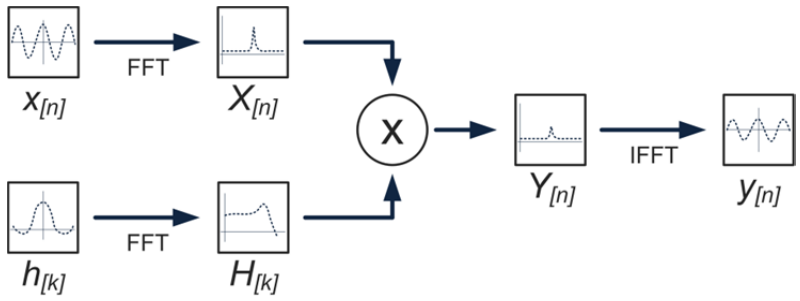
EE 419 - Project 3
Discrete Fourier Transform and FFT Signal Processing



Learning Objectives:
Use Matlab to process discrete-time signals using the DFT/FFT, including:

- **Signal Spectrum Analysis**
- **Fast Convolution**

REMINDERS:
1) When you are finished, be sure to always copy your m-files to a place that you can retrieve them again in the future (copy them to a memory stick, email them to yourself, etc.).
2) Please then delete your files from the Lab Computer (so that everyone else has to work as hard as you have!)

1) Plotting the Magnitude Spectrum of a Discrete-Time Signal

In Matlab, the Discrete Fourier Transform DFT (a discrete [sampled] frequency spectrum) can best be computed for a sample sequence **x** using the function call: **`fftx = fft( x );`** The FFT (Fast Fourier Transform) is a fast computation algorithm for the DFT. The fft() function in Matlab, when used as shown above, will return a sequence of numbers that has the same length as the array that was passed to it (**x**); but containing equally-spaced samples of the complex DTF spectrum of the finite sequence passed to it; starting with discrete-time frequency F=0 and ending at the frequency F={ 1 - (1/number_of_samples)}. For example, if **x** contains 10 samples, fft(x) will return the frequency response samples corresponding to Digital Frequency values F=[0 : 9]/10.

Create a Matlab function that plots the DFT frequency spectrum for a signal that is passed to it. The function call should have the format:

```
[ DFTx, Fd ]=plot_DFT_mag(x,fsample,figure_num);
```

where the input arguments are:

```
x = time domain samples of a discrete-time sequence
fsample = sampling frequency (samples / second)
figure_num = number of the figure to use for the two plots
```

and the outputs are:

```
DFTx = DFT spectrum values (complex)
     [same # samples as x]
Fd = Digital frequency values for each DFT sample
```

This function should create one figure with 2 different versions of the DFT magnitude spectrum plot.

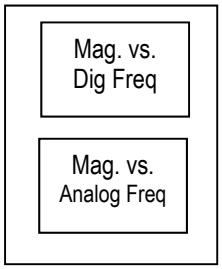
- Use **stem** plots for the Magnitude response, since the DFT and FFT are themselves discrete sequences (with complex values, so they have a magnitude and phase). Use the '.' marker instead of the standard circle for an easier to read stem plot.
- All plots use a **linear** scale for the **frequency** axis, and for the magnitude response (not dB).

- **Normalize the magnitude responses** by dividing the DFT values by the number of samples. This should yield magnitude values similar to a continuous-time Fourier Transform.
- Label all axes appropriately, and provide a title for the each plot.

The two versions created should be the following:

(1): Plot the DFT magnitude vs. **digital** frequency on the upper subplot.

(2): Plot the DFT magnitude vs. equivalent **analog** frequency [assuming ideal sampling]. Plot this in the lower subplot.



Test your function (and turn in the results) using the following test signals:

1. Cosine function: freq= 8 Hz, phase=0, amplitude=1, fsample= 20 Hz, # samples=100;
2. Cosine function: freq= **10 Hz**, phase=0, amplitude=1, fsample= 20 Hz, # samples=100;
3. Cosine function: freq= **12 Hz**, phase=0, amplitude=1, fsample= 20 Hz, # samples=100;
4. Cosine function: freq= **7.05 Hz**, phase=0, amplitude=1, fsample= 20 Hz, # samples=100;
5. Cosine function: freq= 7.05 Hz, phase=0, amplitude=1, fsample= 20 Hz, # samples=100 **added to** Cosine function: freq=**8.17** Hz, phase=0, amplitude=**0.25**, fsample= 20 Hz, # samples=100;
6. Same as (5), but multiplied by a **Blackman window** (100 point window).
   xn .* blackman(100).' (window functions must be transposed by .' operator.)
7. Same as (5), but # samples = **200**; [2 signals, 200 samples, no window]
8. Same as (6), but # samples = **200**; [2 signals, 200 samples, with window]
9. Cosine function: freq= 7.05 Hz, phase=0, amplitude=1, fsample= 20 Hz, # samples=200 added to Cosine function: freq= **7.25** Hz, phase=0, amplitude=**1**, fsample= 20 Hz, # samples=200;
10. Same as (9), but multiplied by a **Blackman window** (200 point window)
11. Unit sample sequence: # samples = 40; fsample= 1 KHz;
12. Unit sample response of the filter with difference equation:
    y[n] = 0.2x[n] + 0.2x[n-1] + 0.2x[n-2] + 0.2x[n-3] + 0.2x[n-4] # samples = 40; fsample= 1 KHz;

Answer the following questions based on your results:

1. Explain the position (frequency) of the cosine function's magnitude response peaks in Test Signal #3 above.
2. Describe and explain the unusual spectrum of Test Signal #4.
3. Compare the spectra of Test Cases #5 & 6. Explain what you observe about the ability to distinguish the spectrum of each signal, and why it is better in one case than the other.
4. Compare the spectra of Test Cases #5 & 6 to #7 & 8. Describe the effect of increasing the number of true sample values on the spectra.
5. Compare the spectra of Test Cases #9 & 10. Explain what you observe about the ability to distinguish the spectrum of each signal, and why it is better in one case than the other.
6. Why does the unit sample sequence (Test Signal #11) have the frequency spectrum shown? Is it what you expected?
7. What filter response characteristic is shown in the DFT results of Test Signal #12 (unit sample response)?
8. Based on the appearance of the DFT results of Test Signal #12, what sort of filter would you say created the tested unit sample response (low-pass, band-pass, high-pass, or band-stop)?

2) Fast Convolution Using FFT With Matlab

Using Matlab, create an m-file function that will perform fast Linear time-domain convolution of two arbitrary, finite-length sequences, using frequency domain spectral multiplication and Fast Fourier Transforms (FFT). To accomplish linear convolution (to prevent circular convolution), the function needs to ensure that the number of spectral samples to be multiplied (and therefore the FFT lengths) are sufficient to contain the full result of the linear convolution. Therefore, the **individual sequences must be zero padded** to this length before DFT transformation to the frequency domain (which the Matlab FFT function will take care of for you, if you specify the proper FFT length (see "help fft")). Note that you will have most of the code for this function, if you were present (and awake) during the first DFT review lecture.

Since we will want to use this function to evaluate the results of digital filtering of very long sequences, it is important to maximize the speed of these computations. Therefore, in addition to using a "Fast Fourier Transform" for these computations, you will need to **optimize its performance by forcing all sequence lengths (all FFT lengths) to be exact powers of 2** (M=1024, 2048, 4096,...]. One simple way to compute the nearest power of 2 that is greater than a particular number is:

$$2 = 10^{\log_{10}(2)}$$

$$2^n = 10^{n\log_{10}(2)}$$

$$M < 2^n \text{ so } M < 10^{n\log_{10}(2)}$$

$$\log_{10}(M) < n\log_{10}(2)$$

$$n > \frac{\log_{10}(M)}{\log_{10}(2)}$$

$$n = ceil\left[\frac{\log_{10}(M)}{\log_{10}(2)}\right]$$

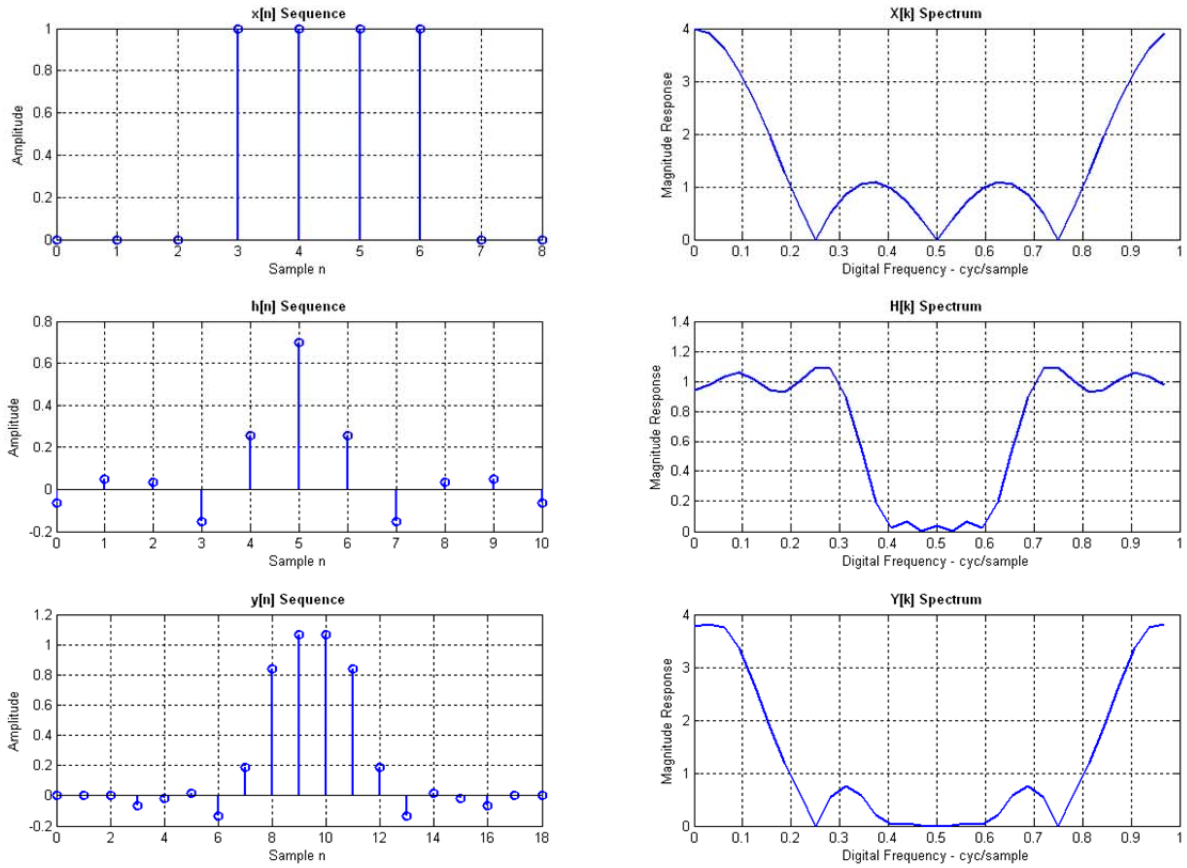
[or in Matlab: $n = \text{nextpow2}(M)$]

Note that if you zero pad your sequences beyond the length required for linear convolution, your resulting time function (yn) will have extra samples with values of zero at the end. **These should be removed by your program, so that it returns a sequence of the correct length for a convolution result**.

Your function should have the following format:

```
function yn = fftconv( xn, hn )
```

To help in debugging your function, and to confirm that it is working properly, have your function plot the three time sequences (xn, hn, yn) and their corresponding magnitude spectra (|Xk|, |Hk|, |Yk|) in a single figure, similar to that shown below:



where: `hn = fir1(10, 0.7, rectwin(11))` using Matlab, and x[n] is as shown.

For convolving very long sequences, the arrays and computations required for creating this figure will bog down the computer (and possibly "crash" Matlab if the system runs out of memory). Therefore, as a precaution, you should have your program skip the above plot if the sequence or FFT lengths are very long (>1000) [like with an "IF" statement.]

Turn in:

a) A listing of your m-file.

b) **The plot resulting from the following test case** (will NOT match the figure above).
xn= a 9-point hann window = `hann(9).'`
hn = [0 0 1 0 0 0 0 1 0 0]

3) [Lab] Verify the Multiple Notch 60 Hz Harmonic Filter Design

For your last homework assignment, you designed a multiple notch filter to eliminate 60 Hz noise and all of its harmonics (120 Hz, *etc.*) using the method of Pole/Zero placement. To preserve the rest of the signal, you wanted your filter notches to have a maximum -3dB width of 6 Hz. Your filter should NOT have eliminated the DC content of the signal, and should have maintained a gain of 1.0 (+/- 2%) in the centers of all of the passbands. The system had a sampling frequency of 360 Hz. You were to use as few poles/zeros as possible (to minimize the cost of implementing the filter.)

A) FIR Filter Design: First you tried to eliminate the noise **using an FIR filter** (all poles at Z=0). When you checked to see if you had achieved the specifications for this filter design by closely examining the frequency response plots produced by your Matlab analysis programs, you should have found that you had not achieved the specified 3dB notch width and passband center gains.

B) IIR Filter Design: So, to achieve the desired filter performance specifications, you had to **add some non-zero poles to your filter design**. This changed your filter to an IIR design. You experimented with the placement of the poles until you achieved the desired filter response [hopefully!!].

C) **Verification Testing:** NOW…**Test your IIR filter** from Part B using the corrupted tone data file (in a Matlab .mat file) available on the EE459 class PolyLearn site. (File: corrupted_tones.mat Duration: 2 seconds Sample Rate: 360 Hz Data in array called "y"). This file contains time-domain samples of a signal comprised of two tones (single frequency sine waves) that are corrupted by other tones at 60 Hz and its harmonics.

1) Using Time-Domain Filtering by Difference Equation Implementation
   a) Copy the corrupted tone input data file to your Matlab work directory.
   b) Use the Matlab command: `load corrupted_tones` to load the signal array "y" into Matlab.
   c) Plot a **magnitude** spectrum of the corrupted input data, using your plot_DFT_mag() function to generate its frequency spectrum.
   d) Process this corrupted signal through your filter's difference equation in the time domain using the Matlab "filter( )" function to produce the filtered output signal.
   e) Plot the **magnitude** spectrum of the filtered output signal, found again using your Matlab function to compute the frequency spectrum of the time-domain output signal. Plot the input and output signal spectra in different figures.
   f) Turn in BOTH of the two frequency spectrum magnitude plots.

2) Using Frequency-Domain Filtering by DFT (FFT) Processing and Spectral Multiplication
   a) Use your fftconv( ) function to convolve the corrupted input data signal with the unit sample response of the filter you designed.
   b) Try using both a 20 point unit sample response, and a 60 point unit sample response.
   c) Turn in the plot outputs for BOTH of the cases.

LAB REPORT REQUIREMENTS:

- Describe your design from Part (B), by providing the specifications, difference equation and Pole/Zero locations; and the Pole-Zero Diagram and Frequency Response plot (analog frequency axis) from your Matlab filter analysis program.
- Provide a complete report for the verification tests of Part (C). Include any Matlab m-files or command sequences used for generating the output signal and all the plots requested.
- Explain any differences between the time-domain filtering results and each of the two frequency domain filtering tests. Which results were the most accurate? Why were there inaccuracies in some of these results?