ECE 203
Lab 5: The DFT and Spectrum Analysis

1 Introduction

The objective of this lab is to use computers to analyze the spectra of real-world signals. The Discrete Fourier Transform (DFT), which is the discrete-time version of the Fourier series, is the main tool we will be using. Once a signal is sampled, the spectrum can be analyzed by applying the DFT to the samples. The DFT can be rapidly computed using an algorithm known as the Fast Fourier Transform (FFT). Matlab has a built-in function, `fft`, that implements this algorithm. The FFT algorithm was discovered by Gauss and later rediscovered and popularized by Cooley and Tukey in the late 1960's. The ability to perform spectrum analysis on a computer revolutionized science and engineering, and the FFT is one of the most important tools in signal processing and computational science. In fact, you have all encountered the FFT before; every time you listen to an MP3 file or view a JPEG image you are using an FFT-based algorithm. See `http://en.wikipedia.org/wiki/Fft` for more information on the FFT.

2 Overview

The DFT is the Fourier series representation for discrete-time signals. Let $x[n]$, $n = 0, \ldots, N-1$, denote signal of duration N samples. The DFT coefficients of $x[n]$ are computed by

$$a_k = \frac{1}{N} \sum_{n=0}^{N-1} x[n]\, e^{-j2\pi \frac{k}{N} n}, \quad k = 0, \ldots, N-1 \tag{1}$$

The signal $x[n]$ can then be represented or synthesized by the following weighted sum of complex exponential functions

$$x[n] = \sum_{k=0}^{N-1} a_k\, e^{j2\pi \frac{k}{N} n}$$

The DFT coefficient a_k appropriately weights the corresponding complex exponential function,

$$e^{j2\pi \frac{k}{N} n} = \cos\left(2\pi \frac{k}{N} n\right) + j \sin\left(2\pi \frac{k}{N} n\right)$$

The digital frequency of this function is $\widehat{f} = k/N$ cycles/sample. To relate the digital frequency to the corresponding frequency of the original sampled signal, simply covert from cycles per sample to samples per second. If $x[n]$ resulted from sampling a continuous-time signal $x(t)$ at a rate of f_s samples/second, then the digital frequency $\widehat{f}$ corresponds to

$$\widehat{f} \text{ (cycles/sample) } \times f_s \text{ (samples/sec) } = f \text{ (cycles/second or Hz)}$$

Matlab defines the DFT just a little differently. Instead of including the $1/N$ normalization in the calculation of the DFT coefficients, it includes the normalization factor in the synthesis formula. Specifically, the function `fft` computes $X(k) = \sum_{k=0}^{N-1} x[n]\, e^{-j2\pi \frac{k}{N} n}, \quad k = 0, \ldots, N-1$. The function `ifft` computes the synthesis (or inverse) of the DFT, $x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X(k)\, e^{j2\pi \frac{k}{N} n}$. Note that since $X(k) = N a_k$, this is equivalent to the synthesis formula above.

3 Warm-up

3.1 Computing the DFT

Consider the continuous-time sinusoid

$$x(t) \;=\; A\cos(2\pi f_0 t + \phi)$$

Sampling this sinusoid at a rate of f_s (i.e., one sample every $T_s = 1/f_s$ seconds) results in the discrete-time signal

$$x[n] \;=\; A\cos(2\pi f_0 T_s n + \phi)$$

If we sample the sinusoid over the time period $0 \leq t \leq T$ seconds, then we will acquire $N = T \times f_s$ samples. The following Matlab script generates and plots a sampled signal:

```
A = 10;
f0 = 100;
phi = pi/4;
fs = 400;
Ts = 1/fs;
T = 1;
N = T*fs;
tn = (0:N-1)/fs;
x = A*cos(2*pi*f0*Ts*(0:N-1)+phi);
figure(1)
plot(tn,x)
```

What is the frequency of the sinusoid that was sampled? Was it adequatley sampled, at or above the Nyquist rate? How many samples were acquired per cycle of the sinusoid?

The DFT of this signal can be computed by simply implementing the mathematical formula above in (1). For example, this function computes the DFT coefficients:

```
function X = mydft(x)
N = length(x);
for k=1:N
   X(k) = 0;
   for n=1:N
      X(k) = X(k)+x(n)*exp(-j*2*pi*(k-1)/N*(n-1));
   end
end
```

Make sure you understand how this function computes the coefficients given by the formula in (1). The output X is a vector of the DFT coefficients. Create another function, `my_idft`, that takes X and synthesizes according to $x[n] = \frac{1}{N}\sum_{k=0}^{N-1} X(k)\, e^{j2\pi\frac{k}{N}n}, \quad n = 0, \ldots, N-1$. Compare the your functions to the corresponding built-in Matlab functions `fft` and `ifft`. Do your functions generate the same results? Compare the speed of your functions to the built-in Matlab functions using this script:

```
tic;
myX = mydft(x);
mytime = toc;
tic;
matX = fft(x);
mattime = toc;
```

How does `mytime` compare to `mattime`? Is the FFT fast?

Instructor Verification

3.2 Spectrum Analysis with the DFT

The DFT computes the coefficients corresponding to the digital frequencies k/N, $k = 0, \ldots, N-1$. Note that the frequencies are uniformly spaced between 0 and 1. For values of $k > N/2$, the frequencies are in the range between 1/2 and 1. The corresponding complex exponentials $e^{j2\pi(k/N)n}$ are equivalent to $e^{j(2\pi(k/N)n - 2\pi n)} = e^{-j2\pi(1-k/N)n}$ (i.e., complex exponentials with negative frequency $-(1-k/N)$, which lies between $-1/2$ and 0). It is often convenient to reorganize the DFT coefficients to range over digital frequencies from $-1/2$ to $1/2$, instead of the range 0 to 1. By doing this, the complex exponential pairs (which combine to cancel the complex j sin components) appear symmetrically on either side of the origin. This makes it easier to interpret the plot of the coefficients as a spectrum. The

Matlab function `fftshift` automatically performs this reorganization. The following script plots the magnitude of the DFT coefficients over the $[0, 1]$ range and the $[-1/2, 1/2]$ range:

```
figure(2);
X = fft(x);
fhata = (0:N-1)/N;
subplot(2,1,1);
plot(fhata,abs(X));
fhatb = (-N/2:N/2-1)/N;
subplot(2,1,2);
plot(fhatb,fftshift(abs(X)));
```

To more easily interpret the magnitude of the DFT coefficients, we can scale the frequency axis to convert from units of cycles/sample to cycles/second (Hz). To do this we simply scale the frequency values k/N by the sampling rate f_s:

```
figure(3);
fHz = (-N/2:N/2-1)/N*fs;
plot(fHz,fftshift(abs(X)));
xlabel('frequency in Hz');
ylabel('magnitude of DFT coefficients');
title('Spectrum of signal x(t) = A cos(2*pi*f*t+phi)');
```

Note that, as expected, we see two peaks at -100Hz and 100Hz. Now generate samples of a sinusoidal signal with A= 5, f= 150Hz, phi= 0, and fs= 600 over a time period of T= 1.25 seconds. How many samples are generated? Plot the spectrum of the signal. Is it what you expected?

Instructor Verification

4 Spectrum Analysis and the DFT

4.1 Spectrum Analysis and Leakage

Mathematically explain why the spectrum of the 150Hz sinusoid in the warm-up did not consist of two perfect peaks (see Section 13-5.5 in the textbook). This effect is often called "spectral leakage".

4.2 Digital Music Equalizer

The DFT is the ideal tool for designing a digital music equalizer. We can boost (or suppress) various frequencies in a song using the DFT. The rap song `rap.mat` is in the "music clips" folder that you can download from the moodle site. The signal is 1048576 samples in length. The sampling rate is 44.1kHz. You can play the clip by executing the commands `load rap.mat` followed by `soundsc(x,fs)` in Matlab.

a. Given the length and the sampling rate, how many seconds of the song were recorded?

b. Use the Matlab command `fft` to compute the DFT of the signal and plot its spectrum (plot the log of the magnitude of the DFT, using `semilogy` instead of `plot`, and scale the frequency axis to display Hz). Do you observe a distinct drop in energy above a certain frequency? What would you suggest is the "effective" bandwidth (i.e., the maximum frequency) of the signal? Was it oversampled?

c. To "boost" the bass, amplify the low frequency components of the signal corresponding to frequencies at 500Hz and below by a factor of 3. Determine which DFT coefficients should be amplified given this specification. *Note: you need to amplify both the positive and negative frequencies in order to preserve the symmetry of the DFT coefficients.* After amplifying the low frequency DFT coefficients, reconstruct a new signal using the `ifft` command. Listen to the boosted track and plot its spectrum to compare with the original spectrum plotted in b.

d. Try out your **MEGAbass** effect on another song. There are several .wav files in the "music clips" folder. Some of these tracks are longer in duration, but all are sampled at 44.1KHz. For example, to read and play "freak.wav" use this script:

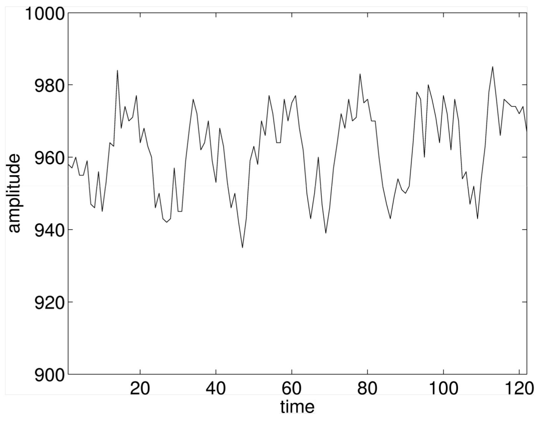
```
x = wavread('freak');
soundsc(x,44100);
```

In case you are interested, there are several free pieces of software that convert from MP3 format to wav format. For example, try `http://media.io`. Or you can try using mp3 readers built for Matlab
`http://www.mathworks.com/matlabcentral/fileexchange/13852-mp3read-and-mp3write`.

4.3 Functional Magnetic Resonance Imaging

Magnetic resonance imaging (MRI) can be used to detect regions in the brain that are activated by a task or stimulus. The basic idea is to collect a series of MRI brain images over time, a movie of the brain so to speak, and then look at how the value of each pixel varies over time. If the stimulus or task is periodic, then the pixels in regions of the brain that are activated by the stimulus or task will also display a periodic variation. This sort of brain analysis is called functional MRI (fMRI). See `http://en.wikipedia.org/wiki/Fmri` for more information.

The signal depicted below comes from a fMRI experiment. The signal reflects the variation of the blood oxygenation level in a certain region of the brain over time. This variation is correlated with neural processing. In this case, the person being imaged was periodically tapping one finger, and the oscillation in the fMRI signal (recorded from a region in the motor cortex) correlates with this repetitive exercise. A Matlab file containing the signal, called `fmri_sig.mat`, is available at on the moodle page.



Analyze the spectrum of the fMRI signal using the DFT (more specifically, use Matlab's `fft` function to compute the DFT coefficients of the signal).

a. Plot the magnitude of the DFT.

b. Identify the dominant sinusoidal component.

c. Assuming a sampling rate of 0.5 samples/sec, what is the frequency (in Hz) of the dominant sinusoidal component?

d. Find a good DC offset plus sinusoidal fit to the fMRI signal using the magnitude and phase of the DFT coefficients associated with the dominant component. Plot and compare your fit with the raw data.

Next, you will use the results above to map neural activity in the brain using fMRI. In fMRI, a series of MRI brain images are collected over time. Because oxgenated and deoxygenated hemoglobin have slightly different magnetic characteristics, variations in the MRI intensity indicate areas of the brain with increased blood flow and hence neural activity. The main challenge in fMRI is reliably detecting neural activity at different spatial locations (pixels) in the brain. The signals are noisy and the variation in intensity due to activation is very subtle. Consequently, statistical signal detection methods are routinely used to derive an "activation" map; a 2-d binary image of active and non-active brain regions. The objective of this problem is to develop an DFT-based procedure for producing an activation map. To get you started, start with the simple program `fmri.m` that inputs the images from the datafile `fmri.mat`, both of which are available on the moodle page. The basic idea is the compute the DFT of each pixel's time signal, and then check/detect the presence of a peak at the frequency corresponding to the period of the activation signal. In this case, the person was periodically tapping their finger, as above, and a representative response from an activated pixel is given in `fmri_sig.mat`. In developing your brain-mapping scheme, address/consider the following issues.

e. Determine an appropriate threshold for deciding whether or not a pixel is active. Discuss how and why you selected a particular threshold.

f. Discuss other variations in the pixel signals (DC offset, slowly varying drifts, noise) and their impact on your mapping procedure.

g. Produce a map of activated pixels by superimposing (in color) the activated pixel locations over a gray-scale image of the brain.