ECEn 380 Signals & Systems Laboratory

Assignment #5: Discrete-Time Bandpass Filters

Due Date

This is a three week lab. Neither section will meet during the week of Thanksgiving, November 24–28. Labs submission for both sections is Wed., Dec 10 in the homework box, due by 5:00 p.m.

Note that the two tasks are not the same length, so work hard until you are done with both and don't just plan on one day per task.

Objective

The purpose of this assignment is to design and apply bandpass filters to data captured from a laser tag prototype system. Each laser tag player's gun modulates the laser intensity at a different frequency. The bandpass filters are used to determine the modulating frequency of the received light beam. The results are used to determine who the shooter was.

Reading Assignments

This lab brings together a number of different concepts. As such it is a nice way to end the class. The material needed from the text is listed below.

Topic	Reading Assignment
Analog Butterworth filters	Section 6-8, pp. 279–287
Sampling	Section 6-12, pp. 306–321
Discrete-time convolution	Section 7-5, pp. 348–350
The z-transform	Section 7-6, pp. 350–354
Stability	Section 7-11, pp. 365–367
System frequency response	Section 7-12, pp. 367–372
DTFT	Section 7-14, pp. 377–383
DFT	Section 7-15, pp, 383–387
Windowing	Section 7-15.2, pp. 384–385
Basic discrete-time filter concepts	Section 8-1, pp. 397–402

A handout entitled "An Introduction to Discrete-Time Filter Design" applies the reading assignments from the text to filter design.

The following MATLAB® commands will also be used: `load`, `:`, `length`, `log10`, `exp`, `conv`, `fft`, `freqz`, `fftshift`, `zplane`, `filter`, `pwelch`, `rectwin`, `butter`.

In addition, the following MATLAB® plotting commands will be used: `plot`, `subplot`, `grid`, `xlabel`, `ylabel`, `axis`.

Introduction

The ECEn 390 Junior Project involves the design of a ten-player laser tag system. A block diagram is shown in Figure 1. When one of the players shoots, an LED produces a light with a sinusoidally modulated intensity. Each player is given a unique modulating frequency. These frequencies are listed in the table below. Each player's body is equipped with a sensor comprising a light detector (whose output is a voltage proportional to the received light intensity), an amplifier, an anti-aliasing filter, a sampler (or A/D converter) operating at 100 ksamples/s, and a discrete-time processor comprising the A/D converter interface and a programmable processor. The discrete-time processor monitors the samples produced by the A/D converter to determine if the wearer was shot and who the shooter was. This lab is devoted to identifying the shooter.

The A/D output is approximately a sinusoid and a DC offset. The frequency of this sinusoid is unique to each player, as listed in table on the right. So, to identify the shooter, we must determine the frequency of the received light beam. In this lab, we use a bank of ten discrete–time bandpass filters, each centered at one of the frequencies listed in the table to identify the shooter.

Player	Frequency
Player # 1	20 kHz
Player # 2	22 kHz
Player # 3	24 kHz
Player # 4	26 kHz
Player # 5	28 kHz
Player # 6	30 kHz
Player # 7	32 kHz
Player # 8	34 kHz
Player # 9	36 kHz
Player # 10	38 kHz

Because we are operating on sampled data, we are interested in discrete-time filter design. Here we explore two basic forms of discrete-time filters: the FIR filter and the recursive IIR filter. The accompanying background material, "An Introduction to Discrete-Time Filter Design" explains what FIR and IIR filters are and outlines basic design approaches to each.

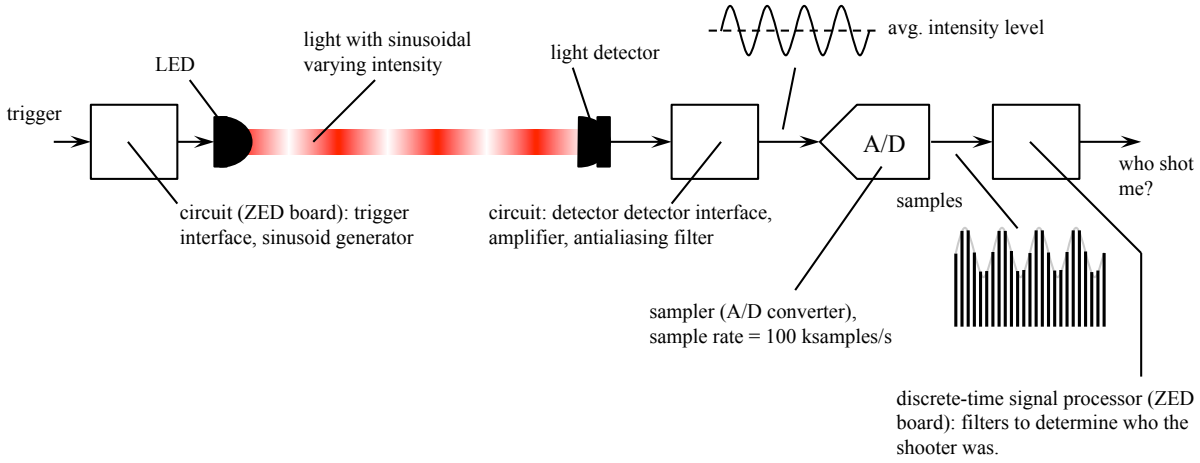

Figure 1: A block diagram of the laser tag system.

Task 1: FIR Filters

1. Windows (not Microsoft!):

   (a) Select three of the windows in Table 1 of the handout "An Introduction to Discrete-Time Filter Design" plot them for the length set equal to 51. What do they have in common? How are they different?

   (b) Using the MATLAB® commands `freqz` and `plot`, plot the DFT of the three windows you chose in part (a). The plots should be the magnitude in dB versus the discrete-time frequency. (See the example MATLAB® code in the "An Introduction to Discrete-Time Filter Design" for examples of how to make these plots.) How are they the same? How are they different?

2. Filter parameters. Here, we investigate the properties of the FIR filter

$$h_{\text{FIR}}[n] = w[n] \times h_{\text{ideal}}[n]$$

   where $h_{\text{ideal}}[n]$ is given by (18) in the handout "An Introduction to Discrete-Time Filter Design" and $w[n]$ is your favorite window from part (a).

   (a) Using the MATLAB® commands `freqz` and `plot`, plot the frequency response (DFT) of $h_{\text{FIR}}[n]$ using your favorite window $w[n]$ for the length fixed at 501 and for bandwidths $B = 0.002$, $B = 0.005$, $B = 0.02$ cycles/sample. The plots should be the magnitude in dB versus the discrete-time frequency.

(b) Using the MATLAB® commands `freqz` and `plot`, plot the frequency response (DFT) of $h_{\text{FIR}}[n]$ using your favorite window $w[n]$ and bandwidth $B = 0.01$ cycles/sample for lengths 101, 501, and 1001.

(c) Based on your plots from parts (a) and (b), what are the tradeoffs involved with filter length and bandwidth?

3. Based on the requirement to correctly identify a sinusoid in the received light beam, design ten bandpass filters. Plot the DFTs of the ten filters on the same set of frequency axes and use the plots to justify your design choices.

4. Apply your filters to the data sets.

   (a) Create an algorithm for processing the BPF outputs to identify the shooter.

   (b) Make adjustments to your filter designs, as necessary, to get the correct answer for each data set. (See the Appendix for how to know what the right answer is.)

   (c) Have the TA check your work and sign it off.

5. Items to include in your lab–book documentation:

   (a) Answer all the questions posed above. Be sure to include the plots used to justify your answers.

   (b) Document the values of L and B you used for your filters.

   (c) Document the post-filter processing algorithm.

   (d) For each data set, plot (or document) the results of the post-filter processing algorithm corresponding to the outputs of the ten bandpass filters.

   (e) Include a spectrum plot for each of the data sets and use the plot to justify your claim that your algorithm works.

   (f) When you have recorded all of this, show the TA and have him/her give you the final sign off for Task 1.

6. Extra Credit: In Example 7-31 (pg. 382), the impulse response $h[n]$ of an ideal discrete-time LPF is derived. [See Equation (7.165).] The impulse response of the ideal discrete time LPF is very similar to the impulse response of the ideal discrete-time BPF $h_{\text{ideal}}[n]$. How are they the same? How are they different? The relationship between the two has an interesting interpretation. Using DTFT Property 3 in Table 7-7 (pg. 378), provide an interpretation of the relationship between $h[n]$ and $h_{\text{ideal}}[n]$.

Task 2: IIR Filters

In general, recursive IIR filters are designed using pole placement. (See Section 8-1, pp. 397–402, of the text for a very nice discussion of the relationship between pole locations and frequency response.) Recursive BPF design is often carried out in two steps:

1. Design a recursive LPF to meet the system requirements.

2. Convert the recursive LPF to a recursive BPF by rotating the poles and zeros.

This approach requires the designer to understand the relationship between LPFs and BPFs. This relationship is illustrated in Figure 2. The key point is that a LPF with bandwidth $W/2$ rads/sample becomes a BPF with bandwidth W rads/sample after translating the pass band from $\Omega = 0$ rads/sample to $\Omega = \Omega_0$ rads/sample.

As explained in the handout "An Introduction to Discrete-Time Filter Design" the conversion for LPF to BPF centered at Ω_0 rads/sample proceeds as follows. Let the $\mathbf{z}$-domain transfer function of the LPF be

$$\mathbf{H}_{\text{LPF}}(\mathbf{z}) = \frac{b_0 + b_1\mathbf{z}^{-1} + b_2\mathbf{z}^{-2} + \cdots + b_M\mathbf{z}^{-M}}{1 + a_1\mathbf{z}^{-1} + a_2\mathbf{z}^{-2} + \cdots + a_N\mathbf{z}^{-N}}. \tag{1}$$

Then the corresponding BPF centered at Ω_0 is

$$\begin{aligned}\mathbf{H}_{\text{BPF}}(\mathbf{z}) = &\frac{b_0 + b_1 e^{j\Omega_0}\mathbf{z}^{-1} + b_2 e^{j2\Omega_0}\mathbf{z}^{-2} + \cdots + b_M e^{jM\Omega_0}\mathbf{z}^{-M}}{1 + a_1 e^{j\Omega_0}\mathbf{z}^{-1} + a_2 e^{j2\Omega_0}\mathbf{z}^{-2} + \cdots + a_N e^{jN\Omega_0}\mathbf{z}^{-N}} \\ &+ \frac{b_0 + b_1 e^{-j\Omega_0}\mathbf{z}^{-1} + b_2 e^{-j2\Omega_0}\mathbf{z}^{-2} + \cdots + b_M e^{-jM\Omega_0}\mathbf{z}^{-M}}{1 + a_1 e^{-j\Omega_0}\mathbf{z}^{-1} + a_2 e^{-j2\Omega_0}\mathbf{z}^{-2} + \cdots + a_N e^{-jN\Omega_0}\mathbf{z}^{-N}}.\end{aligned} \tag{2}$$

We can write this as

$$\mathbf{H}_{\text{BPF}}(\mathbf{z}) = \frac{\mathbf{B}^+(\mathbf{z})}{\mathbf{A}^+(\mathbf{z})} + \frac{\mathbf{B}^-(\mathbf{z})}{\mathbf{A}^-(\mathbf{z})}. \tag{3}$$

Using the simple rules for addition of fractions, we have

$$\mathbf{H}_{\text{BPF}}(\mathbf{z}) = \frac{\mathbf{B}^+(\mathbf{z})\mathbf{A}^-(\mathbf{z}) + \mathbf{B}^-(\mathbf{z})\mathbf{A}^+(\mathbf{z})}{\mathbf{A}^+(\mathbf{z})\mathbf{A}^-(\mathbf{z})}. \tag{4}$$

Note that the order of the LPF is N, but the order of the BPF is $2N$.

Consequently, the starting point is pole placement for the LPF. Of the many possible pole placement criteria, we will use the the Butterworth criteria. We use the Butterworth criteria because you have already seen continuous-time Butterworth filters. MATLAB® includes the command `butter` that computes the coefficients $b_0, b_1, \ldots, b_N$ and $a_1, a_2, \ldots, a_N$ for the equivalent

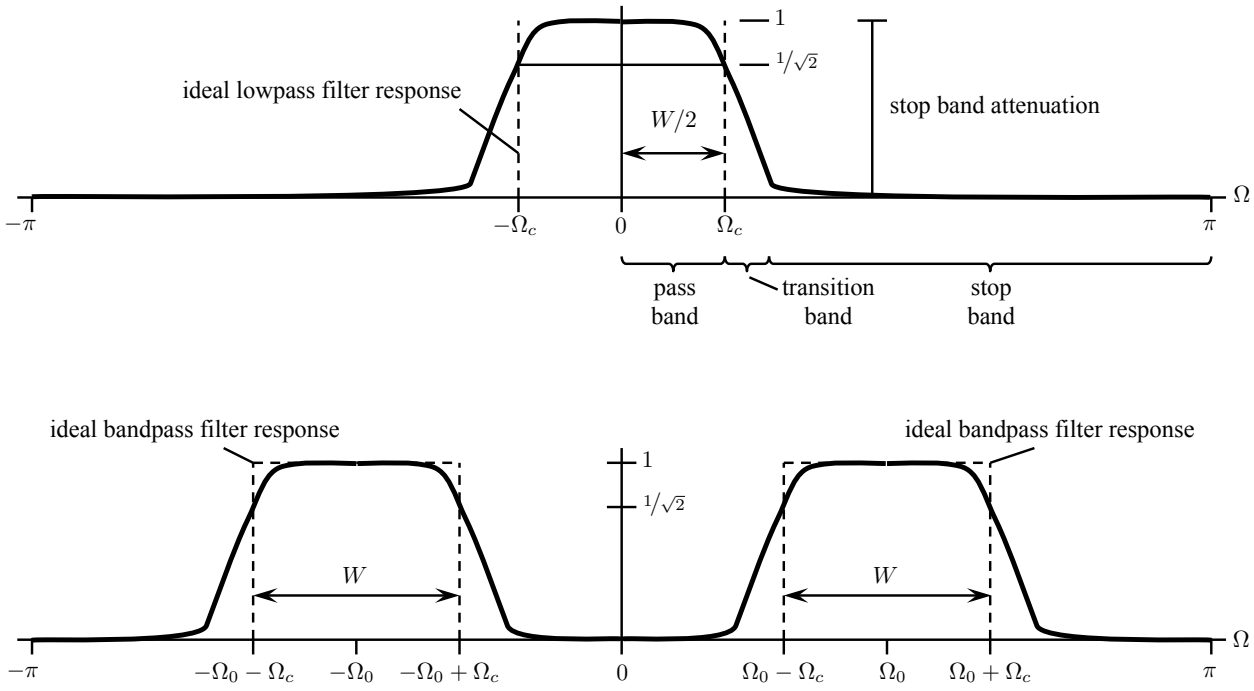

Figure 2: An illustration of converting a lowpass filter (top) to a bandpass filter (bottom) in terms of the discrete-time frequency variable Ω (rads/sample).

discrete-time filter.

As an example, suppose we want to design a 6-th order discrete-time Butterworth BPF centered at $\Omega_0 = 2\pi \times 0.2$ rads/sample and with a bandwidth $W = 2\pi \times 0.02$ rads/sample. The first step is to design a 3rd order discrete-time Butterworth LPF with bandwidth $2\pi \times 0.01$ rads/sample. The following segment of MATLAB® code creates performs this task. The code computes the polynomial products of (4) by exploiting the fact that the coefficients of the product of two polynomials is given by the convolution of the coefficients of multiplicand and multiplier polynomials. The two plots produced by this segment of MATLAB® code are shown in Figure 3.

```
n = 3;                                          % LPF filter order
Wc = 2*pi*0.01;                                 % LPF corner frequency
W0 = 2*pi*0.2;                                  % BPF center frequency
[b,a] = butter(n,Wc/pi);                        % create 3rd order B'worth LPF

aplus = a.*exp(1i*W0*(0:n));                    % rotate poles by W0
bplus = b.*exp(1i*W0*(0:n));                    % rotate zeros by W0
aminus = a.*exp(-1i*W0*(0:n));                  % rotate poles by -W0
bminus = b.*exp(-1i*W0*(0:n));                  % rotate zeros by -W0

bb = conv(bplus,aminus) + conv(bminus,aplus);   % BPF zeros
aa = conv(aplus,aminus);                        % BPF poles
aa = real(aa);                                  % eliminate round-off error

figure(1);
subplot(211);
zplane(b,a);                                    % pole-zero plot of LPF
axis(1.2*[-1 1 -1 1]);
subplot(212);
zplane(bb,aa);                                  % pole-zero plot of BPF
axis(1.2*[-1 1 -1 1]);

N = 1024;                                       % # points on unit circle
FF = -0.5:1/N:0.5-1/N;                          % corresponding freq. axis
H_lpf = freqz(b,a,N,'whole');                   % DFT of LPF
H_bpf = freqz(bb,aa,N,'whole');                 % DFT of BPF

figure(2);
subplot(211);
plot(FF,20*log10(abs(fftshift(H_lpf))));        % plot DFT of LPF
grid on;
xlabel('frequency (cycles/sample)');
ylabel('magnitude (dB)');
set(gca,'XTick',-0.5:0.1:0.5);
axis([-0.5 0.5 -60 3]);

subplot(212);
plot(FF,20*log10(abs(fftshift(H_bpf))));        % plot DFT of BPF
grid on;
xlabel('frequency (cycles/sample)');
ylabel('magnitude (dB)');
set(gca,'XTick',-0.5:0.1:0.5);
axis([-0.5 0.5 -60 3]);
```

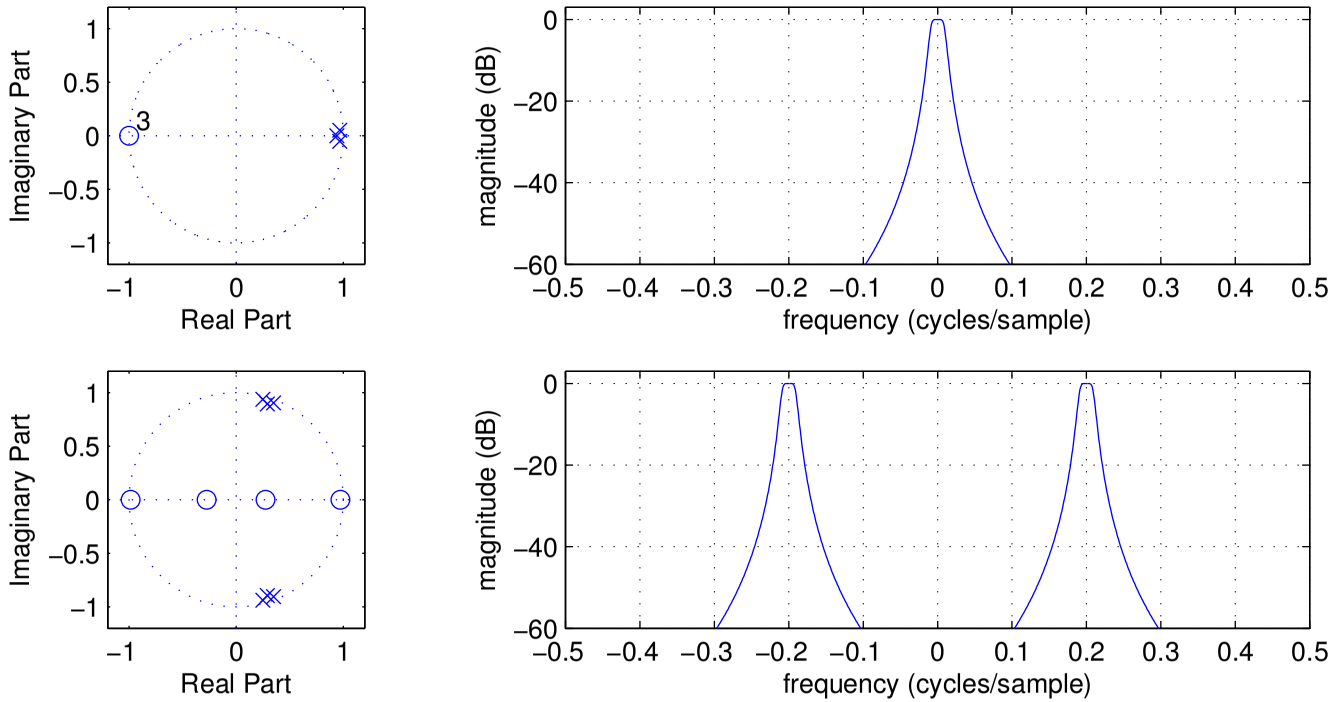

Figure 3: Pole-zero and corresponding DTFT plots: (top) the he 3-rd order Butterworth lowpass filter; (bottom) the 6-th order bandpass filter crated from the lowpass filter.

Task 2 steps:

1. Filter parameters:

   (a) Using the MATLAB® commands `freqz` and `plot`, plot the frequency response of a 6-th order discrete-time Butterworth BPF centered at $\Omega_0 = 2\pi \times 0.2$ rads/sample for bandwidths $W = 2\pi \times 0.01$, $2\pi \times 0.02$, and $2\pi \times 0.05$.

   (b) Using the MATLAB® commands `freqz` and `plot`, plot the frequency response of a discrete-time Butterworth BPF centered at $\Omega_0 = 2\pi \times 0.2$ rads/sample with bandwidth $W = 2\pi \times 0.02$, rads/sample for orders 2, 6, and 10.

   (c) Based on your plots from parts (a) and (b), what are the tradeoffs involving bandwidth and order?

2. Based on the requirement to correctly identify a sinusoid in the received light beam, design ten bandpass filters. Plot the DFTs of the ten filters on the same set of frequency axes and use the plots to justify your design choices.

3. Apply your filters to the data sets.

   (a) Use the same post-BPF processing algorithm from Task 1 to identify the shooter.

   (b) Make adjustments to your filter designs, as necessary, to get the correct answer for each data set. (See the Appendix for how to know what the right answer is.)

4. Extra Credit: Pre-warping the frequency. In the handout "An Introduction to Discrete-Time Filter Design" the process for deriving a discrete-time filter from a continuous-time filter using the bilinear transform is outlined. The explanation stated that the relationship between ω_c rads/s (the corner frequency of the continuous-time filter) and Ω_c rads/sample (the corner frequency of the corresponding discrete-time filter) is $\omega_c = \Omega_c/T_s$. But this is not quite right. (It is true for small ω_c, but becomes less true as ω_c approaches the half sample rate frequency π/T_s.) This extra credit problem asks you to derive the true relationship between ω_c and Ω_c. The starting point is the bilinear transform

$$\mathbf{s} = \frac{2}{T_s}\frac{\mathbf{z}-1}{\mathbf{z}+1}.$$

   The corner frequency in continuous time is $\mathbf{s} = j\omega_c$ whereas the corner frequency in discrete-time is $\mathbf{z} = e^{j\Omega_c}$. Use the substitutions in the bilinear transform to derive the relationship

between ω_c and Ω_c. With this relationship in place, explain how the continuous-time to discrete-time conversion outlined in the handout must be modified.

5. Have the TA check your work and sign off.

Items to include in your lab book:

1. Answer all the questions posed in Tasks 1 and 2. Be sure to include the plots used to justify your answers.

2. Document the order and bandwidth you used for your filters.

3. For each data set, plot (or document) the results of the post-filter processing algorithm corresponding to the outputs of the ten bandpass filters.

4. Include a spectrum plot for each of the data sets and use the plot to justify your claim that your algorithm works.

5. Perform and document complexity comparison between your FIR filter design (Task 1) and your recursive IIR filter design (Task 2). To perform the complexity comparison, calculate the number of additions and multiplications *per input sample* for your Task 1 and Task 2 filter banks. Use these numbers to quantify the computation complexity of the two approaches.

6. Have the TA review your final lab book entries and sign give you a final sign of for Task 2.

Appendix: How to Know What the Right Answer Is

The frequency of the sinusoid contained in one of the data sets may be seen by examining the DTFT of the data sequence. From Table 7-8 (pg. 380 of the text) we know that the DTFT of a sinusoid at a frequency Ω_0 rads/sample is a pair of impulses at $\pm\Omega_0$ rads/sample. Thus, we may answer the question, "how do we know what the right answer is?" by looking for impulses in the DTFT of the data. But how do we compute the DTFT of a given data sequence? The reason this is a question is because the DTFT is a *continuous* function of Ω, and computers are not good at computing continuous functions. Here, we answer the question by considering a sampled version of the DTFT. The sampled DTFT is called the discrete Fourier transform, or DFT.

Let $x[n]$ be a discrete-time sequence with N_x samples and let

$$\mathbf{X}(e^{j\Omega}) = \sum_{n=0}^{N_x-1} x[n]e^{-j\Omega n} \tag{5}$$

be the DTFT of $x[n]$. Because $\mathbf{X}(e^{j\Omega})$ is a continuous function of Ω, it is hard to evaluate it on a computer. But, for a given, fixed value of Ω, it is easy for a computer to evaluate the DTFT for that fixed value of Ω. For this reason, the text introduced the Discrete Fourier Transform (DFT) in Section 7-15. (Figure 7-23, pg. 383, is a nice illustration of this concept.) The DFT is a sampled version of the DTFT:

$$\mathbf{X}(e^{j\Omega_k}) = \mathbf{X}(e^{j\Omega})\big|_{\Omega=\Omega_k} = \sum_{n=0}^{N_x-1} x[n]e^{-j\Omega_k n}. \tag{6}$$

The text uses the notation $\mathbf{X}_k$ for $\mathbf{X}(e^{j\Omega_k})$. So, we have

$$\mathbf{X}_k = \mathbf{X}(e^{j\Omega})\big|_{\Omega=\Omega_k} = \sum_{n=0}^{N_x-1} x[n]e^{-j\Omega_k n}. \tag{7}$$

Given these definitions, what are good choices for the Ω_k? The usual approach is to sample Ω at N equally spaced points in the interval $[0, 2\pi)$. The frequencies are

$$0, \frac{2\pi}{N}, 2\frac{2\pi}{N}, 3\frac{2\pi}{N}, \ldots, (N-1)\frac{2\pi}{N} \tag{8}$$

which means that $\Omega_k = 2\pi k/N$ for $0 \leq k < N$. Because the spacing between each frequency-domain sample is $2\pi/N$ rads/sample, we say this DFT has a *resolution* of $2\pi/N$ rads/sample (or $1/N$ cycles/sample.)

The form of the DFT using N equally spaced samples of Ω is

$$\mathbf{X}_k = \sum_{n=0}^{N_x-1} x[n]e^{-j\pi k/Nn}, \qquad 0 \le k < N. \tag{9}$$

This is easy to evaluate on a computer: given a data sequence $x[n]$, a value for N, and an index $0 \le k < N$, the computations are straight-forward. In most cases, an efficient algorithm known as the "Fast Fourier Transform" (FFT) is used to evaluate the DFT. The name FFT is unfortunate, because the FFT is a fast algorithm for computing the DFT as opposed to some new transform. You don't need to know the details of how the FFT works[1] other than the fact that the order of the FFT output corresponds to (8). Because it is customary to plot the DTFT for $-\pi \le \Omega \le \pi$ (this places the 0-frequency component in the middle of the spectrum), we prefer to plot the $\mathbf{X}_k$ of the DFT in the same order, according to the frequencies

$$\begin{array}{cccccccc} \frac{N}{2}\frac{2\pi}{N} & \left(\frac{N}{2}+2\right)\frac{2\pi}{N} & \cdots & (N-1)\frac{2\pi}{N} & 0 & \frac{2\pi}{N} & \cdots & \left(\frac{N}{2}-1\right)\frac{2\pi}{N} \\ -\frac{N}{2}\frac{2\pi}{N} & \left(-\frac{N}{2}+1\right)\frac{2\pi}{N} & \cdots & -\frac{2\pi}{N} & 0 & \frac{2\pi}{N} & \cdots & \left(\frac{N}{2}-1\right)\frac{2\pi}{N}. \end{array} \tag{10}$$

Creating such a plot requires the reordering of the FFT output samples. The reordering follows the frequency ordering (10). The reordering of the DFT samples produced by the FFT is such a commonly used function, MATLAB® has a built-in function called `fftshift`. The most common use of `fftshift` is to reorder the FFT output to following the ordering (10).

Now that the question about good choices for Ω_k has been answered, the question is now, what are good values for N? One thing we could do make the DFT length the same as the length of the data. Here, we have $N = N_x$. An example using $N = N_x$ applied to the data contained in the file `lab5test.mat` is illustrated in Figure 4. Here, N_x = 20,000 and the resolution of the DFT is $1/20000 = 5 \times 10^{-5}$ cycles/sample. The DC component is clearly visible at $F = 0$ as well as two spectral lines at $F = \pm 0.2$ cycles/samples (or, if we were using the quantized Ω axis, at $\Omega = \pm 0.4\pi$ rads/sample). This shows that the frequency of the sinusoid contained in the file is 0.2 cycles/sample. This plot was created by the following segment of MATLAB® code:

[1] For those of you interested in how the FFT works, see Section 7-16 of the text. If that is not enough, consider enrolling in ECEn 487.

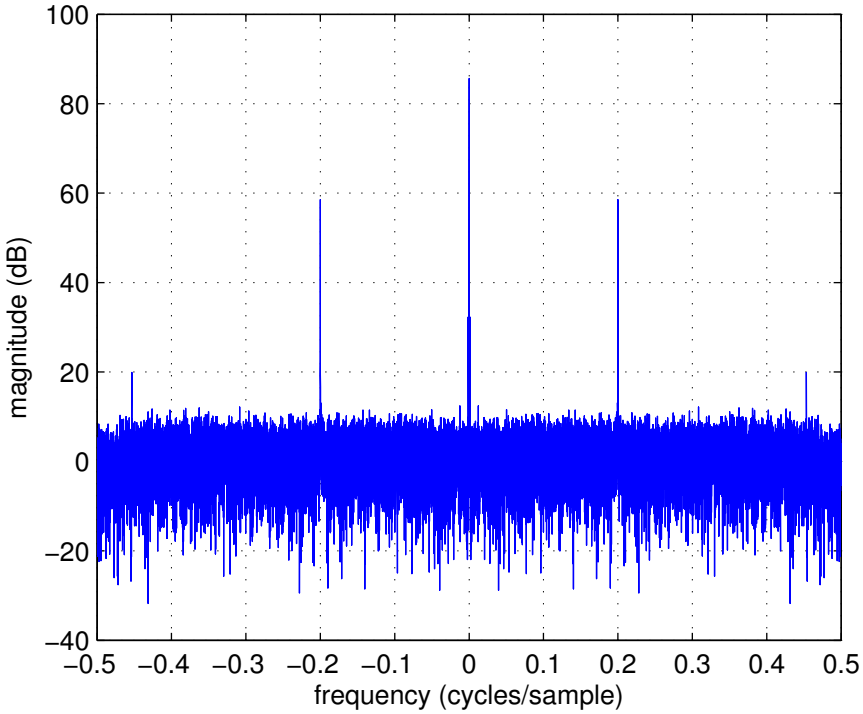

Figure 4: An example of the DFT for the data in file `lab5test.mat`. Because the data comprises 20,000 samples, this is a length-20,000 DFT.

```
load lab5test.mat                          % the file contains the vector x
Nx = length(x)                             % length of x
X = fft(x);                                % use the FFT algorithm to compute
                                             the length-Nx DFT of x
FF = -0.5:1/Nx:0.5-1/Nx;                   % frequency axis for the plot
plot(FF,20*log10(abs(fftshift(X))));       % create the plot, note the use of
grid on;                                   % fftshift to reorder X to align
xlabel('frequency (cycles/sample)');       % with the frequencies in FF.
ylabel('magnitude (dB)');
```

Sometimes N_x is too big for this approach to be practical. If we set N to a reasonable value, then we are faced with the issue of how to use *all* of the data samples in computing the spectrum. The Welch periodogram method solves this problem by partitioning the data into length-N segments, computing the length-N DFT of each segment, then averaging the squared magnitudes of the DFTs of each segment. The process is illustrated in Figure 5. This approach has the added benefit that the averaging "cleans up" variations in the spectrum due to noise perturbations in the received samples.[2] In this case, the Welch periodogram method trades resolution for "signal-to-noise ratio."

[2]It is often the case (and this example is one of them) that the samples contain a desired signal plus noise.

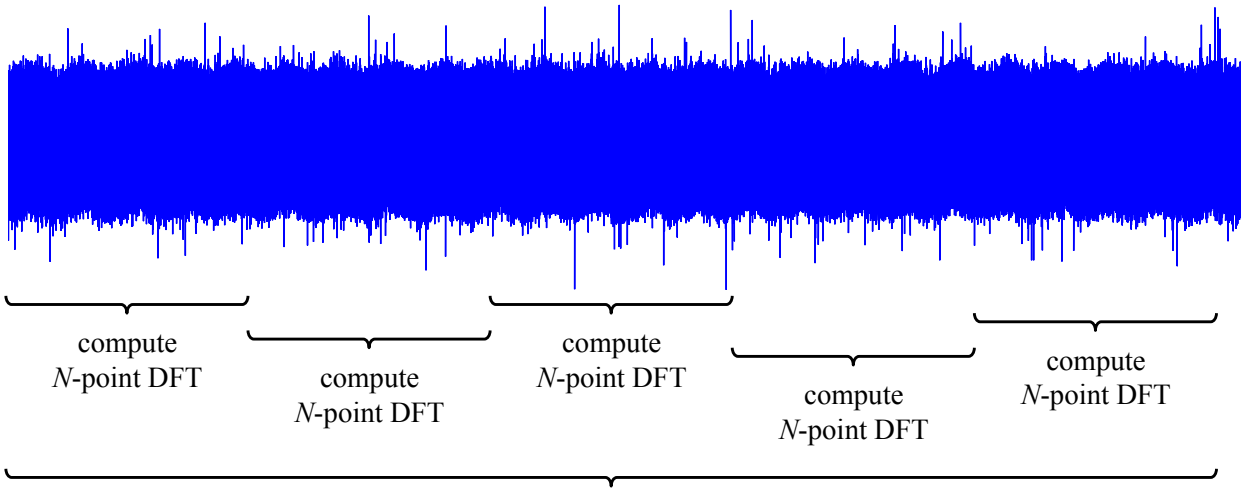

Figure 5: An illustration of the Welch Method for averaging periodograms to estimation the spectrum of a signal. The plot is a time-domain plot of the data in the file `lab5test.mat`.

The result of the Welch periodogram method for $N = 1000$ applied to the data set in file `lab5test.mat` is shown in Figure 6. The Welch periodogram method was based on length-1000 DFTs. Because the data comprises 20,000 samples and the periodograms were computed on non-overlapping sections of length 1000, the resulting spectrum is based on the average of the square magnitudes of 20 length-1000 DFTs. As before, the spectrum shows a line at DC ($F = 0$) and a pair of lines at $F = \pm 0.2$ cycles/sample. This shows that the frequency of the sinusoid is 0.2 cycles/sample. Relative to the spectrum shown in Figure 4, the Welch method produces a "cleaner" estimate of the signal spectrum. The segment of MATLAB® code that produced this plot is shown below. The code segment uses the built-in function `pwelch` to perform the Welch periodogram averaging.

```
load lab5test.mat                          % the file contains the vector x
N = 1000;                                  % length of the DFTs
X = pwelch(x,rectwin(N),0,N,'twosided'); % Welch periodogram method

FF = -0.5:1/N:0.5-1/N;                     % frequency axis for the plot
plot(FF,10*log10(abs(fftshift(X))));       % create the plot, note the use of
grid on;                                   % fftshift to reorder X to align
xlabel('frequency (cycles/sample)');       % with the frequencies in FF.
ylabel('magnitude (dB)');
```

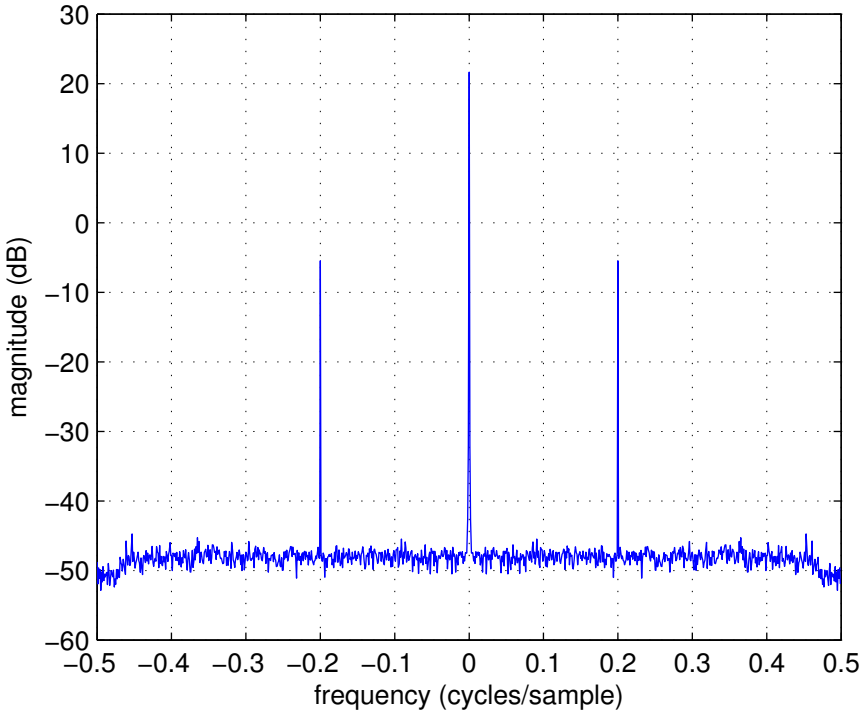

Figure 6: An result of the Welch periodogram method applied to the data in file `lab5test.mat` (cf., Figure 4). The method was based on length-1000 DFTs.