Music 421A
Winter 2011-2012
Homework #5
Spectral Peaks + Noise
Due in one week

Theory Problems

1. (10 pts) Suppose we are going to analyze a sinusoid whose frequency is $f_0 = 440$ Hz using a rectangular window. Assume the window length is $M = 255$ and the sampling rate is $f_s = 8192$ Hz.

   (a) (3 pts) What is the relative error (*i.e.,* $\Delta f/f_0 = |f_0 - \hat{f}|/f_0$) between the actual frequency (f_0) of the sinusoid and the peak frequency ($\hat{f}$) when there is no zero-padding?

   (b) (3 pts) Repeat part (a) with the zero-padding factor of 5.

   (c) (4 pts) Repeat part (a) with the zero-padding factor of 5 and parabolic interpolation using the peak and its two neighbors.

   Solution:

   (a) The kth bin frequency is given by $f_k = k \cdot f_s/M$, and the 14th bin frequency $f_{14} = 449.76$ Hz is closest to $f_0 = 440$ Hz. Therefore, $\varepsilon = \Delta f/f_0 = |f_0 - \hat{f}|/f_0 = 9.76/440 \approx 2.218\%$.

   (b) After zero-padding by a factor of 5, the effective window length N is now $N = 5M = 1275$. Therefore, the closest bin to f_0 is now the 68th bin, whose frequency is 436.91 Hz, and the error is $\varepsilon = \Delta f/f_0 = |f_0 - \hat{f}|/f_0 = 3.09/440 \approx 0.70\%$.

   (c) From part (b), we know that the peak is at the 68th bin, and thus we will use it together with its two neighbors (67th and 69th bin) for parabolic interpolation. Since the magnitudes of the DFT in dB at those bins are $X(67) = 40.84 = \alpha, X(68) = 42.0299 = \beta$, and $X(69) = 42.0213 = \gamma$, the interpolated peak location p is given by $p = \frac{1}{2}\frac{\alpha+\gamma}{\alpha-2\beta+\gamma} \approx 0.4928$. Therefore, the final frequency estimate is $\hat{f} = (k^* + p)f_s/N \approx 440.07$ Hz, and the relative error is $\varepsilon = \Delta f/f_0 = |f_0 - \hat{f}|/f_0 = 0.07/440 \approx 0.016\%$. Remember that the peak magnitudes in dB result in a better frequency estimate.

```
fs = 8192;
f0 = 440;
M = 255;
n =0:M-1;
x = [cos(2*pi*f0*n/fs) zeros(1,M*4)];
xf = fft(x);
xf = 20*log10(abs(xf));
plot(xf)
[v,i] = max(xf);
a = xf(i-1);
b = xf(i);
c = xf(i+1);
p = (a-c)/(a+c-2*b)/2;
((i + p-1)/M/5*fs - f0)/f0*100 % 0.016
```

2. (7 pts) **Least Squares Sinusoidal Parameter Estimation**

(a) (5 pts) Use the *orthogonality principle* to construct an optimal least-squares estimator for the amplitude A and phase ϕ of a sinusoid having known frequency ω_0 in the presence of another sinusoid at some unknown frequency. That is, the measurements consist of N samples of the following signal

$$x(n) = \mathcal{A}e^{j\omega_0 n} + s(n)$$

where $\mathcal{A} = A\exp(j\phi)$ and $s(n)$ is an unknown *interferer* (sinusoid at some other unknown frequency ω_i, phase ϕ_i, and amplitude A_i).

Solution: Same derivation as sinusoid with known frequency in white noise, as worked in class.

(b) (2 pts) Under what conditions is the estimate $\hat{\mathcal{A}}$ unbiased?

Solution: $\omega_i = K2\pi/N$, k=1,2,. . ., $A_i = 0$

3. (2 pts) Let $e(n)$ denote a sample of white noise. Then its power spectral density $S_e(\omega)$ is constant from $-\pi$ to π. By the stretch/repeat DTFT theorem, the signal $x = \text{STRETCH}_2(e)$ also has a constant power spectral density, yet every other sample is constrained to be zero. Is $x(n)$ also white noise? Explain.

Solution: The stretched sequence is not white noise because it is not *stationary.* That is, its statistics are not time invariant. Every other sample is zero, which means that the variance, for example, is zero at all odd sample times n and nonzero at even sample times. A stationary random process cannot have a time-varying variance.

4. (8 pts) Find the (cyclic) unbiased autocorrelation of the following sequences in $\mathbf{R}^8$:

$$\begin{array}{llll} (a) & [1,0,0,0,0,0,0,0] & (b) & [1,1,0,0,0,0,0,0] \\ (c) & [1,1,1,1,0,0,0,0] & (d) & [1,1,1,1,1,1,1,1] \end{array}$$

Solution: (6 pts) To do this problem, we just plug in the given sequences into the cyclic autocorrelation function. This can be implemented in matlab as follows. (You need not do this in matlab, but your answer should be equivalent to using (2.8).) Note that in the result quoted, the first column value corresponds to lag zero, and the last column value to lag N. Other ordering is acceptable as long as you clearly specify correct corresponding lag values.

```
vrows = [1 0 0 0 0 0 0 0
         1 1 0 0 0 0 0 0
         1 1 1 1 0 0 0 0
         1 1 1 1 1 1 1 1];
clear r;
for part = 1:4,
    v = vrows(part,:);
    N = length(v);
    for lag = 0:N-1,
        runsum = 0;
        for n = 0:N-1,
            idx = n+1;
            runsum = runsum + conj(v(idx))*v(mod(idx+lag-1,N)+1);
        end
        ridx = lag+1;
        r(part,ridx) = runsum;
    end
end
% output produced is as follows:
% r =
%      1     0     0     0     0     0     0     0   % 3.3.1
%      2     1     0     0     0     0     0     1   % 3.3.2
%      4     3     2     1     0     1     2     3   % 3.3.3
%      8     8     8     8     8     8     8     8   % 3.3.4
```

At the end you need to divide your output by $N = 8$ to make the quantities given "normalized" (unbiased). Such answers should receive full credit.

5. (5 pts) Spectrum analysis of a weighted moving average process (FIR filter). To smooth or average out a time sequence, we may use a *weighted moving average.* Given the discrete time triangular filter $h(t)$

$$h(t) = \begin{cases} 4 - |t|, & |t| < 4 \\ 0, & |t| \geq 4 \end{cases},$$

we need to investigate the frequency response so as to characterize how the filter will respond to a noisy input sequence.

(a) (2 pts) Compute the acyclic biased autocorrelation of $h(t)$.

(b) (3 pts) Compute the biased power spectral density of $h(t)$.

Solution:

(a) (2 pts) The acyclic autocorrelation can be computed directly as the convolution of the filter with itself (as the signal is symmetric about $t = 0$. Written out, we see

$$\begin{aligned}
(h \star h)(\pm 0) &= 1^2 + 2^2 + 3^2 + 4^2 + 3^2 + 2^2 + 1^2 = 44 \\
(h \star h)(\pm 1) &= 1 \cdot 2 + 2 \cdot 3 + 3 \cdot 4 + 4 \cdot 3 + 3 \cdot 2 + 2 \cdot 1 = 40 \\
(h \star h)(\pm 2) &= 1 \cdot 3 + 2 \cdot 4 + 3 \cdot 3 + 4 \cdot 2 + 3 \cdot 1 = 31 \\
(h \star h)(\pm 3) &= 1 \cdot 4 + 2 \cdot 3 + 3 \cdot 2 + 4 \cdot 1 = 20 \\
(h \star h)(\pm 4) &= 1 \cdot 3 + 2 \cdot 2 + 3 \cdot 1 = 10 \\
(h \star h)(\pm 5) &= 1 \cdot 2 + 2 \cdot 1 = 4 \\
(h \star h)(\pm 6) &= 1 \cdot 1 \\
\textit{otherwise} &= 0
\end{aligned} \tag{1}$$

This can also be computed in matlab via `conv(h, h)`.

(b) (3 pts) The power spectral density of the weighted moving average filter can be computed by first noticing that the weighting moving average filter is a triangular window centered on 0 lag. This can be thought of as the convolution of two rectangular windows, each of length 4.

Knowing the DTFT of $\text{rect}_4(t)$ is $4 \cdot \text{asinc}_4(t)$, the DTFT of $h(t)$ is then $4^2 \cdot \text{asinc}_4(t)^2$, by the convolution theorem. Then again using the convolution theorem, the power spectral density is then

$$\text{DTFT}_w(h \star h) = 4^4 \cdot \text{asinc}_4(w)^4. \tag{2}$$

Lab Assignments

1. (20 pts) **Spectral Peak Estimation**

(a) (10 pts) Construct a program to find the frequencies (in Hz) and the magnitudes (in linear amplitude) of the positive-frequency peaks of an input DFT of a given signal, using the code skeleton shown below. Be sure to convert matlab frequency index numbers to frequencies in Hz.

```
function [peaks,freqs]=findpeaks(Xwdb,maxPeaks,fs,win,N)
% peaks = a vector containing the peak magnitude estimates (linear) using
%         parabolic interpolation in order from largest to smallest peak.
% freqs = a vector containing the frequency estimates (Hz) corresponding
%         to the peaks defined above
% Xwdb  = DFT magnitude (in dB scale) vector of a windowed signal.
%         NOTE that it may contain
%         only low-frequency (length < N/2+1), positive-frequency
%         (length = N/2+1), or all (length = N) bins of the FFT.
% maxPeaks = the number of peaks we are looking for
% fs       = sampling frequency in Hz
% win      = window used to obtain Xwdb (assumed zero phase)
% N        = NFFT, the number of points used in the FFT creating Xwdb

%-- Find all peaks (magnitudes and indices) by comparing each point of ---%
%-- magnitude spectrum with its two neighbors ---%
allPeaks = [];
for i=2:length(Xwdb)-1
    ...
    ...
end

%-- Order from largest to smallest magnitude, keep only maxPeaks of them --%
peaks = ...

%-- Do parabolic interpolation in dB magnitude to find more accurate peak --%
%-- and frequency estimates --%
for i=1:maxPeaks
    idx=find(Xwdb==peaks(i));
    %parabolic interpolation
    a=Xwdb(idx-1);
    b=Xwdb(idx);
    c=Xwdb(idx+1);
    ...
    ...
end
%-- Return linear amplitude and frequency in Hz --%
% NOTE that we must use knowledge of the window to normalize amplitude here
% if we have a TD cosine of amplitude 0.6, this output should be 0.6
peaks = ...
freqs = ...
```

(b) (4 pts) Test your `findpeaks.m` function on a length-255, 400-Hz cosine having amplitude 1 and no phase offset, i.e.,

$$x(n) = \cos(2\pi \cdot 400nT), \quad n = -127, \dots, 127.$$

(Implicitly, a rectangular window of the same length is used here). Use your zero-phase zero-pad window function to extend the signal to length 2048 before peak detection. Let the sampling rate be $f_s = 8000$ Hz. Show your plot of the magnitude spectrum with the found peak clearly marked. Also plot the phase spectrum. The spectrum should be real because we zero-phase windowed a cosine, which is even (*i.e.*, the FFT of a real, even sequence is real and even). Thus, the phase should be $\pm\pi$. (Matlab may produce some "imaginary dust" in the spectrum due to round off error.) Here is some code to get you started:

```
subplot(211);
plot(abs(fft(zpzpwin(cos(2*pi*400/8000*(-127:127)'),boxcar(255),2048))));
subplot(212);
plot(angle(fft(zpzpwin(cos(2*pi*400/8000*(-127:127)'),boxcar(255),2048))));
```

(c) (3 pts) Download `s1.wav`[1] and find the amplitudes and frequencies of each sinusoidal component in the signal. You need not zero-phase window the signal, nor should you zero-pad it. Just use a rectangular window whose length is the same of the signal. To avoid getting NaN in your spectrum due to the log of zero, use something like `Xwdb = 20*log10(abs(Xlinear)+eps);` .

(d) (3 pts) Again, with `s1.wav` but now using the Hamming window of length equal to that of $x(n)$ (255 samples) and with the same amount of zero-padding (to length 2048), find the frequency and amplitude estimates. Do you expect the estimates to be better in the case of Hamming window here? Why or why not?

Solution:

(a) (10 pts) One possible version of *'findpeaks'* is shown below. Variations are acceptable if they achieve the noted lines' effect. Additionally, some flexibility should be allowed on the handling of negative frequencies. If students write 2 functions, one of which assumes that the first length(Xwdb)/f frequencies are positive, and another which assumes all input frequencies are positive, that is fine. (This is because some students designed the function first to only work on problem 1.) Also, some students changed out the find() function in the code skeleton to use a more robust version, and that is fine. Finally, there are a few ways to normalize the output peak magnitudes. In all cases, successful and "reasonably efficient" code should be accepted.

```
function [peaks,freqs]=findpeaks(Xwdb,m,fs,win,N)
% Find the peaks(up to m peaks) of the spectrum Xwdb
% peaks = a vector containing the peak magnitude estimates (dB) using
% parabolic interpolation in order from largest to smallest peak.
% freqs = a vector containing the frequency estimates (Hz) corresponding
% to the peaks defined above
% Xwdb = DFT vector of a windowed signal in dB
% m = the number of peaks we are looking for
% binsize = the size of each DFT bin = f_s/N_fft
% win = the window used to obtain Xwdb (for unbiased amplitude estimate)

Xwdb=Xwdb(:)';

%Find peaks and their locations by comparing with adjacent neighbours
allPeaks=[];
for i=2:length(Xwdb)-1
    % MUST HAVE NEXT TWO LINES OR EQUIVALENT
    if Xwdb(i)>Xwdb(i-1) & Xwdb(i)>Xwdb(i+1)
        allPeaks(length(allPeaks)+1)= Xwdb(i);
    end
end

%Then choose 'm' of them
allPeaks = flipud(sort(allPeaks'));
```

[1] http://ccrma.stanford.edu/~jos/sasp/hw/s1.wav

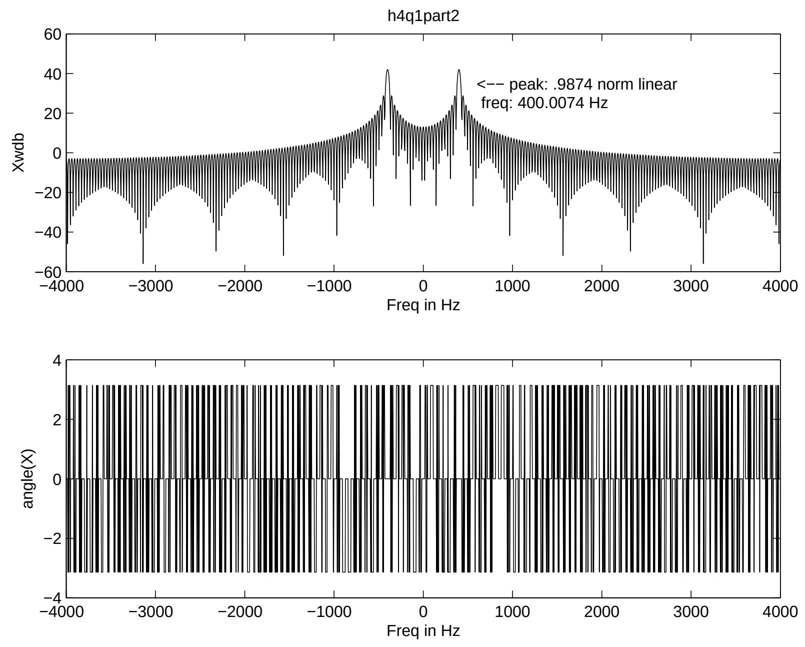
```
peaks = allPeaks(1:m);

%find associated frequencies(Hz) of those peaks
for i=1:m
    idx=find(Xwdb==peaks(i)); % Quick and Dirty way to get freqs;
    idx=idx(1);
    %parabolic interpolation
    a=Xwdb(idx-1);
    b=Xwdb(idx);
    c=Xwdb(idx+1);
    % MUST HAVE NEXT 4 LINES OR EQUIVALENT
    df=0.5*(a-c)/(a-2*b+c);
    newidx = idx+df; % gives exact freq in terms of freq indices
    freqs(i) = idx2freq(newidx,N,fs);
    peaks(i) = b-0.25*(a-c)*df;
    %if don't want parabolic interpolation, comment in below instead
    %freqs(i)=idx2freq(idx,N,fs);
end
% MUST HAVE NEXT 4 LINES (2 of content) OR EQUIVALENT
% Get rid of a biasing window in the amplitude
% Note that the window is divided by 2. Since we are trying
% to estimate real sinusoids (not complex) the amplitude in the spectrum has
% to be multiplied by 2 to get the amplitude in the time domain.
% For example, if we have:
%
%    8*cos(250*2*pi*t) (sinusoid with amplitude 8)
%
% Its spectrum is:
%
%       4*exp(-j*pi/2)       4*exp(j*pi/2)
%            ^                     ^
%            |                     |
%            |                     |
%            |                     |
%   ---------------------------------------------> f(in Hz)
%          -250         0         250
%

peaks=peaks-20*log10(sum(win)/2);
% Linear amplitude (if required)
peaks=(10.^(peaks/20));
```

(b) (4 pts) Only the plot is required:

```
fs = 8000;
N = 2048;
M = 255;
win = boxcar(M);
x = cos(2*pi*400/fs*(-(M-1)/2:(M-1)/2)');
Xlinear = fft(zpzpwin(x,win,N));
Xwdb = 20*log10(abs(Xlinear)+eps);
[peak,freq] = findpeaks(Xwdb,1,fs,win,N)
```



```
% now do plots:
subplot(211);
plot(-fs/2:fs/N:fs/2-fs/N,fftshift(Xwdb));
ylabel('Xwdb'); xlabel('Freq in Hz')
text(600,30,...
  sprintf('<-- peak: .9874 norm linear\n freq: 400.0074 Hz'))
title('h4q1part2')

subplot(212);
plot(-fs/2:fs/N:fs/2-fs/N,fftshift(angle(Xlinear)));
xlabel('Freq in Hz')
ylabel('angle(X)');
```

Using *findpeaks* from above with a 400 Hz cosine of unity amplitude, the *peak* estimate in dB is linearly .9874 and the frequency estimate is 400.0074 Hz. The plot of the spectrum and the peak is shown below.

(c) (3 pts) *s1.wav* consists of three sinusoid components at 600, 1200 and 1800 Hz with amplitudes of 0.3, 0.3 and 0.3 respectively. The fact that three components are present can be seen by a quick spectral plot. We may obtain estimates from findpeaks() via:

```
M = 8000; N = 8000; fs = 8000
Xwdb = 20*log10(abs(fft(wavread('s1.wav')))+eps);
[peaks,freqs] = findpeaks(Xwdb(1:N/2),3,fs,boxcar(M),N)
```

to get:

```
peaks (linear output) =
    0.3000
    0.3000
    0.3000
freqs =
    1200         600        1800
```

No plot is required for this problem. No code is required either; answers giving the three peaks are acceptable, even if the answer includes other peaks that are much lower. (However, a comment should be made that peaks of magnitude less than -120 dB below maximum are in general inaudible).

(d) (3 pts) Generally, using a Hamming window instead, the estimates would should be better since the level of sidelobe interference from one peak on the other is smaller. However, in this case, we are using a much shorter window length (255 as opposed to 8000) in the time domain, and the estimates are thus less accurate, namely, using the code:

```
M = 255; N = 2048;
[s1,fs] = wavread('s1.wav');
Xwdb = 20*log10(abs(fft(hamming(M).*s1(1:M),N))+eps);
[peaks,freqs] = findpeaks(Xwdb(1:N/2),3,fs,hamming(M),N)
```

we get

```
peaks =
    0.3011
    0.3009
    0.3012
freqs =
   1.0e+03 *
    1.2000    1.8002    0.5999
```

No code or plots is required in this part, just the result and comment.

2. (14 pts) **Simple Pitch Estimation:** In this problem, you will be estimating the pitch of an oboe sound, using the `findpeaks` function written previously. This problem is also concerned with the *resolvability* of the spectral peaks under different windows.

(a) Download the sound file

(b) Download `oboe.ff.C4B4.wav`[2] and similarly the Matlab files `oboeanal.m`[3] and `dbn.m`[4] into your directory. The posted version of `oboeanal.m` should be compatible with your your `findpeaks()` function prototype.

(c) (4 pts) Run the program `oboeanal.m`. Make sure you have your `findpeaks` function working already. Verify that the pitch estimate agrees with the plot. Explain what peak is used to estimate the pitch.

(d) (4 pts) Comment on the resolvability of peaks for the three windows.

(e) (6 pts) Modify the code on line 54, changing K to $(K-1)$ and then $(K+1)$ in order to vary the length of the signal under the windows. (NOTE: this line of code is after the window name has already been chosen, and currently contains only a comment line. Do NOT change K before this comment line. Note that some of the plots may then extend above 0 dB. Do not worry; you can zoom in on them anyway.) Comment on the resulting resolvability. Which case is more resolvable and amenable to spectral peak analysis?

Solution:

(a) (just a download... no points for that :))

[2] `http://ccrma.stanford.edu/~jos/sasp/hw/oboe.ff.C4B4.wav`
[3] `http://ccrma.stanford.edu/~jos/sasp/hw/oboeanal.m`
[4] `http://ccrma.stanford.edu/~jos/sasp/hw/dbn.m`

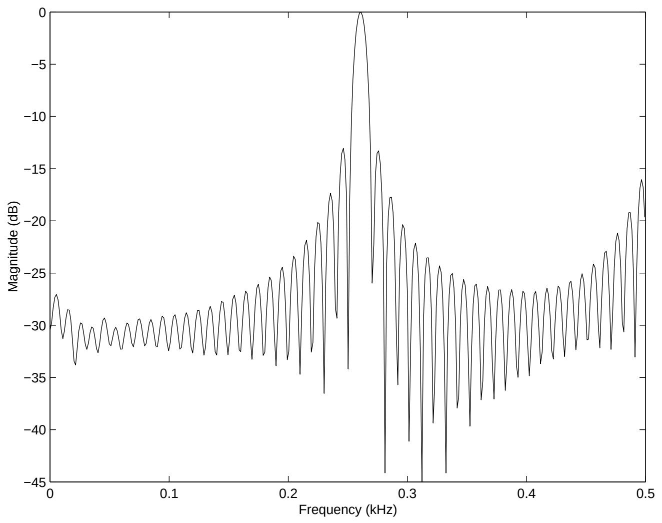


Figure 1: The spectrum interval where the peak of the fundamental frequency is searched for.

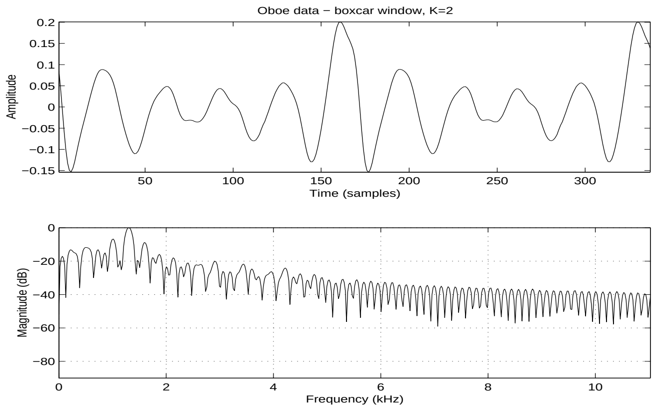


Figure 2: "Just resolvable" rectangular window

(b) (4 pts) The pitch estimate is found from near the DC region of the spectrum where the fundamental frequency resides. Note that this is ensured by oboeanal, which only passes the lower frequencies from Xwdb for consideration, specifically those frequencies at or below 500 Hz. From *oboeanal*, the pitch estimate is 260.710019 Hz, in agreement with the plot of the spectrum as shown in Figure 1. NOTE: due to normalization issues, plots showing the various windows are acceptable if the peaks are cut off above 0dB, assuming the plots are otherwise correct.

(c) (4 pts) The mainlobe width and side lobe level in general determine the resolvability of the peaks in the spectrum. The wider the mainlobe, the harder it is to resolve two nearby peaks, however, the lower the side lobe level, the easier it is to resolve peaks in general. Oboeanal.m produced plots that had various K values for various windows, so people who specified either point above about resolvability should get full credit.

(d) (6 pts) For the original K, the length of the boxcar window is just enough to resolve the harmonics of the oboe sound (Figure 2). When K is changed to $K-1$, the signal under the window is smaller than that required for resolvability. This can be seen in the spectra plotted where some peaks are merged (Figure 3). With K becoming $K+1$, a longer period of signal is taken into account and resolvability increases as can be seen by clearer and well-separated peaks in the spectra (Figure

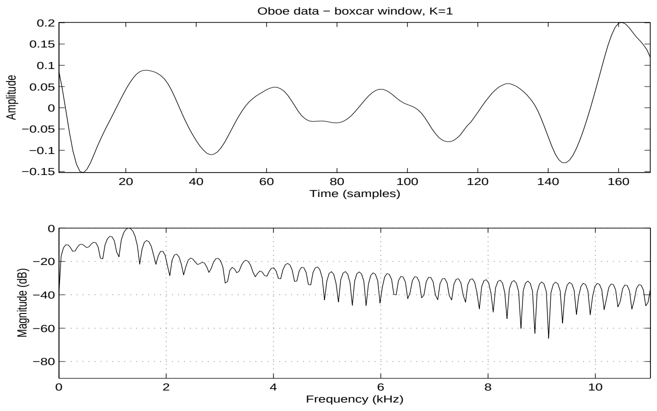


Figure 3: “Less than resolvable length” rectangular window

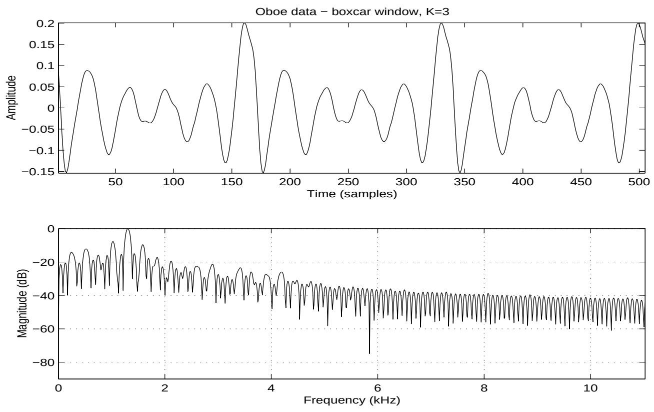


Figure 4: “Highly resolvable” rectangular window

4). For full credit a student need only comment that resolvability increases with larger K, and no plots are needed.