ECE 437 Lab #3 (MATLAB Project)

Introduction to (Frequency Domain) Digital Filtering

Purpose

This lab consists of two parts, namely,

- Part A: Plotting of power spectral density (PSD) in Matlab, and
- Part B: Digital filtering, which is in used many modern scientific instruments, software radios, and entertainment devices (stereo amplifiers, DVD players, etc.).

Part A: Plotting of power spectral density (PSD) in Matlab

Matlab has many built-in functions for analyzing a signal. One such function called *psd* plots the power spectral density of a signal. Before using this function, however, we will first generate a simple PSD plot using Matlab's fast Fourier transform function, *fft*. Then we will compare it with the plot generated by using Matlab's *psd* function.

Notice that fast Fourier transform (FFT) is really a fast way of computing the (discrete) Fourier transform (DFT) of a discrete-time signal. It is explained on pages 130-141 of Lathi's text, but we will not cover it in the class. However, since it is a wonderful tool used in signal processing and communication systems, we will just learn the basic ideas about it here. Please read the above pages of the text if you wish to get some knowledge of DFT and FFT.

Suppose we wish to analyze a continuous signal, x(t), using a computer. Since a digital computer cannot handle continuous-time signals, first we sample x(t) at uniform intervals, Ts, to get a discrete-time signal, x(nTs) or simply x(n). The reciprocal of Ts, i.e., 1/Ts, is called the sampling rate, Fs. An FFT of size N computes the so called n-point discrete Fourier transform (DFT) of x(n).

Suppose we denote the N-point FFT of x(n) by X(k), $1 \leq k \leq N$. One can show that k stands for the analog frequency value of k*Fs/N Hz. Thus X(k) represents an approximate value of the continuous Fourier transform of x(t), i.e., X(ω), at the frequency point, $\omega = 2\pi k*Fs/N$. If N is large enough, the approximation is pretty good. However, due to various other reasons, this approximation is often corrupted by noise.

Thus a plot of $|X(k)|^2$ against a correctly scaled frequency axis provides a good glimpse of the energy spectral density (ESD) of x(t). The power spectral density (PSD) is simply computed as an average of ESD calculated over the signal duration, T.

The above idea is illustrated in a program called *ECE437lab3a* that computes the approximate PSD of a rectangular pulse train. This rectangular pulse train was chosen because you are familiar with its PSD. This program also illustrates the Parseval's relation that we studied in class.

As mentioned before, PSD can also be plotted in Matlab using a sophisticated built-in function called *psd*. Execute

```
help psd
```

to learn more about this command. A starting set of commands for plotting the PSD is given below:

```
[Px,f] = psd(x,512,Fs);    % Computes PSD at frequency points stored in vector, f.
figure(3)
plot(f,10*log10(Px))     % plots the PSD in dB
```

Please note that in order to really use this function in an intelligent way, you have to play with different values of the parameters. Compare the plot generated using *psd* with what you obtained earlier. The purpose of the above exercise is to illustrate the idea that many tools exist, but as an engineer you have to choose the right one for a particular application.

Part B: Digital filtering

Background Information

We know that the response, y(t), of a linear system (such as, a filter) due an excitation, x(t), can be represented by an integral operation, known as "convolution operation":

$$y(t) = x(t) * h(t), \quad (1)$$

where h(t) denotes the unit impulse response of the system and "*" denotes the convolution operation. One of the remarkable properties of Fourier transform is that it allows us to simplify the above operation in the frequency domain.

Taking Fourier transform of both sides of equation (1) and using the convolution property of Fourier transform, we obtain:

$$Y(\omega) = H(\omega)X(\omega). \quad (2)$$

where $Y(\omega)$, $H(\omega)$, $X(\omega)$ denote the Fourier transforms of y(t), h(t), and x(t), respectively. Note that $H(\omega)$ also represents the frequency response function of the system.

Equation (2) provides the basis of linear filtering in the frequency domain. It involves the following steps:

1. Design a suitable frequency response function, $H(\omega)$;
2. Compute $X(\omega)$;
3. Compute $Y(\omega)$ from equation (2);
4. Compute y(t) by taking inverse FT of $Y(\omega)$.

When the above steps are performed in a computer (hardware or software) using sampled versions of x(t), h(t) and y(t), we call it a digital filter. Here we will simulate a DFT based digital

filter. Such a filter uses both DFT and its inverse, which is known as inverse discrete Fourier transform, or IDFT. As mentioned before, usually we use fast algorithms to compute DFT and IDFT – these are known as FFT and IFFT. Two such algorithms are available in MATLAB, namely, *fft* and *ifft*.

To implement a digital filter in MATLAB, we will assume that we have a sampled (and digitized) signal, $x(nT_s)$, to start with. For simplicity, we will denote it by x(n). Then proceed as follows:

1. Choose a frequency response function, $H(\omega)$, of a desired shape. Sample it at N points to get H(k). Basically, H(k) represents the N-point DFT of h(t).
2. Using MATLAB's *fft* function, compute N-point DFT of x(n) to get X(k);
3. Compute Y(k) = H(k)X(k);
4. Using MATLAB's *ifft* function compute IDFT of Y(k) to obtain the response y(n), or equivalently, $y(nT_s)$.

Description of the problem

The data file, lab408dat, contains a digitized audio clip corrupted by a hissing sound (which is typically what you hear from an old tape recorder). The audio clip is stored in a variable, x, and the sampling rate is given by Fs. To hear the audio clip, execute the following command:

soundsc(x,Fs).

Our goal is to restore this audio file using a simple filter.

Step 1. Perform a power spectrum analysis of the data using any of the two methods you learnt in Part A.

Step 2. The purpose of PSD analysis is to identify the frequency bands occupied by *useful signal* and *noise*. If we know the frequency band occupied by noise, we can design a suitable filter to remove it. So, analyze the PSD plot and make a judicious choice of the filter and its cut-off frequency, ω_c, required to restore the audio file.

Step 3. In this homework, we wish to implement an *ideal, zero-phase, frequency domain filter* using DFT and its inverse. The corresponding commands in MATLAB are: *fft* and *ifft*.

First compute the FFT of x, i.e., X(k). Then *select the filter's frequency response function, H(ω),* and then sample it to get H(k). Next, compute H(k)X(k), and its IDFT. See steps 1-4 discussed above.

Step 4. Play y(t) and make sure that the hissing sound is completely gone.

Please prepare a lab report describing what you did and the results you got. Also, please attach a copy of your program, and highlight the section containing implementation of the filter.