Music 421A
Winter 2011-2012
Homework #2
Windows and Fourier Theorem Review
Due in one week

Theory Problems

1. (4 pts) Consider the 8 point DFT of some arbitrary input, where the sampling rate is 3 Hz. Write the frequency in Hz represented by every point of the DFT.

   Solution: Recall the unit circle drawing from class. The first frequency sample is always for zero frequency, and the others are uniformly spaced on the unit circle. This leads to:
   0; $1 \times 3/8$; $2 \times 3/8$; $3 \times 3/8$; $4 \times 3/8$; $5 \times 3/8$; $6 \times 3/8$; $7 \times 3/8$
   If we consider frequencies from $-\pi$ to π we have:
   0; $1 \times 3/8$; $2 \times 3/8$; $3 \times 3/8$; $\pm 4 \times 3/8$; $-3 \times 3/8$; $-2 \times 3/8$; $-1 \times 3/8$;
   Either set of answers is acceptable, but the order within an answer is not negotiable. The first sample must be zero.

2. (5 pts) For the sequence $x(n) =$ `[1:8,8:-1:1]` (matlab notation), sketch or plot $\text{STRETCH}_2(x)$ and its magnitude spectrum.

   Solution: The effect of $\text{STRETCH}_2(x)$, inserting zeros at every other sample, is a compression in frequency domain by a factor of 2, and also a repeating spectral image over the remaining frequency axis. (Recall the discussion in SASP Appendix D.10) The plots are shown in Figures 1 and 2.

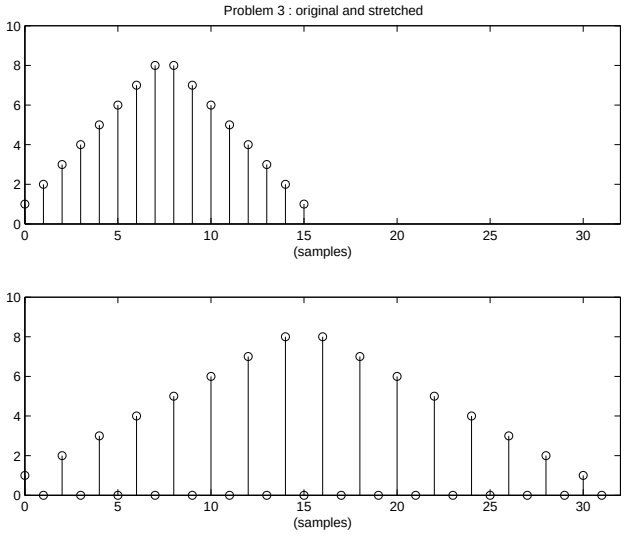


Figure 1: The original signal sequence (top) and its "stretched-by-two" version (bottom)

```
x=[1:8,8:-1:1];
xp=zeros(1,length(x)*2);
xp(1:2:end)=x;
subplot(211);stem(0:length(x)-1,x)
```

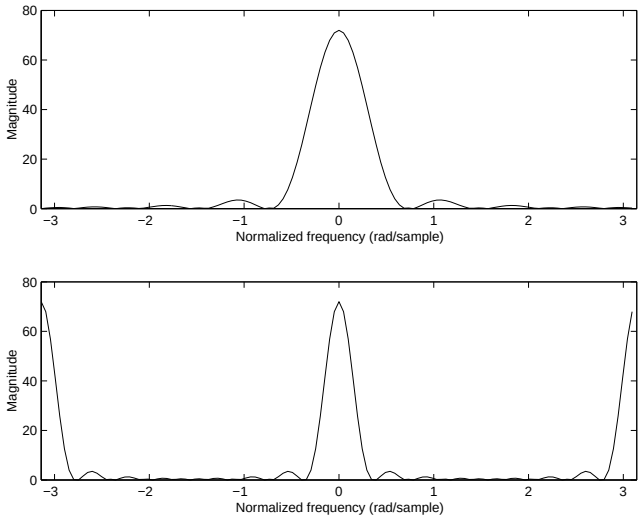


Figure 2: The original signal spectrum (top) and its "stretched-by-two" version (bottom)

```
subplot(212);stem(0:length(xp)-1,xp)
subplot(211);axis([0 32 0 10])
subplot(212);axis([0 32 0 10])
Nfft=128;
subplot(211);plot([0:2*pi/Nfft:2*pi*(1-1/Nfft)]-pi,fftshift(abs(fft(x,Nfft))));
axis([-pi pi ylim];
subplot(212);plot([0:2*pi/Nfft:2*pi*(1-1/Nfft)]-pi,fftshift(abs(fft(xp,Nfft))));
axis([-pi pi ylim];
```

3. (5 pts) You are given a windowed sinusoid of the following form:

$$x(t) = w(t)\cos(\omega t + \phi).$$

The window function $w(t)$ and sinusoidal phase ϕ are unknown. Show how you can remove the sinusoidal component $\cos(\omega t + \phi)$ in two steps to recover the unknown window. For the first step, use properties of sinusoids to convert $x(t)$ into an analytic signal $z(t)$ (having only positive or only negative frequencies, but not both). Once in analytic form, use common signal processing operators to "separate" or demodulate the window function from the sinusoid. Run a small simulation in matlab showing the procedure.

Solution:

Step 1 (3 points)
Any real sinusoid $w(t)\cos(\omega t+\phi)$ may be converted to a positive-frequency complex sinusoid $w(t)\exp[j(\omega t+\phi)]$ by simply generating a phase-quadrature component $w(t)\sin(\omega t + \phi)$ to serve as the "imaginary part" resulting in:

$w(t)\exp[j(\omega t + \phi)] = w(t)(\cos(\omega t + \phi) + j\sin(\omega t + \phi))$.

The phase-quadrature component can be generated from the in-phase component by a simple quarter-cycle time shift. For this simplified situation, we can assume that this is achievable and the analytic signal $z(t) \approx w(t)\exp[j(\omega t + \phi)]$.

The analytic signal can be also obtained by replacing the negative frequency components of frequency resposne of the given signal with zero and multiply with 2.

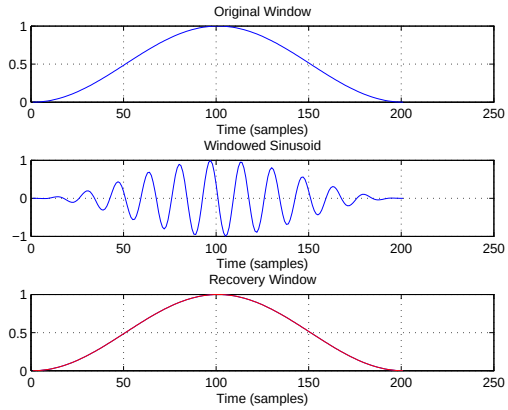


Figure 3: Window Recovery

For more complicated signals, a *filter* can be constructed which shifts each sinusoidal component by a quarter cycle. This is called a *Hilbert transform filter*. Ideally, this filter has magnitude 1 at all frequencies and instroduces a phase shift of $-\pi/2$ at each positive frequency and $+\pi/2$ at each negative frequency.

Step 2 (2 points)
Once the sinusoid is converted to a positive-frequency complex sinusoid $z(t) = w(t)\exp[j(\omega t + \phi)]$, we can demodulate or recover the windowed sinusoid by simply taking the absolute value of the analytic signal:

$|z(t)| \approx |w(t)\exp[j(\omega t + \phi)]| \approx |w(t)|$.

For those interested, please see
`https://ccrma.stanford.edu/~jos/mdft/Analytic_Signals_Hilbert_Transform.html`.

Please see Fig. 3 and matlab code below.

```
clear all; close all; clc;

% sampling rate
fs = 100;
% sample frequency
f = 6;
% time axis
t = 0:fs*2;
% generate the window
win = hann(length(t));
% sample phase
phi = pi/2;
% generate the tones
xcosw = cos(2*pi*f.*t./fs + phi).*win';
xsinw = sin(2*pi*f.*t./fs + phi).*win';

% plot
subplot(3,1,1)
plot(win); title('Original Window'); grid on; xlabel('Time (samples)')
```

```
subplot(3,1,2)
plot(xcosw)
title('Windowed Sinusoid'); grid on; xlabel('Time (samples)')


% Compute the Hilbert Transform
X = fft(xcosw);
N = length(X);
Z = 2*X;
Z(ceil(N/2):end) = 0;
z = ifft(Z);
%z = xcosw + j*xsinw;

subplot(3,1,3)
plot(abs(z))
hold on; xlabel('Time (samples)')
plot(win,'r'); title('Recovery Window'); grid on; ylim([0 1]);
```

4. (5 pts) Derive the expression and sketch the window $w(n)$ corresponding to the window transform

$$W(\omega) = \frac{1}{2}\text{asinc}_M(\omega + \omega_1) + \text{asinc}_M(\omega) + \frac{1}{2}\text{asinc}_M(\omega - \omega_1)$$

where $\omega_1 = 2\pi/M$.

Solution: From our standard transform of a rectangular window of length M, but here, with DC gain normalized to unity,

$$R_M(n) \longleftrightarrow \text{asinc}_M(\omega)$$

where

$$R_M(n) = \begin{cases} \frac{1}{M}, & -\frac{(M-1)}{2} \leq n \leq \frac{(M-1)}{2} \\ 0, & \text{otherwise} \end{cases}$$

and the shift theorem,

$$x(n) \cdot \mathrm{e}^{j\omega_1 n} \longleftrightarrow X(\omega - \omega_1)$$

we have,

$$\begin{aligned} w(n) &= \frac{1}{2}R_M(n)\mathrm{e}^{-j\omega_1 n} + R_M(n) + \frac{1}{2}R_M(n)\mathrm{e}^{j\omega_1 n} \\ &= [\cos(\omega_1 n) + 1]R_M(n) \\ &= [\cos(2\pi\, n/M) + 1]R_M(n) \end{aligned}$$

The plots are shown in Figure 4.

5. (5 pts) Sketch the window $w(n)$ corresponding to the window transform

$$W(\omega) \triangleq -\frac{1}{2}\text{asinc}_M(\omega + \omega_1) + \text{asinc}_M(\omega) - \frac{1}{2}\text{asinc}_M(\omega - \omega_1)$$

where $\omega_1 = 2\pi/M$ and M is assumed to be odd.

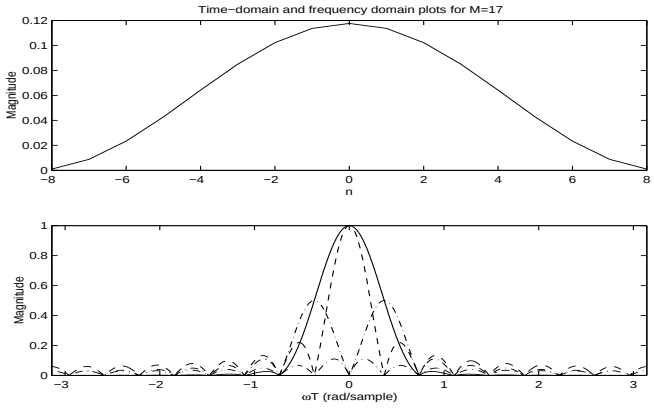


Figure 4: The time domain(top) and magnitude of frequency response(bottom, superimposed by its rectangular window transform components) of the window given in problem 5, for M=17

Solution: (5 pts) Because the two shifted asincs are negated, the time-domain window is causal (instead of zero-centered). We have

$$\begin{aligned} w(n) &= -\frac{1}{2}R_M(n)\mathrm{e}^{-j\omega_1 n} + R_M(n) - \frac{1}{2}R_M(n)\mathrm{e}^{j\omega_1 n} \\ &= [1-\cos(\omega_1 n)]R_M(n) \\ &= [1-\cos(2\pi\, n/M)]R_M(n) \\ &= [2\sin^2(\pi\, n/M)]R_M(n) \end{aligned}$$

where

$$R_M(n) = \begin{cases} \frac{1}{M}, & -\frac{(M-1)}{2} \le n \le \frac{(M-1)}{2} \\ 0, & \text{otherwise} \end{cases}$$

The plots are shown in Figure 5. The window is non-negative as all windows should be.

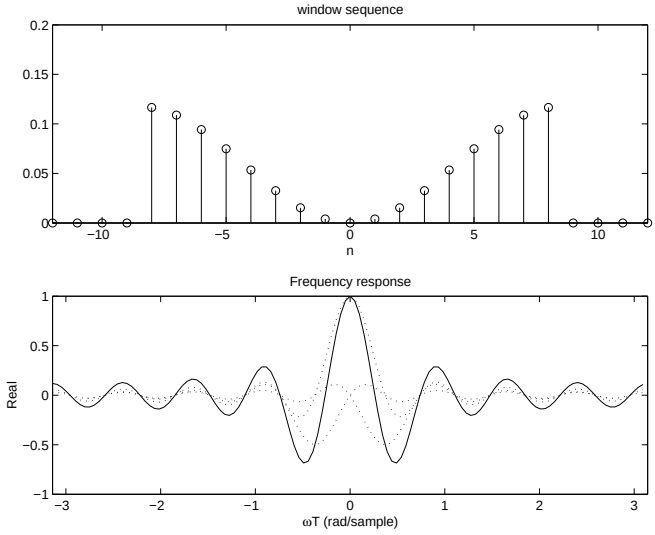


Figure 5: The window sequence and its magnitude response for this problem.

6. (5 pts) Sketch the window $w(n)$ corresponding to the window transform

$$W(\omega) \triangleq e^{j\omega/2}\text{asinc}_M(\omega)$$

where M is an even integer and $\omega \in [-\pi, \pi)$. [Hint: The resulting window is called a "DFT-even" window.]

Solution:

By applying IDTFT,

$$\begin{aligned}
w(n) &= \frac{1}{2\pi}\int_{-\pi}^{\pi} e^{j\omega/2}\frac{\sin(\omega M/2)}{M\sin(\omega 2)}e^{j\omega n}d\omega \\
&= \frac{1}{2\pi M}\int_{-\pi}^{\pi} e^{j\omega/2}\frac{e^{j\omega M/2}-e^{-j\omega M/2}}{e^{j\omega/2}-e^{-j\omega/2}}e^{j\omega n}d\omega \\
&= \frac{1}{2\pi M}\int_{-\pi}^{\pi} e^{j\omega M/2}\frac{1-e^{-j\omega M}}{1-e^{-j\omega}}e^{j\omega n}d\omega \\
&= \frac{1}{2\pi M}\int_{-\pi}^{\pi} e^{j\omega M/2}(\sum_{k=0}^{M-1} e^{-jk\omega})e^{\omega n}d\omega \\
&= \sum_{k=0}^{M-1}\frac{1}{2\pi M}\int_{-\pi}^{\pi} e^{j(M/2-k+n)\omega}d\omega \\
&= \frac{1}{2\pi M}\sum_{k=0}^{M-1}\frac{e^{j(M/2-k+n)\pi}-e^{-j(M/2-k+n)\pi}}{j(M/2-k+n)} \\
&= \frac{1}{M}\sum_{k=0}^{M-1}\frac{\sin((M/2-k+n)\pi)}{(M/2-k+n)\pi} \\
&= \frac{1}{M}\sum_{k=0}^{M-1}\text{sinc}(M/2-k+n)
\end{aligned}$$

For sample index n, $w(n)$ ends up with,

$$w(n) = \begin{cases} \frac{1}{M}, & n = -M/2, \ldots, M/2-1 \\ 0, & \text{otherwise} \end{cases}$$

Lab Assignment

1. In this problem you will write two matlab functions, and at least the second will be useful later in the course. Turn in a copy of your code for each part.

   (a) (3 pts) Write a matlab function that windows and zero-pads an input signal. Your function should take 3 arguments: input signal x to be windowed, window w to be used, and final length N of the zero padded sequence. The code below will get you started:

   ```
   function [output_sig] = zeropadwin(input_sig, window, padded_length)
   % function windows a signal and adds zeros to the end
   % check that input_sig is the same size vector as the window:

   % multiply window by input_sig:

   % add zeros to end so that length(output_sig) is padded_length:
   output_sig =
   ```

   (b) (5 pts) Write a matlab function that **zero-phase** windows and zero-pads an input signal. This is the same as the previous problem, except you must be careful about the order of the output. The output should look similar to Figure 2.6(b) of SASP (p. 30 in the March 2009 draft).

   Solution: The code below takes care of things as do other versions. The important things in part (a) are that the input signal and window are the same size (or transpose correctly if same length but not same size), and that the output is the right length with zeros in the right place, and the same row / column status as the input. In part (b), you must do everything as in part (a) but you must also check that the input window and signal are odd length, and load the buffer correctly.

   (a) (3pts)

   ```
   function [output_sig] = zeropadwin(input_sig, window, padded_length)
   % 420 hw2q1a
   % function [output_sig] = zeropadwin(input_sig, window, padded_length)
   % windows a signal and adds zeros to the end
   % check that input_sig is the same size vector as the window:
   if sum(size(input_sig) ~= size(window)) ...
      | min(size(window))~= 1 ...
      | length(size(window)) ~= 2,
   error('input_sig and window must be matching-dimension vectors');
   end  % NOTE: code checking length (instead of size)
        % also OK, but then must ALSO
        % transpose in step below if mismatch
        % OR: must refuse to allow either column or row vectors
        % if not checking
   % multiply window by input_sig:
   temp = input_sig.*window;
   % add zeros to end so that length(output_sig) is padded_length:
   % add zeros in column or row as appropriate:
   if size(input_sig,1) > size(input_sig,2) % column
   output_sig = [temp; zeros(padded_length-length(input_sig),1)];
   else
   output_sig = [temp, zeros(1,padded_length-length(input_sig))];
   end
   ```

(b) (5pts)

```
function [output_sig] = zpzpwin(input_sig, window, padded_length)
% 420 hw2q1b
% function [output_sig] = zpzpwin(input_sig, window, padded_length)
% zero phase windows a signal and adds zeros to the end
% check that input_sig is the same size vector as the window:
if sum(size(input_sig) ~= size(window)) ...
   | (length(input_sig)+1)/2 ~= round((length(input_sig)+1)/2) ... check odd
   | min(size(window))~= 1 ...
   | length(size(window)) ~= 2,
error('input_sig and window must be odd length matching-dimension vectors');
end  % NOTE: code checking length (instead of size)
     % also OK, but then must ALSO
     % transpose in step below if mismatch
     % OR: must refuse to allow either column or row vectors
     % if not checking
% multiply window by input_sig:
temp = input_sig.*window;
% add zeros to end so that length(output_sig) is padded_length:
% add zeros in column or row as appropriate:
if size(input_sig,1) > size(input_sig,2) % column
output_sig = [temp((length(input_sig)+1)/2:end); ...
      zeros(padded_length-length(input_sig),1); ...
      temp(1:(length(input_sig)-1)/2)];
else
output_sig = [temp((length(input_sig)+1)/2:end),  ...
              zeros(1,padded_length-length(input_sig)), ...
      temp(1:(length(input_sig)-1)/2)];
end
% now important last step:  shift zeros to center
```

2. In this problem, we consider the question of what happens to the spectrum of a window when we change its length and/or sampling rate. When plotting magnitude spectra in this problem, always normalize for a peak value of 0 dB, and always use `fftshift()` so that frequency 0 is in the center of the plot.

   (a) (2 pts) Plot, using `stem()`, the time domain representation of a 51-sample rectangular window, zero-phase zero-padded to length 256. Double label the time axis in both samples and seconds, assuming a sampling rate of 8192 Hz. How long is your window in seconds?

   (b) (2 pts) Plot the log-magnitude spectrum of the window, and label the frequency axis in Hz. Manually (or using matlab) label the frequency of the first positive zero crossing in Hz.

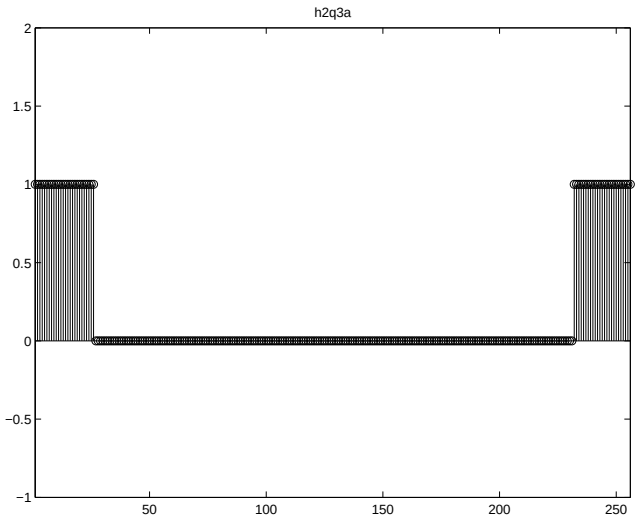
   (c) (2 pts) Change the window length to 91 samples and repeat the previous item. How long is this window in seconds?

   (d) (2 pts) Using the window length 91 samples, change the sampling rate to 8000 Hz. How long in seconds is the window now? Plot the log-magnitude spectrum of the window, and label the frequency axis in Hz. Manually label the frequency of the first positive zero crossing in Hz.

   Solution: For this problem, when graphical and theoretical values are given, either is valid.

   (a) Code below works:

   ```
   % part a:
   ```




```
M = 51;
N = 256;
win = (zpzpwin(ones(M,1),boxcar(M),N));
stem(win);
axis([1 256 -1 2]);
title('h2q3a')
```

See plot. The samples should be strictly as in the plot, meaning that the "center" sample for the rect window must be the first sample shown. The time in seconds should be labeled as follows:

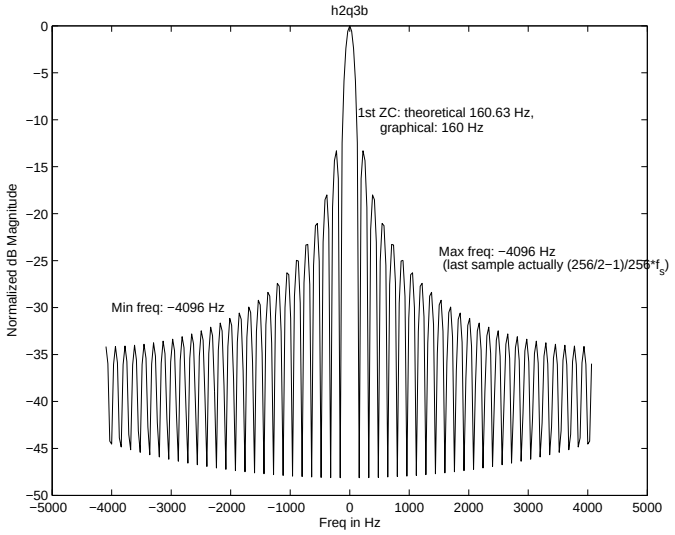
sample index	corresponding time
1 through 26	0 through 3.05 ms
27 through 231	(No time label needed)
232 through 256	-3.05 ms through -.122 ms

The window is 51/8192 = 6.2ms long. (the rest of the samples are zero padding for spectral interpolation, not to be understood as additional time length).

(b)
```
% part b:
f_s = 8192;
WIN = fft(win);
freqs = -f_s/2:(f_s/N):((N/2-1)/N)*f_s;
plot(freqs,fftshift(20*log10(abs(WIN)./max(abs(WIN)))));
% label freq axis in Hz:
text(-4000,-30,'Min freq: -4096 Hz')
text(1500,-25,sprintf('Max freq: -4096 Hz \n
(last sample actually (256/2-1)/256*f_s)'))
text(135,-10,sprintf('1st ZC: theoretical 160.63 Hz, \n
        graphical: 160 Hz'))
xlabel('Freq in Hz');
title('h2q3b');
ylabel('Normalized dB Magnitude');
```

(c) Repeat above, but with with `M=91`. The window is now 9.89(91/81) ms long. Notice that the main lobe is now narrower as we expect from a longer window.

(d) Use a similar approach, just changing f_s. The window length is now 10.13 ms.



3. (8 pts) Repeat Problem 2 for a Hann window. Compare and contrast the main-lobe width and side-lobe level for analogous plots.

   Solution: The approach is entirely similar to Problem 2, but using `hann()` (or other Hann window definition, see below) in all cases where `boxcar()` was used. In all analogous parts of this problem with the above, the time domain length is the same, but the main lobe width is twice as wide, and the Hann sidelobes are -31 dB down (then falling off as -18 dB per octave) rather than 13 dB down and 6dB per octave for the rect.

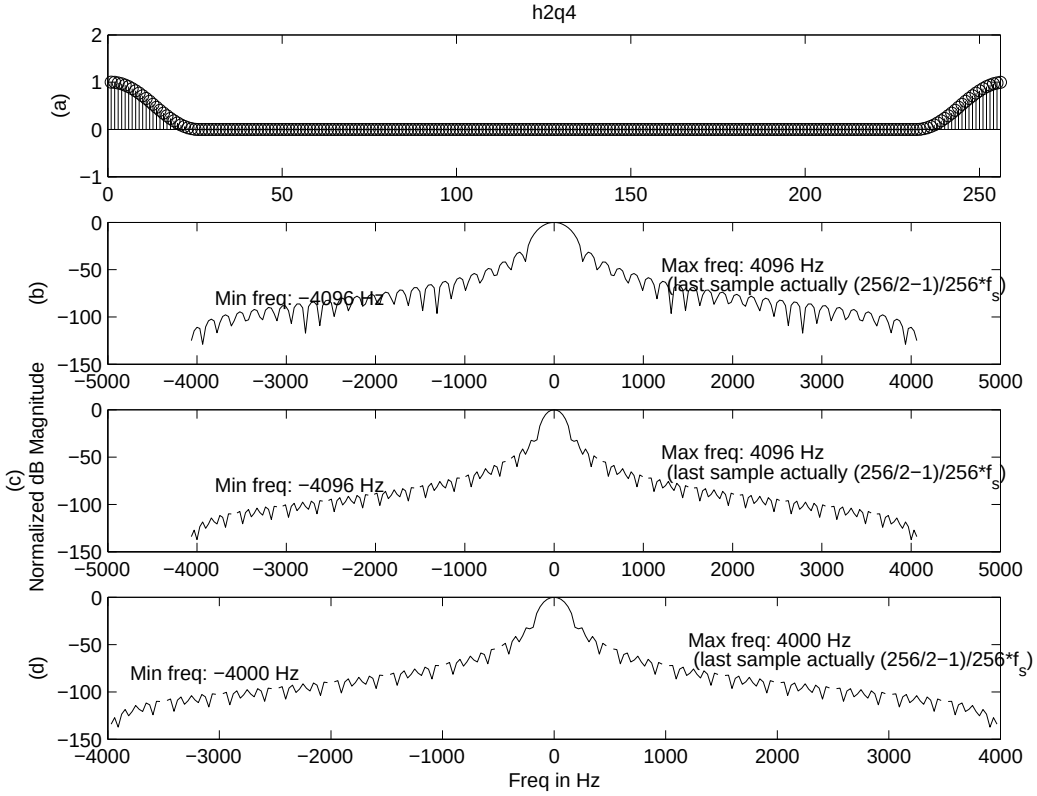
   Depending on the definition of the Hann window used in this problem, you will get different locations of the first zero crossing. The table below shows what corresponds to what. Any consistent answers will be accepted, as per the chart below.

definition	first ZC in Hz	(b)	(c)	(d)
`hann(M)`	$2f_s/(M-1)$	327.68	204.8	200
	$2f_s/M$	321.25	202.27	197.53
`hanning(M)`	$2f_s/(M+1)$	315.08	199.80	195.12

   Margins of error are acceptable for graphic solutions due to the frequency interpolation required by N = 256. Specifically, the distance between bins is 8192/256 = 32 Hz for parts (b) and (c), so the maximum graphical solution error is 16 Hz. Similarly, for part (d), the maximum graphical error is (1/2)*8000/256 = 15.63 Hz.

   In the plot shown for this problem, `hann(M)` was used in matlab.

   No points should be taken off if people confuse the signs on how many "dB down" a sidelobe level (SLL) is. For complete accuracy, however, we note that the proper terms used as "the sidelobe is 13 dB down" or "the sidelobe level is -13 dB."


h2q4
(a)
(b)
(c)
(d)
Normalized dB Magnitude
Min freq: −4096 Hz
Max freq: 4096 Hz
(last sample actually (256/2−1)/256*f_s)
Min freq: −4096 Hz
Max freq: 4096 Hz
(last sample actually (256/2−1)/256*f_s)
Min freq: −4000 Hz
Max freq: 4000 Hz
(last sample actually (256/2−1)/256*f_s)
Freq in Hz