



**COMMUNICATION SYSTEMS
ECE 437
School of Engineering and Computer Science
Oakland University**

LAB 3

Introduction to PSD and Frequency Domain Filtering Using MATLAB

Introduction to PSD and Frequency Domain Filtering Using MATLAB

Purpose

This lab consists of two parts

Part 1: Plotting of power spectral density (PSD) in Matlab, and

Part 2: Digital filtering, which is in used many modern scientific instruments, software radios, and entertainment devices (stereo amplifiers, DVD players, etc.).

Part 1: Plotting power spectral density (PSD) in Matlab

Background Information

Matlab has many built-in functions for analyzing a signal. One such function is called periodogram. It returns an estimate of the power spectral density of the function using a modified periodogram method. 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 *periodogram* 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 in section 3.9 of Lathi's text.

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, T_s , to get a discrete-time signal, $x(nT_s)$, or simply $x(n)$. The reciprocal of T_s , i.e., $1/T_s$, is called the sampling rate, F_s . 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 kF_s/N Hz. Thus $X(k)$ represents an approximate value of the continuous Fourier transform of $x(t)$, i.e., $X(\omega)$, at the frequency point, $\omega = 2\pi kF_s/N$. If N is large enough, the approximation is pretty good. However, due to various other reasons, this approximation is often corrupted by noise. Even though it is noisy, a plot of $|X(k)|^2$ against a correctly scaled frequency axis provides a reasonably 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 .

Description of the problem

The above idea is illustrated in a program called *lab3part1* that computes the approximate PSD of a rectangular pulse train. This rectangular pulse train was chosen because you are familiar with its PSD.

As mentioned before, PSD can also be plotted in Matlab using a built-in function called *periodogram*. To understand how it works, execute

```
>>help periodogram
```

Before running *lab3part1* script, read over the code and make sure that you understand each command line.

Lab report requirement for Part 1

- In your version of *lab3part1* script, comment on the main commands of the code. Attach a copy of your commented code with your lab report.
- Additionally, include an explanation of each Figure generated by the program.

Part 2: Digital filtering

Background Information

We know that the response, $y(t)$, of a linear system such as a filter due to excitation, $x(t)$, can be represented by the 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. Notice that $H(\omega)$ also represents the frequency response function of the system.

Equation (2) provides the basis for 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, $y(nT_s)$.

For a more detailed discussion of the above procedure, please read Lathi's text (4th edition), pages 160-167.

Description of the problem

The data file *lab3Audio* contains a digitized audio clip corrupted by a hissing sound. The sampling frequency is 22050. So in the command window enter

```
>>Fs = 22050;
```

The audio clip is stored in a variable, x , and the sampling rate is given by F_s . 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 the *fft* method you learned in Part 1.

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

Step 3. In this step, 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*, $H(\omega)$, 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 the filtered audio clip, $y(k)$, through speakers to make sure that the hissing sound is gone.

Lab report requirement for Part 2

- Explain what you did and what you observed.
- In your version of *lab3part2* script, comment on the main commands of the code. Attach a copy of your commented code with your lab report.
- Include an explanation of the Figures 2-6 generated by the program.