MAT 271: Applied & Computational Harmonic Analysis Comments on Homework 2

Problem 1: This was an easy problem. Most of you answered correctly. But you have to be careful. $\exp(2\pi\mathrm{i}(k-\ell)/N)$ could be 1 if $k-\ell$ is an integer multiple of N. I implicitly assumed that $0 \le k, \ell, \le N-1$ here, so this will not happen. If you explicitly say $\exp(2\pi\mathrm{i}(k-\ell)/N) \neq 1$ for all integers k, ℓ, it is incorrect.

Problems 2–3: **(a)** A few people didn't simplify $\cos(\pi k)$ as $(-1)^k$ and $\sin(\pi k)$ as 0. Since k is an integer, you should use $(-1)^k$. By the same token, $\sin(\pi k) = 0$, of course. Also a few people did not treat the case of $k = 0$. c_0 carries very important information, i.e., the so-called DC component of an input function. Thus, do not forget to compute c_0.

(b) The point of this problem is to figure out the difference between the hand-derived Fourier series coefficients and the the DFT coefficients computed via the FFT function of MATLAB. Many people plotted and compared the absolute values of the FFT coefficients and hand-computed Fourier coefficients. That only gives you a part of the story. You really need to plot and compare the real part and imaginary part separately without taking the absolute values in order to see the real difference between the hand-computed Fourier coefficients and the output of the MATLAB `fft` function!

First of all, you need to go back to the original definition of the DFT and the Fourier coefficients.

$$\begin{aligned}
c_k &= \int_{-\frac{1}{2}}^{\frac{1}{2}} f(x)\mathrm{e}^{-2\pi\mathrm{i}kx}\,\mathrm{d}x \\
&\approx \sum_{\ell=0}^{N-1} f(x_\ell)\mathrm{e}^{-2\pi\mathrm{i}kx_\ell}\Delta x, \quad x_\ell = -\frac{1}{2} + \ell\Delta x \text{ and } \Delta x = \frac{1}{N} \\
&= \frac{1}{N}\sum_{\ell=0}^{N-1} f\left(-\frac{1}{2}+\frac{\ell}{N}\right)\mathrm{e}^{-2\pi\mathrm{i}k(-1/2+\ell/N)} \\
&= \frac{\mathrm{e}^{-\pi\mathrm{i}k}}{N}\sum_{\ell=0}^{N-1} f\left(-\frac{1}{2}+\frac{\ell}{N}\right)\mathrm{e}^{-2\pi\mathrm{i}k\ell/N} \\
&= \frac{(-1)^k}{N}\sum_{\ell=0}^{N-1} f\left(-\frac{1}{2}+\frac{\ell}{N}\right)\mathrm{e}^{-2\pi\mathrm{i}k\ell/N} \\
&= \frac{(-1)^k}{N}\sum_{\ell=0}^{N-1} f_\ell\,\mathrm{e}^{-2\pi\mathrm{i}k\ell/N}.
\end{aligned}$$

And finally, the summation portion can be computed by `fft` (non-unitary original version) in MATLAB. Here is my MATLAB script for Problem 2 if we want to match the hand-computed Fourier coefficients and the MATLAB `fft` output as much as possible. I also put my codes online, so please download them and run them to see how much these two sets of coefficients agree.

```
% Problem 2

% define the basic parameters.
N=1024;
a=-0.5;
b=0.5;

% Create an array of N equidistant points over [a,b].
% Trying to exclude the point x=b=0.5 from the samples.
x=linspace(a,b,N+1);
x=x(1:end-1);

% Create a function.
y = x; % In problem 3, this should be y=x.^2, of course.

% Normalize the function to have a unit L^2 norm.
ey = norm(y);
y = y/ey;

% Do the fft to approximate the Fourier series coefficients over this
% interval.  Note that we need to have 1/N here.  You need to go back
% to the original definition of the Fourier coefficients and its
% approximation by the trapezoidal rule.
% Note that fft essentially view the input data is defined over the
% interval on [0,1], instead of [-1/2,1/2].  So you need to do either
% of the following two:
% 1) Apply fftshift to the input vector before taking fft; or
% 2) Apply the complex exponential factor exp(pi*k)=(-1)^k to the output
%   of fft, which is equivalent to changing the signum of the fft results
%   alternatively as fy(2:2:N)=-fy(2:2:N), where fy=fft(y)/N.

fy = fft(y)/N;
fy(2:2:N)=-fy(2:2:N);

% Now, prepare the analytical Fourier coefficients you derived by
% hand.

c = zeros(1,N);
% c(1) = 1/12.0; % for problem 3.
for k=1:N-1
  c(k+1)=i*(-1)^k/(2*pi*k); % c(k+1)=(-1)^k/(2*(pi*k)^2); % for problem 3.
```

```
end


% Normalize the coefficients
c = c/ey;

% Now plot real and imaginary part separately using the semilog plot.

figure(1)
clf;
subplot(1,2,1);
plot(real(c(1:N/2)));
grid
hold on
plot(real(fy(1:N/2)),'r.');
title('Real Part')
hold off

subplot(1,2,2);
plot(imag(c(1:N/2)));
grid
hold on
plot(imag(fy(1:N/2)),'r.');
title('Imaginary Part')
hold off

% Let's look at the more details around the origin.
figure(2)
clf;
subplot(1,2,1);
stem(real(c(1:N/16)),'o');
grid
hold on
stem(real(fy(1:N/16)),'r.');
title('Real Part')
hold off

subplot(1,2,2);
stem(imag(c(1:N/16)),'o');
grid
hold on
stem(imag(fy(1:N/16)),'r.');
title('Imaginary Part')
hold off
```

Do the similar computation for Problem 3. You can see that they match closely, but not exact due to the approximation error by the trapezoidal rule and sampling. What happens if we increase the number of samples, e.g., to $N = 2^{15}$?

Problem 4: 1)–3) Several people use the normal distribution factor as the parameter a, i.e., $a = 1/(\sigma\sqrt{2\pi})$. But my intention was to use a as the normalization constant so that the ℓ^2 norm of the input vector becomes 1. Once you do that, then you can follow the same strategy here as above.

4) You got mixed results. In fact, it is true that the larger the value of σ (i.e., the wider the Gaussian is), the faster the the decay of its Fourier *transform* because of it is proportional to $\exp(-2\pi\sigma^2\xi^2)$ in the Fourier domain. But unfortunately, I am asking the decay of the Fourier *coefficients* of the Gaussian on the *finite* interval $[-1/2, 1/2)$. So, the boundary effects at $x = \pm 1/2$ becomes more prominent compared to the smoothness. The Fourier coefficient magnitudes follow more like $\exp(-2\pi\sigma^2\xi^2)$ in the low frequency region. But then, the boundary effects start dominating. This fact was obscured if you use the wrong normalization a.

5) It seems that the decay of the Fourier coefficients of the Gaussian functions are faster than that of the polynomials such as ax or ax^2, which is the case in the low frequency region. However, the periodized Gaussian over the interval $[-1/2, 1/2)$ is not a C^∞ function. It's simply a continuous function, not even C^1 function because the derivatives do not match at the boundary. Of course this derivative mismatch is less pronounced for small values of σ or extremely large σ. Hence, in the finite length DFT, the quadratic polynomial behaves similarly to the Gaussian with appropriate value of σ for the higher frequency part. Thus, the decay of the Fourier coefficients of ax^2 is slower than those of the Gaussian, but the decay curve in the high frequency range looks similar to that of the Gaussian with $\sigma = 1$.

6) The Fourier transform of the Gaussian in this case is the following using Problem 3 of HW #1:

$$\mathcal{F}\{a\mathrm{e}^{-x^2/(2\sigma^2)}\} = \mathcal{F}\{a\sqrt{2\pi}\sigma g(x;\sigma)\} = a\sqrt{2\pi}\sigma \int_{-\infty}^{\infty} g(x;\sigma)\mathrm{e}^{-2\pi\mathrm{i}\xi x}\,\mathrm{d}x = a\sqrt{2\pi}\sigma\mathrm{e}^{-2\pi^2\sigma^2\xi^2}.$$

On the other hand, using the MATLAB `fft` function, we can only approximate the Fourier coefficients of the periodized Gaussian (or mutilated Gaussian) on $[-1/2, 1/2)$:

$$c_k = \int_{-\frac{1}{2}}^{\frac{1}{2}} g(x;\sigma)\mathrm{e}^{-2\pi\mathrm{i}kx}\,\mathrm{d}x.$$

Actual output is even different from this c_k due to the error by the trapezoidal rule used to obtain FFT/DFT. Therefore, there are two errors involved here: 1) Truncation of the interval; and 2) error due to the trapezoidal rule. For more details, I strongly recommend to read [1, Chap. 6].

Problem 5: (a) Please read the problem carefully. $D_N \boldsymbol{f}$ represents the FFT operation in MATLAB `fft(f)`. Hence,

$$D_N = \sqrt{N} W_N^*$$

Note that $W_N^* \boldsymbol{f}$ is the unitary (i.e., normalized) version of the DFT.

(b), (d) As I mentioned in the class on Feb. 10, my old DFT note contained a wrong definition of the vector $\widetilde{\boldsymbol{w}}_N^k$. Hence, if you assume this wrong definition and my old DFT note, you should get the solution easily as follows:

$$\widetilde{W}_N^* \boldsymbol{f} = T_N^* S_N D_N R_N \boldsymbol{f} / \sqrt{N}.$$

If you follow my revised DFT note, it should be

$$\widetilde{W}_N^* \boldsymbol{f} = T_N^* S_N D_N S_N T_N \boldsymbol{f} / \sqrt{N}.$$

In either case, note that $T_N^* = T_N^T = T_N^{-1}$ whereas $S_N^* = S_N^T = S_N^{-1} = S_N$.

References

[1] W. L. Briggs and V. E. Henson, *The DFT: An Owner's Manual for the Discrete Fourier Transform*, SIAM, Philadelphia, PA, 1995.