $O(1/N^2)$ for discontinuous-slope ρ, but we explained in class why this ρ_1 happens to do better. The ρ_3 convergence goes as $O(1/N^4)$; the general upper bound for discontinuous–second-derivative ρ is $O(1/N^3)$, but this function happens to do better because of a similar accident as ρ_1 (we have constructed ρ_3 so that its $c_0 = 0$ is summed exactly, whereas this is *not* true for ρ_2) .

The initial "bumpiness" of the ρ_1 convergence is not random, however—it follows a straight line with $1/N$ convergence. In fact, if we computed Δ_N for more N values we would see that there are a whole sequence of N points, extending to large N, where Δ_N is $\Theta(1/N)$ instead of $\Theta(1/N^2)$. What's going on? The answer is that there is a bug in the code: at the point $x = \pi$, $\rho_1(\pi)$ should be zero, but because of rounding error in the `linspace` function the middle x value is sometimes slightly greater or less than `pi`, and for *these* values of N we get `sum(rho)` $\neq 0$. Then the c_0 Fourier coefficient is not computed exactly and the error returns to the $1/N$ upper bound. To fix this and get $O(1/N^2)$ convegence everywhere we could simply set `rho(N/2+1)=0`, however.

Finally, in figure 3, we show Δ_N for $\rho(x) = 1/(1+2\cos x) - \frac{1}{2\pi}\int_0^{2\pi} \frac{dx}{1+2\cos x}$, which is smooth on the real-x axis and should thus give geometric convergence. We therefore plot on a semilog scale, and indeed obtain the expected straight-line $e^{-\alpha N}$ behavior. At least, we get a straight line until the error reaches 10^{-16} or so, at which point it saturates—this is nothing more than the limit imposed by the finite precision of double-precision floating-point arithmetic (numbers on the computer are represented with only about 16 significant digits of precision).

Problem 3: FFTs

For radix-2 DIT Cooley-Tukey, the outputs are computed via (from class, taking $N_1 = 2$ and $N_2 = N/2$):

$$y_{\frac{N}{2}k_1+k_2} = \sum_{n_1=0}^{1} \left(\omega_N^{n_1 k_2} \sum_{n_2=0}^{\frac{N}{2}-1} x_{n_1+2n_2} \omega_{N/2}^{n_2 k_2} \right) \omega_2^{n_1 k_1}$$

where the inner sum is a size-$N/2$ DFT computed recursively, and the outer sum is a size-2 DFT.

(a) Let $T(N)$ be the number of real-arithmetic operations. Then $T(N) = 2T(N/2) + \Theta(N) + (N/2)T(2)$, where the $\Theta(N)$ represents the multiplications by the twiddle factors $\omega_N^{n_1 k_2}$. To get $T(N)$ more explicitly, we need to count the number of operations in $T(2)$ and the twiddle multiplications.

$T(2)$ is easy since $\omega_2 = e^{-\pi i} = -1$: a DFT of length 2 is just an addition $y_0 = x_0 + x_1$ and

a subtraction $y_1 = x_0 - x_1$, and since these are complex-number additions we get $T(2) = 4$.

Naively, we might think that the $\omega_N^{n_1 k_2}$ multiplications require N complex multiplications, but many of these are multiplications by 1 and hence have no cost. In particular, for $n_1 = 0$ they are multiplications by 1 so we only need to consider the $N/2$ multiplications for $n_1 = 1$. Some of these also simplify: for $k_2 = 0$ it is 1, and for $k_2 = N/4$ it is $-i$ which is also cost-free, and if you *really* care about counting operations you also must note that for $k_2 = N/4 \pm N/8$ you get $-(i \pm 1)/\sqrt{2}$ which requires only 2 real multiplications and 2 real additions. However, the important point is that the number of these special cases is bounded (at most four of them), and so we know that the twiddle multiplications require $6N/2 - O(1)$ real-arithmetic operations (since each general complex multiply takes 6 operations). Thus, we now know that

$$\begin{aligned} T(N) &= 2T(N/2) + 5N - O(1) \\ &= 2[2T(N/4) + 5N/2 - O(1)] + 5N - O(1) \\ &= \cdots \end{aligned}$$

Now we are almost done. If we repeat this recursively down to $N = 2$, there are $\log_2 N - 1$ stages. From each stage we pick up $5N$ operations (N halves at each stage but the number of transforms doubles), which gives us $5N(\log_2 N - 1)$ operations. From each stage we also pick up $O(1)$ operations multiplied by $1, 2, 4, \cdots$ but the sum of this geometric series is $\sim 2^{\log_2 N} = N$ and thus this term is at most $O(N)$. Thus,

$$T(N) = 5N \log_2 N - O(N)$$

and $\# = 5$.[1]

(b) Let $T_r(N)$ denote the number of real-arithmetic operations required for the same algorithm specialized to the case of real inputs. Then the Cooley-Tukey algorithm breaks the DFT of length N into two DFTs of length $N/2$, both of which again have *real* inputs and thus $T_r(N) = 2T_r(N/2)+??$. Denote the results of these two real-input sub-DFTs as e_{k_2} and o_{k_2} (the transforms of the even- and odd-indexed x_n, respectively). Note that $e_{N/2-k_2} = e_{k_2}^*$ by the usual conjugate symmetry, and similarly for o_{k_2}. In order to combine the outputs of these two sub-DFTs, we must do:

$$y_k = e_k + \omega_N^k o_k$$

for $0 \le k < N/2$, and $y_{N/2+k} = e_k - \omega_N^k o_k$ for the other half—this is the DFT of size 2 (the n_1 sum). However, since $y_{N-k} = y_k^*$, we need only compute half of these outputs ($0 \le k \le N/2$). Right away, this saves us half of the $4N/2$ additions that were previously required for the $T(2)$ transforms. However, it still seems like we are multiplying by $N/2$ twiddle factors ω_N^k—we need to save half of these somehow.

We can save half of the twiddle multiplies by using the conjugate symmetry of e_k and o_k, to get $y_{N/2-k}$ for $k < N/4$. In particular:

$$y_{N/2-k} = e_{N/2-k} + \omega_N^{N/2-k} o_{N/2-k} = e_k^* - (\omega_N^k o_k)^*,$$

where we have used the fact that $\omega_N^{N/2-k} = -\omega_N^{-k}$. Therefore, we only need to compute $\omega_N^k o_k$ for $k \le N/4$ and obtain the $N/4 < k < N/2$ values by conjugation (which is only a sign flip and requires no multiplies or additions). Thus, we have saved roughly half of the $5N$ operations that were previously required to combine the sub-transforms:

$$T_r(N) = 2T_r(N/2) + \frac{5}{2}N + O(1)$$

where $O(1)$ is again some $\pm$ bounded value that takes into account a bounded number of extra optimizations and special cases: the few twiddle factors that simplify as before such as $k = N/4$ and $k = N/8$, the $k_1 = 1$, $k_2 = 0$ term for $y_{N/2}$, and the $k_1 = k_2 = 0$ term which is purely real.

Then, by the same arguments as above, $T_r(N) = \frac{5}{2} N \log_2 N + O(N)$ and we asymptotically save half the operations. Q.E.D.

[1] In 1968, Yavne's "split-radix" algorithm achieved $T(N) = 4N \log_2 N + \Theta(N)$, which was the best count for $N = 2^m$ for 36 years. The current best count for $N = 2^m$, achieved in 2004, is $T(N) = \frac{34}{9} N \log_2 N + \Theta(N)$.