The Discrete Fourier transform

The objective of this lab is to explore how the discrete Fourier transform reveals the frequencies present in a signal. We will also see how the frequency content of a signal can be manipulated using the Fourier transform in order to change the signal in some desired way, for example, to remove noise or apply a high-pass or low-pass filter.

For numerical signal processing, we need to use the discrete Fourier transform, which is a bit different from the Fourier coefficient formula for $L^2(\mathbb{T})$ functions. The definition of the discrete Fourier transform for a discrete signal $\{x(0), x(1), \ldots, x(N-1)\} \in \mathbb{R}^N$ or $\mathbb{C}^N$ is

$$\hat{x}(k) = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x(n) e^{-2\pi i k n/N} \quad \text{for } k = 0, 1, \ldots, N-1. \tag{1}$$

This yields a new sequence $\{\hat{x}(0), \hat{x}(1), \ldots, \hat{x}(N-1)\} \in \mathbb{C}^N$ (which we often regard as extended to an N-periodic sequence), where $\hat{x}(k)$ is the amplitude associated with frequency k. Note that $2\pi n/N$ here represents a discrete value of time t, as though we sampled uniformly on the interval $[0, 2\pi)$ with steps $2\pi/N$. We also have a nice formula for the inverse discrete Fourier transform:

$$x(k) = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} \hat{x}(n) e^{2\pi i k n/N} \quad \text{for } k = 0, 1, \ldots, N-1. \tag{2}$$

In Matlab, we compute the discrete Fourier transform efficiently using `fft` (fast Fourier transform) and `ifft` (inverse fast Fourier transform).

Convolution is defined in this context as

$$(x * y)(n) = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} x(k) y(n-k) \quad \text{for } k = 0, 1, \ldots, N-1. \tag{3}$$

An important result is the Convolution Theorem for the DFT:

$$\widehat{x * y}(k) = \hat{x}(k)\hat{y}(k) \quad \text{for } k = 0, 1, \ldots, N-1. \tag{4}$$

We apply this property of the convolution to "filter" signals by considering the product in the transform domain. We can alter the signal x by changing its frequency content through multiplying its transform $\hat{x}$ by some $\hat{y}$ designed to make the desired changes.

Instructions: Submit the requested figures as .fig or .tif files to tleise@amherst.edu by Wed 10/21.

Exercise 1 (FFT in Matlab) Use Matlab to construct a signal of length $2^{13} = 8192$ that contains only three frequencies with the following commands:

```
N=8192;
t=2*pi*(0:N-1)'/N;
x=0.5*sin(200*t)+0.2*sin(455*t)-0.3*sin(672*t);
figure;subplot(2,1,1),plot(t,x);xlabel('Time t');ylabel('Signal x')
```

Next compute and plot the Fourier transform (noting that the discrete Fourier transform of a real-valued signal will be symmetric, so we'll just examine the first half; we divide Matlab's FFT by $\sqrt{N}$ because its formula doesn't have this normalizing factor):

```
X=fft(x)/sqrt(N);
freq=(0:N/2-1)';
subplot(2,1,2),stem(freq,abs(X(freq+1)));
xlabel('Frequency k');ylabel('DFT xhat(k)')
```

Dividing by sqrt(N) and replacing `abs` with `2*real` gives the cosine term amplitudes and with `-2*imag` gives the sine term amplitudes for each frequency. Plot both of these as stem graphs (combined into a single plot) and use the data cursor feature of the figure window to verify that the three spikes give the correct values of the amplitudes and corresponding frequencies of the three sine terms. Submit the sine amplitude graph with appropriate labels.

Exercise 2 (Basic convolution and filtering) Suppose we want a "low-pass filter" that removes all frequencies larger than 400. The easiest way to do this is to multiply the signal's Fourier transform by an array that equals 1 for values of k we want to retain and equals 0 otherwise. In Matlab this looks like

```
Y=[ones(400,1); zeros(N-2*400,1); ones(400,1)];
```

Redo the plots of Exercise 1 for a new signal z that has DFT `Z=X.*Y` and can be computed via `z=real(ifft(Z))*sqrt(N)`. Submit a graph showing both the original signal and the filtered signal, with the horizontal axis set to [0 0.5].

To create a high-pass filter, we could flip the zeros and ones in `Y` to retain only the high frequencies and remove frequencies less than 400. Apply this high-pass filter to the original signal and submit a graph showing both the original signal and the filtered signal, with xlim set to [0 0.5].

Exercise 3 (Detecting a signal above the noise) Add noise to your signal from Exercise 1:

```
xnoisy=x+randn(size(x));
```

Plot the noisy signal and then find and plot the absolute value of its DFT. We can't make out the signal very well in the time domain–it looks buried in noise–but it's clear in the frequency domain. The "white noise" we added has a uniform distribution in frequency, so we can try to remove it by keeping only the DFT coefficients whose absolute value rises above the noise threshold. Examine your DFT plot and determine a threshold value that

most of the noisy DFT coefficients fall below. Then clean up the signal as follows:

```
Xcleaned=X.*(abs(X)>=threshold); % sets to zero xhats below the threshold value
signal=real(ifft(Xcleaned)); % take real part to remove any tiny imaginary part
```

Submit a figure with 3 subplots showing the noisy signal, the absolute value of its DFT, and the recovered signal plotted with the original (non-noisy) signal (set xlim to give a clear view in each subplot, label the axes, and add a legend in the 3rd subplot).

Exercise 4 (Sunspots) Read a bit of background about sunspot numbers on the NASA website http://solarscience.msfc.nasa.gov/SunspotCycle.shtml. Download the sunspot data file from the Math 320 website (spot_num.csv) to the same folder as your .m file for this lab. This file contains monthly sunspot number data from 1749 to the present. To read the file into Matlab:

```
M=dlmread('spot_num.csv',',',1,0); % reads data from comma-delimited text file
t=M(:,1)+M(:,2)/12; % first column is year, second column is month
y=M(:,3); % third column has sunspot numbers
```

Plot this data so you can see how the sunspot number varies over time (submit a labeled plot). There is clearly some periodic behavior occurring, which we can try to detect using the DFT. Plot the absolute value of the DFT of the sunspot numbers. The frequencies will be the integers n divided by the total time duration (`t(end)-t(1)`). For some applications, an easier way to interpret the DFT is to plot it against period, rather than frequency, noting that period is the reciprocal of frequency.

Note: don't change the `freq` vector–just plot as before but use `freq/(t(end)-t(1))` as the x-component of the graph to plot as frequency, or `(t(end)-t(1))./freq` to plot as period.