Music 270a: Matlab Tutorial 3

Tamara Smyth, trsmyth@ucsd.edu
Department of Music,
University of California, San Diego (UCSD)

January 28, 2013

This tutorial is available in its original form at http://ccrma.stanford.edu/ jos/mdft/

FFT of a simple sinusoid

The FFT, or "fast fourier transform" is an efficient implementation of the discrete fourer transform if the signal length is a power of two. In this example, we will use Matlab to take the FFT. Recall our simple discrete sinusoid is

$$x(t) = \cos(\omega_0 nT + \phi). \tag{1}$$

which we may implement in Matlab as

```
N = 64; % signal length (power of 2)
T = 1;   % sampling period (and rate) is set to 1
A = 1; % sinusoid amplitude
phi = 0; % phase of zero
f = 0.25; % frequency (under Nyquist limit)
nT = [0:N-1]*T; % discrete time axis
x = cos(2*pi*f*nT + phi); % sinusoid
X = fft(x);
```

In the last line, we use Matlab's `fft` function to obtain the spectrum of the sinusoid. Since `X` is complex, we do no usually plot it as is. Rather, to obtain a more meaningful graph, we first obtain the magnitude before plotting. Recall that the magnitude of a complex number $z = x + jy$ is given by

$$\text{mag } z = |z| = \sqrt{x^2 + y^2}. \tag{2}$$

In matlab we can use the `abs` function to obtain the abolute value of the spectrum. Therefore if we add the line

```
magX = abs(X);
```

to our code above we will have a sequence of real numbers representing the magnitude of the frequency components.

Likewise, we may obtain the phase using Matlab's `angle` function:

```
argX = angle(X);
```

Alternatively, we could use the following:

```
angleX = atan2(imag(X), real(X));
```

which implements the fact that the angle is given by

$$\angle z = \tan^{-1}\left(\frac{\mathrm{Im}\{X\}}{\mathrm{Re}\{X\}}\right).$$

Let's now make 3 plots: one to plot the time-domain evolution, one to plot the magnitude of the fft on a linear scale, and finally one to plot the magnitude on a dB scale.

```
% Plot time data:
figure(1);
subplot(3,1,1);
plot(n,x,'*k');
ni = [0:.1:N-1];     % Interpolated time axis
hold on;
plot(ni,A*cos(2*pi*ni*f*T+phi),'-k'); grid off;
title('Sinusoid at 1/4 the Sampling Rate');
xlabel('Time (samples)');
ylabel('Amplitude');
text(-8,1,'a)');
hold off;

% Plot spectral magnitude:
magX = abs(X);
fn = [0:1/N:1-1/N];  % Normalized frequency axis
subplot(3,1,2);
stem(fn,magX,'ok'); grid on;
xlabel('Normalized Frequency (cycles per sample))');
ylabel('Magnitude (Linear)');
text(-.11,40,'b)');

% Same thing on a dB scale:
spec = 20*log10(magX);        % Spectral magnitude in dB
spec = spec - max(spec);      % Normalize to 0 db max
subplot(3,1,3);
plot(fn,spec,'--ok'); grid on;
```

```
axis([0 1 -350 50]);
xlabel('Normalized Frequency (cycles per sample))');
ylabel('Magnitude (dB)');
text(-.11,50,'c)');
```

FFT of a slightly less simple sinusoid

The above example was somewhat contrived. It is more likely that the signals you analyse will have inusoidal components that don't complete a full cycle or, said differently, have frequency components that lie between fft bins. Let's see what happens in such a case.

First, increase the frequency in the above example by one-half of a bin:

```
% Example 2 = Example 1 with frequency between bins

f = 0.25 + 0.5/N;   % Move frequency up 1/2 bin

x = cos(2*pi*f*nT); % Signal to analyze
X = fft(x);         % Spectrum
```

Plot the resulting magnitude spectrum as you did above. Notice that at this frequency, we get extensive "spectral leakage" into all the side bins. To get an idea of where this is coming from, let's look at the periodic extension of the time waveform:

```
% Plot the periodic extension of the time-domain signal
plot([x x],'k');
title('Time Waveform Repeated Once');
xlabel('Time (samples)'); ylabel('Amplitude');
```

Notice the "glitch" in the middle where the signal begins its forced repetition. This results in "undesirable" frequency components in our spectrum.

Zero-padding

To get finer resolution in the frequency domain, we can zero-pad the signal, that is, append zeroes to the end of the time-domain signal.

```
zpf = 8;            % zero-padding factor
x = [cos(2*pi*f*nT),zeros(1,(zpf-1)*N)]; % zero-padded
X = fft(x);         % interpolated spectrum
magX = abs(X);      % magnitude spectrum
...                 % waveform plot as before
nfft = zpf*N;       % FFT size = new frequency grid size
fni = [0:1.0/nfft:1-1.0/nfft]; % normalized freq axis
```

```
subplot(3,1,2);
% with interpolation, we can use solid lines '-':
plot(fni,magX,'-k'); grid on;
...
spec = 20*log10(magX); % spectral magnitude in dB
% clip below at -40 dB:
spec = max(spec,-40*ones(1,length(spec)));
...                 % plot as before
```

Using a window

The previous examples can all be interpreted as using a **rectangular** window to select a finite segment (of length N) from a sampled sinusoid. To see the spectral characteristics of the rectangle window, try the following in Matlab:

```
N = 64;
zpf = 8;

x = [ones(1, N) zeros(1, (zpf-1)*N)]; %rectangle window

subplot(311); plot(x);
title('Rectangle Window');
xlabel('Time(s)');
ylabel('Amplitude');

Nfft = N*zpf;
X = fft(x);
magX = abs(X);
spec = 20*log10(magX);
spec = spec-max(spec);
spec = max(spec, -40*ones(1, length(spec)));
fn = 0:1/Nfft:1-1/Nfft;
subplot(312); plot(fn, spec);
xlabel('Normalized Frequency (cycles per sample)');
ylabel('Magnitude (dB)');

%replot showing negative frequencies below zero
X = fftshift(X);
magX = abs(X);
spec = 20*log10(magX);
spec = spec-max(spec);
```

```
spec = max(spec, -40*ones(1, length(spec)));
fn = -.5:1/Nfft:.5-1/Nfft;
subplot(313); plot(fn, spec);
xlabel('Normalized Frequency (cycles per sample)');
ylabel('Magnitude (dB)');
```

In practical spectrum analysis, such excerpts are normally analyzed using a window which is tapered more gracefully to zero on the left and right. In this section, we will look at using a Blackman window on our example sinusoid. The Blackman window has good (though suboptimal) characteristics for audio work.

In Octave or the Matlab Signal Processing Toolbox, a Blackman window of length $M = 64$ can be designed very easily:

```
M = 64;
w = blackman(M);
```

Many other standard windows are defined as well, including hamming, hanning, and bartlett windows.

In Matlab without the Signal Processing Toolbox, the Blackman window is readily computed from its mathematical definition:

```
w = .42 - .5*cos(2*pi*(0:M-1)/(M-1)) ...
       + .08*cos(4*pi*(0:M-1)/(M-1));
```

To see the spectral characteristics of the Blackman window try the following:

```
M = 64;
w = blackman(M);
figure(1);
subplot(3,1,1); plot(w,'*'); title('Blackman Window');
xlabel('Time (samples)'); ylabel('Amplitude'); text(-8,1,'a)');

% Also show the window transform:
zpf = 8;                        % zero-padding factor
xw = [w,zeros(1,(zpf-1)*M)];  % zero-padded window
Xw = fft(xw);                   % Blackman window transform
spec = 20*log10(abs(Xw));     % Spectral magnitude in dB
spec = spec - max(spec);      % Normalize to 0 db max
nfft = zpf*M;
spec = max(spec,-100*ones(1,nfft)); % clip to -100 dB
fni = [0:1.0/nfft:1-1.0/nfft];   % Normalized frequency axis
subplot(3,1,2); plot(fni,spec,'-'); axis([0,1,-100,10]);
xlabel('Normalized Frequency (cycles per sample))');
```

```
ylabel('Magnitude (dB)'); grid; text(-.12,20,'b)');

% Replot interpreting upper bin numbers as frequencies<0:
nh = nfft/2;
specnf = [spec(nh+1:nfft),spec(1:nh)];  % see fftshift()
fninf = fni - 0.5;
subplot(3,1,3);
plot(fninf,specnf,'-'); axis([-0.5,0.5,-100,10]); grid;
xlabel('Normalized Frequency (cycles per sample))');
ylabel('Magnitude (dB)');
text(-.62,20,'c)');
cmd = ['print -deps ', '../eps/blackman.eps'];
disp(cmd); eval(cmd);
disp 'pausing for RETURN (check the plot). . .'; pause
```