Spring 2013

EE 341
Lab 3: The DFT and Digital Filtering

When using a digital computer, frequency analysis and filtering requires that we use the Discrete Fourier Transform (DFT). In this lab we spend some time becoming familiar with using the Fast Fourier Transform (FFT) implementation of the DFT to study the frequency content of a discrete-time signal.

1. **MATLAB function FFT**

In this problem you will learn how to use the MATLAB command `fft`. First, use the `help` feature in MATLAB to learn the syntax of the `fft` function. The FFT function computes the Discrete Fourier Transform (DFT) of a sequence. In general the FFT of a sequence will be a complex function so you will need to look at the magnitude and phase separately. The MATLAB commands `abs` and `angle` are useful for obtaining the magnitude and phase of a complex valued sequence. Also, since the FFT only has values at discrete frequencies, it may be useful to do the plots with `stem` to reinforce that idea, but continuous frequency plots (i.e. using `plot`) are often used since they approximate the DTFT that you are ultimately interested in. You may use either one for your report.

The FFT outputs a sequence over the frequency range $0 \le \omega \le 2\pi$. You are probably more familiar with seeing the spectrum plotted over the range $-\pi \le \omega < \pi$. The `fftshift` function can be used for this purpose.

Plot the magnitude of the FFT of the following signal before and after `fftshift`:

$$x[n] = 1 + \cos(2\pi fn), \qquad 0 \le n \le 127$$

for the cases where $f = 0.25$ and $f = 0.5$. Use your understanding of the relation between discrete and continuous time to plot the magnitude of the Fourier Transform of the continuous time signal that these correspond to, assuming the sampling period is $T = 10^{-4}$. Be sure to label the frequency axis correctly and indicate whether you are plotting in radians or Hertz or normalized frequency.

Turn in a 2-part plot (using the `subplot` command) for each signal: unshifted DFT and shifted DFT. Each plot must have a frequency axis labeled in Hz. Discuss why the frequency peak locations make sense.

In the second part of this assignment you will use `frevalz01.m` (on the class web page) to look at the behavior of system functions in the frequency domain. (The *z*-plane is also displayed, which we study in detail later, and is the discrete-time version of the Laplace *s*-plane for continuous-time systems.) MATLAB has numerous built-in functions for generating discrete-time filters. In this problem we are going to use a few to look at how higher order systems behave in the frequency domain. This will give you some insight into how digital filters are designed and into the properties of different digital filter design algorithms. You will not be expected to understand the details of how these algorithms work; you only need to evaluate their behavior.
Both the filter design programs in the problems below ask you to specify cut-offs W in terms of a normalized frequency $0 < W < 1$, where $W = 1$ corresponds to half the sampling frequency. In other words, $0 < W < 1$ corresponds to $0 < \omega < \pi$ in radians for the DTFT. Another way to normalize frequency, as plotted by `frevalz01`, is in

the range $0 < \overline{f} < 0.5$. Thus, a cut-off frequency of $W = 0.5$ corresponds to $\omega_c = \pi / 2$ in radians and $\overline{f}_c = 0.25$ in normalized frequency on the `frevalz01` frequency response.

2. **FIR (Finite Impulse Response) Digital Filters**
   Use the MATLAB function `fir1` to create a low pass FIR filter of order 10 with cutoff frequency of $\omega_c = 0.3\pi$. Use `frevalz01` to study the system. Turn in the `frevalz01` plots of the system responses. Save your filter in a vector since you will use it again in parts 4 and 5.

3. **IIR (Infinite Impulse Response) Digital Filters**
   Use the MATLAB function `butter` to create a low pass Butterworth filter with cut-off frequency $\omega_c = 0.3\pi$ and filter order of 10. How does the performance of this filter compare to that of the FIR filter? In this context, performance refers to how close a filter matches an ideal low pass filter:

$$H_{ideal}(\omega) = \begin{cases} 1, & |\omega| \le 0.3\pi \\ 0, & \text{otherwise} \end{cases}$$

   Comment on any differences in the phase of the two filters. Turn in the `frevalz01` plot of the system responses. Save your filter coefficients (*a* and *b* vectors) since you'll use them again in part 4.

4. **Filter Implementation**
   Use the MATLAB function `filter` to implement the two low pass filters you produced in problems 2 and 3 with the input signal $x_1[n]$ defined below. Define a pulse of length twenty:

$$x[n] = u[n] - u[n-20]$$

   and let $x_1[n]$ be a zero-padded version of total length 60 (so append 40 zeros on the end of $x[n]$ before filtering). Comment on the differences in the output of the two filters. Hand in plots of the time signal input and the outputs of both filters.

5. **FIR Filtering using the DFT and Linear Convolution**
   Using your FIR filter from part 2, find the linear convolution with the 20-point signal $x[n]$ defined in part 4 (without zero-padding, so only 20 points long). Use the `conv` command in MATLAB. How long is the output signal? Hand in a plot of the time-domain signal output.
   Now you will use the DFT to compute the same convolution in the frequency domain. Use `fft` to find $H[k]$, the DFT of the FIR filter, and $X[k]$, the DFT of $x[n]$. Find the time-domain output signal by multiplying in the frequency domain and then inverting the DFT:

```
y = ifft( X.*H );
```

   The result in $y[n]$ should be the same as when you used `conv`. (A tiny imaginary part may result from quantization error. You may use `real` to keep the real part of $y[n]$.) For this problem, what is the correct choice for the DFT-size `N` when you make your calls to `fft`? Why is it not possible to use the DFT to find the output with the IIR filter from part 3? Use E-Submit to turn in your code for this problem, and include your code with your lab report hard copy.