CS1031 - Lab 3
Fourier Transform and Spectrum Analysis

February 9, 2017

1 Introduction to the Fast Fourier Transform (FFT)

The Discrete Fourier Transform (DFT) is extremely important in the area of spectrum analysis. It takes a discrete signal in the time domain and transforms that signal into its discrete frequency domain representation; this allows the processing of its frequency components through computer-based algorithms, which wouldn't be able to work with a continuous analog function.[1]

The Fast Fourier Transform (FFT) is just a clever implementation of the DFT, specifically designed to be as fast as possible in terms of processing time; its use has become so widespread that the acronyms FFT (a specific implementation) and DFT (the generic analysis tool) have become inter-exchangeable.

1.1 Basic properties of the FFT

The time-continuous version of the DFT is periodic with period f_s, where f_s is the sampling frequency of the discrete-time signal; it thus makes sense to only define the DFT in the region between 0 and f_s.

Furthermore, it's easy to observe that the DFT is symmetric around the centre point $0.5f_s$, which is called the Nyquist frequency (see Fig. 1). For this reason, very often only half of the DFT is shown (e.g. the portion between $f = 0$ and $f = 0.5f_s$).

For a better understanding of these concepts, please refer to the lecture slides "More on Spectrum".

1.2 FFT in Matlab

In Matlab, the FFT function is under the Signal Processing Toolbox. You will need to install it (if you haven't done so when you first installed Matlab).

[1] Note that this is different from the Discrete-Time Fourier Transform (DTFT), which takes an discrete signal in the time domain and maps it into a continuous representation in the frequency domain.

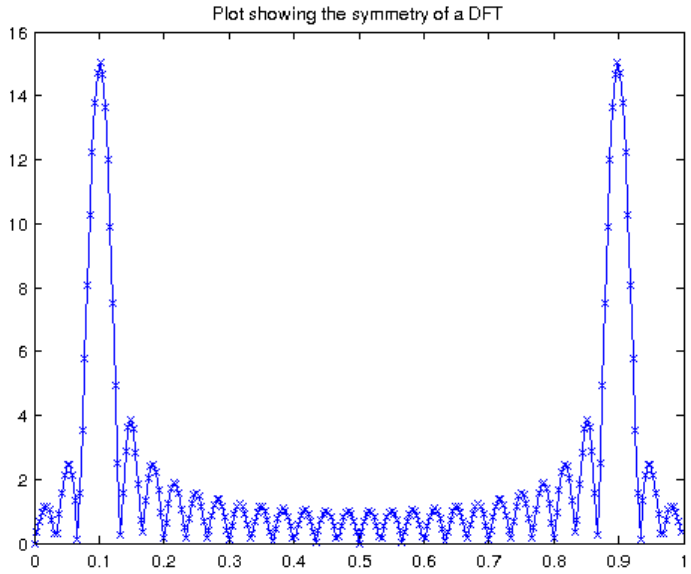

Figure 1: The DFT of a cosine with frequency equal to one tenth of the sampling frequency.

The typical syntax for computing the FFT of a signal is FFT(x,N) where x is the signal, x[n], you wish to transform, and N is the number of points in the FFT. N must be at least as large as the number of samples in x[n].

2 Exercises

2.1 Exercise 1: simple FFT plot

To demonstrate the effect of changing the value of N, produce a sine function with frequency 10Hz, 20 periods long. Remember to choose an appropriate sampling frequency f_s (e.g., 100Hz).

f_s=... put here your sampling frequency

time=... put here the number of seconds required to cover 20 periods of your function (e.g., 2)

frequency=... put here the frequency of your sine function

x=[0:1/f_s:time-1/f_s]; this is the x axis of your function

y=... write here your sine function with correct parameters

Define 3 different values for N, e.g. 64, 128 and 256. Then make the FFT for each of the 3 values of N. Use the abs function to find the magnitude of the transform, as we are not concerned with distinguishing between real and imaginary components. Then use fftshift, and normalise the x axis.

N=... this should be done for 64, 128 and 256;
F=fftshift(abs(...); fill in correctly
newX=-f_s/2:f_s/N:f_s/2-f_s/N; this is the x axis for your spectrum plot
plot(newX,F);

Plot the three different FFT (one for each value of N) in separate plots.

2.2 Exercise 2: Frequency Components With Noise

A common use of Fourier transforms is to find the frequency components of a signal buried in a noisy time domain signal. The array of values given in array.mat contains a signal made up of a number of sine waves with added random noise.

- First load the variable array in matlab assigning it to the variable y: use load('array.mat');
- Then plot the signal as it is (e.g., in the time domain) and check whether you can identify the frequency of the sine functions making up the signal (... probably not).
- Now do the FFT of the above signal, plot the frequency spectrum and tell which frequencies is the signal made of.

You can use code similar to that in exercise 1 to carry out the FFT. Note that the sampling frequency f_s used to produce the array.mat file used was 1000 Hz. Use a value of at least N=1024 for the FFT.

2.3 Exercise 3: Spectrum of Music

In this exercise you will carry out an exercise similar to the one shown at the end of the lecture slides "More on Spectrum".

From the file 'exercise notes.wav', containing samples from two notes, you will need to: separate the two notes into separate arrays, and find their tune, by examining their spectrum of each array individually.

You will need to carry out the following steps in order to complete the exercise:

- First load the sound samples using the following command: [notes,fsampling] = audioread('exercise notes.wav');
- Then separate the two notes into two different arrays. In order to understand where the second node begins you can plot the initial array and/or listen to the two separate arrays that you create using the function: sound(array). Notice that if you have an array 'test' of 100 points you can divide it into two arrays of say 35 and 65 points doing: test1=test(1:35); test2=test(36;100);

- Carry out the FFT for each array and plot the result. For the FFT make sure you use a value of N equal or bigger than the number of samples in the array you are calculating.
- Find the pitch of each of the two sounds by looking at the spectrum and use the pdf document title 'frequency of musical notes.pdf' to identify the note.