1. Given the even part of a real signal is

$$x_e(n) = \frac{x(n) + x(-n)}{2}$$

(a) The DFT of $x_e(n)$ is

$$\frac{1}{2}\sum_{n=0}^{N-1}(x(n) + x(-n))e^{\frac{-j2\pi kn}{N}} \tag{1}$$

Replace all $-n$'s with n's and change properties of the range of summation on second term (now reversing direction)

$$\frac{1}{2}\sum_{n=0}^{N-1} x(n)e^{\frac{-j2\pi kn}{N}} + \frac{1}{2}\sum_{n=0}^{1-N} x(n)e^{\frac{j2\pi kn}{N}} \tag{2}$$

And we know that $x(-n)$ is equivalent to $x(N-n)$ due to circular shifting, which means $n = 0 \leftrightarrow n = N$, so we can add N to all our n's to keep the second term equivalent

$$\frac{1}{2}\sum_{n=0}^{N-1} x(n)e^{\frac{-j2\pi kn}{N}} + \frac{1}{2}\sum_{n=N}^{1} x(n)e^{\frac{j2\pi kn}{N}}e^{j2\pi k} \tag{3}$$

and thus

$$\frac{1}{2}\sum_{n=0}^{N-1} x(n)e^{\frac{-j2\pi kn}{N}} + \frac{1}{2}\sum_{n=1}^{N} x(n)e^{\frac{j2\pi kn}{N}} \tag{4}$$

And again, since we know $n = 0 \leftrightarrow n = N$, the range of our second summation can match the first one

$$\frac{1}{2}\sum_{n=0}^{N-1} x(n)\left[e^{\frac{-j2\pi kn}{N}} + e^{\frac{j2\pi kn}{N}}\right] \tag{5}$$

$$\sum_{n=0}^{N-1} x(n)\cos\left(\frac{2\pi nk}{N}\right) = \mathfrak{Re}\left\{\sum_{n=0}^{N-1} x(n)e^{\frac{-j2\pi kn}{N}}\right\} \tag{6}$$

Meaning the DFT of $x_e(n)$ is $\mathfrak{Re}\{X(k)\}$

(b) First, we know that IDFT of the real component of $X(k)$ is

$$\frac{1}{N}\sum_{k=0}^{N-1}\mathfrak{Re}\{X(k)\}e^{\frac{j2\pi kn}{N}} \tag{7}$$

$$\frac{1}{N}\sum_{k=0}^{N-1}\left[\sum_{n=0}^{N-1} x(n)\cos\left(\frac{2\pi kn}{N}\right)\right]e^{\frac{j2\pi kn}{N}} \tag{8}$$

$$\frac{1}{2N}\sum_{k=0}^{N-1}\left[\sum_{n=0}^{N-1} x(n)e^{\frac{-j2\pi kn}{N}} + \sum_{n=0}^{N-1} x(n)e^{\frac{j2\pi kn}{N}}\right]e^{\frac{j2\pi kn}{N}} \tag{9}$$

Similar to in (a), we replace all the n's with $-n$'s to make the second term's exponential term fit our definition of the DFT. This, again, changes the properties of the range of summation on second term (now reversing direction)

$$\frac{1}{2N}\sum_{k=0}^{N-1}\left[\sum_{n=0}^{N-1} x(n)e^{\frac{-j2\pi kn}{N}} + \sum_{n=0}^{1-N} x(-n)e^{\frac{-j2\pi kn}{N}}\right]e^{\frac{j2\pi kn}{N}} \tag{10}$$

Again, similar to (a), we know that the range of the summation is equivalent to $n = 0 \rightarrow N-1$, so we get

$$\frac{1}{2N}\sum_{k=0}^{N-1}(X(k)+X(-k))e^{\frac{j2\pi kn}{N}} \tag{11}$$

$$\frac{1}{2N}\sum_{k=0}^{N-1}X(k)e^{\frac{j2\pi kn}{N}}+\frac{1}{2N}\sum_{k=0}^{N-1}X(-k)e^{\frac{j2\pi kn}{N}} \tag{12}$$

Showing that the IDFT of the real component of $X(k)$ can be represented in terms of $x(n)$ in the following fashion

$$\frac{1}{N}\sum_{k=0}^{N-1}\mathfrak{Re}\{X(k)\}e^{\frac{j2\pi kn}{N}}=\frac{x(n)+x(-n)}{2} \tag{13}$$

2. A length of sampled speech has 63412 samples. An FIR digital filter has 98 coefficients. We want to apply this filter to the speech using the overlap-add convolution procedure with block size of 512 samples. Assume: input $\rightarrow$ array `x`, filter coefficients $\rightarrow$ array `b`, output $\rightarrow$ array `y`

   (a) The number of blocks of data is calculated by

$$B=\left\lceil\frac{63412}{512-98+1}\right\rceil=153 \tag{14}$$

   (b) The number of zeros added to the filter must be equivalent to the block size minus the length of the FIR digital filter (amount of coefficients). In this case, that's $512-98=414$ zeros. Another way to calculate this is to find the number of speech samples in each block, and then append that minus one zeros, which still results in 414. In MATLAB, we calculate using the latter

```
%% 2
x = normrnd(0, 1, 1, 63412);     % random speech
b = ones(1, 98);                 % all coefficients 1

block_size = 512;
speech_segment_len = block_size - length(b) + 1;
num_blocks = ceil(length(x) / speech_segment_len);

b_pad = [b, zeros(1, speech_segment_len - 1)];
```

```
>> speech_segment_len - 1

  414
```

   (c) The number of zero's to be added to the first block of data is equivalent to the block size minus the length of the speech segment within each block. In this case, that's $512-415=97$. The number of data values used in a block of data has already been calculated in (b), given by `speech_segment_len`

```
% storing each block in 2d array, each row is a different block
x_blocks = zeros(num_blocks, block_size);

% populate the blocks accordingly
for i = 1:num_blocks - 1
    x_blocks(i, :) = [x(((i - 1) * speech_segment_len) + 1:(i * ...
        speech_segment_len)), zeros(1, length(b) - 1)];
end
% populate the last block differently, since the speech length is ...
    not a perfect multiple of speech_segment_len
x_blocks(num_blocks, :) = [x(((num_blocks - 1) * ...
    speech_segment_len):end), zeros(1, block_size - (length(x) - ...
    (num_blocks - 1) * speech_segment_len) - 1)];
```

As seen, the number of zeros appended to each block is

```
>> length(b) - 1

    97
```

And the number of data points within each block is given by

```
>> speech_segment_len

    415
```

(d) The first block would be generated as such. First the output `y` must be initialized with zeros, and the for each block, the selected range of `y` (size of block size) must add the convolution between the FIR filter and the block to itself. Here is the first block in MATLAB

```
% initialize output y
y = zeros(size(x));

% perform dft on padded filter and first block
ft1 = fft(b_pad);
ft2 = fft(x_blocks(1, :));

% add result to appropriate y range. in this case, the first block ...
    uses indicies 1 to 512
y(1:block_size) = y(1:block_size) + ifft(ft1 .* ft2);
```

(e) All blocks of data has already been calculated in part (c), so the second block of data will also append zeros of length 97, given by

```
>> length(b) - 1

    97
```

The DFT of this second block is given by the following MATLAB script

```
% perform dft on second block
ft3 = fft(x_blocks(2, :));
```

(f) Similar to part (d), the second block of data will be added to the output `y` in the following way

```
% add the second block, shifting over by speech_segment_len amount, ...
    to allow for 97 data points of overlap
y((speech_segment_len + 1):(speech_segment_len + block_size)) = ...
    y((speech_segment_len + 1):(speech_segment_len + block_size))) + ...
    ifft(ft1 .* ft3);
```

This homework was difficult to manage by doing blocks individually, so I will include how to calculate all of `y`

```
% dft of padded FIR filter
ft1 = fft(b_pad);
% row-wise dft of each block, N = block_size
x_ft = fft(x_blocks, block_size, 2);

% initialize output y
y = zeros(size(x));
idx = 0;
% populate y accordingly
for i = 1:num_blocks - 1
    y((idx + 1):(idx + block_size)) = y((idx + 1):(idx + ...
        block_size)) + ifft(ft1 .* x_ft(i, :));
    idx = idx + speech_segment_len;
end
% last block is calculated differently due to y being length of ...
    input x
last_part = ifft(ft1 .* x_ft(num_blocks, :));
y(idx:end) = y(idx:end) + last_part(1:length(y(idx:end)));
```

3. (a) An arbitrary rectangular window ranges from $0 \leq n \leq L-1$ with unity gain. The frequency spectra for this rectangular window can be derived by taking the discrete-time Fourier transform of the time domain window. Since the window is $0 \; \forall \; n < 0, \; n > L-1$, the limits of summation range from 0 to $L-1$

$$W_r(f) = \sum_{n=0}^{L-1} e^{-j2\pi fn} \tag{15}$$

And we know that a geometric series result in the following

$$a\sum_{k=0}^{n-1} r^k = a\frac{1-r^n}{1-r} \tag{16}$$

So if we set $r = e^{-j\omega}$, then we evaluate the sum to be

$$\frac{1-e^{-j2\pi fL}}{1-e^{-j2\pi f}} \tag{17}$$

Since this window is defined in the range $n = 0 \to L-1$, the Fourier transform is equivalent to the Fourier transform of the same window in range $n = \frac{-(L-1)}{2} \to \frac{L-1}{2}$ with a time shift of $\frac{L-1}{2}$. Knowing this, we are able to pull out these time-shifting exponential terms and construct sinusoidal functions.

$$\begin{bmatrix}\frac{1}{2j}\\ \frac{1}{2j}\end{bmatrix}\left[\frac{e^{\frac{j2\pi fL}{2}}-e^{\frac{-j2\pi fL}{2}}}{e^{\frac{j2\pi f}{2}}-e^{\frac{-j2\pi f}{2}}}\right]\frac{e^{\frac{-j2\pi fL}{2}}}{e^{\frac{-j2\pi f}{2}}} \tag{18}$$

$$\frac{\sin(\pi fL)}{\sin(\pi f)}e^{-j\pi f(L-1)} \tag{19}$$

And in terms of angular frequency $\omega = 2\pi f$

$$W_r(\omega) = \frac{\sin(\frac{\omega}{2}L)}{\sin(\frac{\omega}{2})}e^{-j\frac{\omega}{2}(L-1)} \tag{20}$$

The Hann window's continuous time spectra can be derived by taking the discrete-time Fourier transform of the time domain window. Similar to the previous window, the limits of summation range from 0 to $L-1$

$$W_h(f) = \sum_{n=0}^{L-1}\left[\frac{1}{2}-\frac{1}{2}\cos\left(\frac{2\pi n}{L-1}\right)\right]e^{-j2\pi fn} \tag{21}$$

$$\frac{1}{2}\sum_{n=0}^{L-1}e^{-j2\pi fn}-\frac{1}{4}\sum_{n=0}^{L-1}\left(e^{\frac{j2\pi n}{L-1}}+e^{\frac{-j2\pi n}{L-1}}\right)e^{-j2\pi fn} \tag{22}$$

We use the geometric series in equation (16) to evaluate all sums. And similar to the previous window, we know that the range $n = 0 \to L-1$ results in a sinusoidal wave with a time shift of $\frac{L-1}{2}$, so we can begin constructing sinusoids

$$\frac{1}{2}\left[\frac{1-e^{-j2\pi fL}}{1-e^{-j2\pi f}}\right]-\frac{1}{4}\sum_{n=0}^{L-1}e^{-j2\pi n\left(f-\frac{1}{L-1}\right)}-\frac{1}{4}\sum_{n=0}^{L-1}e^{-j2\pi n\left(f+\frac{1}{L-1}\right)} \tag{23}$$

$$\frac{1}{2}\left[\frac{\sin(\pi fL)}{\sin(\pi f)}\right]e^{-j\pi f(L-1)}-\frac{1}{4}\left[\frac{1-e^{-j2\pi L\left(f-\frac{1}{L-1}\right)}}{1-e^{-j2\pi\left(f-\frac{1}{L-1}\right)}}\right]-\frac{1}{4}\left[\frac{1-e^{-j2\pi L\left(f+\frac{1}{L-1}\right)}}{1-e^{-j2\pi\left(f+\frac{1}{L-1}\right)}}\right] \tag{24}$$

$$\begin{aligned}\frac{1}{2}\left[\frac{\sin(\pi fL)}{\sin(\pi f)}\right]e^{-j\pi f(L-1)}-\frac{1}{4}\left[\frac{\sin\left(\pi L\left(f-\frac{1}{L-1}\right)\right)}{\sin\left(\pi\left(f-\frac{1}{L-1}\right)\right)}\right]e^{-j\pi(f(L-1)-1)}+\\ -\frac{1}{4}\left[\frac{\sin\left(\pi L\left(f+\frac{1}{L-1}\right)\right)}{\sin\left(\pi\left(f+\frac{1}{L-1}\right)\right)}\right]e^{-j\pi(f(L-1)+1)}\end{aligned} \tag{25}$$

The ±1 in the last two exponential terms will result in $e^{\pm j\pi} = -1$, so we can safely make some sign changes

$$\left[\frac{2\sin(\pi fL)}{\sin(\pi f)}+\frac{\sin\left(\pi L\left(f-\frac{1}{L-1}\right)\right)}{\sin\left(\pi\left(f-\frac{1}{L-1}\right)\right)}+\frac{\sin\left(\pi L\left(f+\frac{1}{L-1}\right)\right)}{\sin\left(\pi\left(f+\frac{1}{L-1}\right)\right)}\right]\frac{e^{-j\pi f(L-1)}}{4} \tag{26}$$

And in terms of angular frequency $\omega = 2\pi f$

$$W_h(\omega) = \left[\frac{2\sin(\frac{\omega}{2}L)}{\sin(\frac{\omega}{2})} + \frac{\sin\left(L\left(\frac{\omega}{2} - \frac{\pi}{L-1}\right)\right)}{\sin\left(\frac{\omega}{2} - \frac{\pi}{L-1}\right)} + \frac{\sin\left(L\left(\frac{\omega}{2} + \frac{\pi}{L-1}\right)\right)}{\sin\left(\frac{\omega}{2} + \frac{\pi}{L-1}\right)} \right] \frac{e^{-j\frac{\omega}{2}(L-1)}}{4} \tag{27}$$

(b) Plotting the results from (a) in MATLAB. Here is the script

```matlab
%% 3 part b
L = 30;
N = 400;
k = -100:100;
omegak = 2 * pi * k / N;

% rectangular window DTFT
Wr_omegak = sin((omegak * L) / 2) .* exp(-1i * (L - 1) * omegak / 2) ...
    ./ sin(omegak / 2);

% Hann window DTFT
Wh_omegak = (2*(sin((omegak * L) / 2) ./ sin(omegak / 2)) + (sin(L * ...
    (omegak / 2 - pi / (L - 1))) ./ sin(omegak / 2 - pi / (L - 1))) ...
    + (sin(L * (omegak / 2 + pi / (L - 1))) ./ sin(omegak / 2 + pi / ...
    (L - 1)))) .* exp(-1i * (L - 1) * omegak / 2) / 4;
```

The results are plotted with respect to `k`, which ranges from -100 to 100.

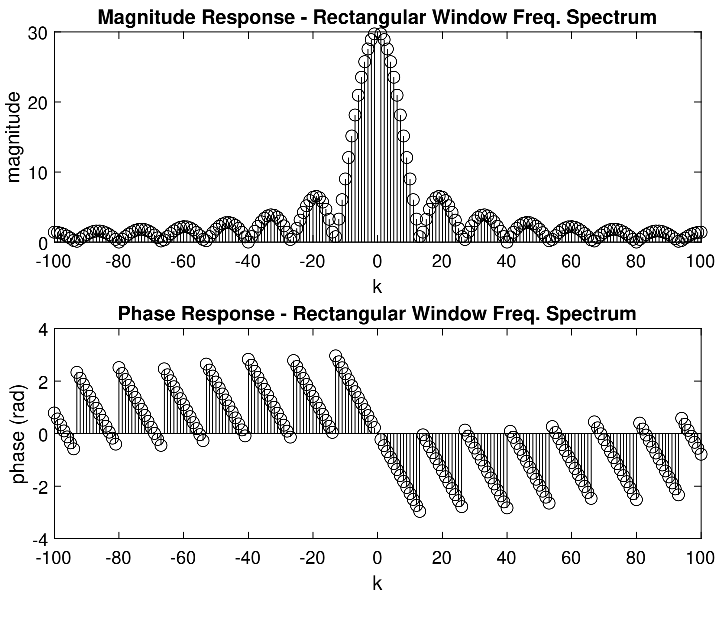


Figure 1: Magnitude and Phase Response of Rectangle Window

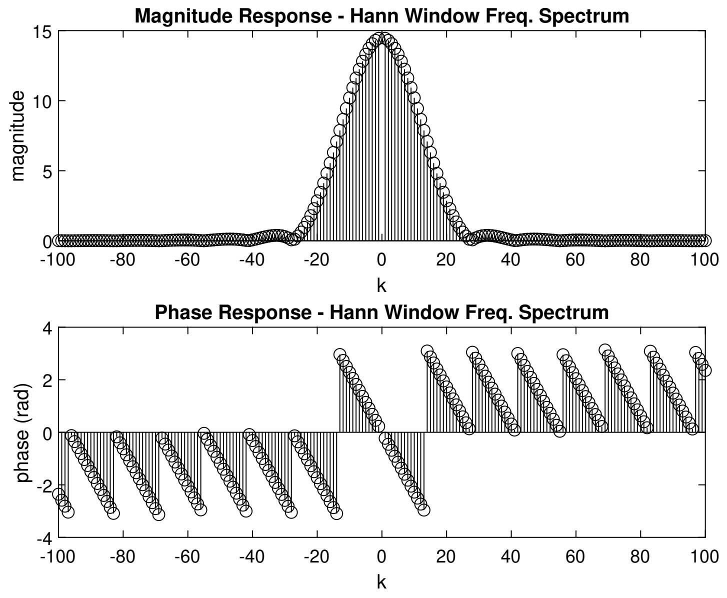


Figure 2: Magnitude and Phase Response of Hann Window

(c) Assume we have an incoming signal $x(t)$ consisting of 3 summated cosinusoidal waveforms with frequencies of 200 Hz, 220 Hz, and 250 Hz. Calculate the N-point DFT of $x_{r_i}(n) = x(n)w_{r_i}(n)$ as well as $x_{h_i}(n) = x(n)w_{h_i}(n)$, where $N = 2^{14}$, and the rectangular window (w_{r_i}) and Hann window (w_{h_i}) have window length $L = 50, 100, 150$ respective to $i = 1, 2, 3$.

Before I show my results, here is the heavily commented MATLAB script. I did not go off the script on the slides, so it might be longer than normal. However, the results are one in the same.

```matlab
% getting time vector or arbitrary (but long) length
% since we are preparing to sample at Fs = 1500 Hz
ΔT = 1e−6;
tfinal = 1;
t = 0:ΔT:tfinal;
t = round(t, 6);

% sampling frequency, calculate nT
Fs = 1500;
Ts = 1 / Fs;
nfinal = tfinal / Ts;
n = 0:nfinal;

% create our signal x(t)
F1 = 200;
F2 = 220;
F3 = 250;
xat = cos(2*pi*F1*t) + cos(2*pi*F2*t) + cos(2*pi*F3*t);

% sample at nT, this algorithm was showcased in my HW3
nT_i = zeros(size(n));
for i = n
```

```
    nT_i(i + 1) = find(round(i * Ts, 6) == t);
end
xanT = xat(nT_i);
nT = t(nT_i);

% get N value for resolution of dft
N = 2^14;
% get L values. I included a 400 just to see what a large window ...
    looked like
L = [50; 100; 150; 400];
% k ranges from 1 -> N (frequency domain). NOT 0 or else we have a ...
    divide by 0
k = 1:N;

% preallocate space for rectangular and Hann windows results
xr = zeros(3, length(xanT));
xh = zeros(3, length(xanT));
% loop through all window lengths
for i = 1:length(L)
    en = 0:L(i) - 1;
    % preallocate our time domain windows with zeros, ...
        computationally efficient
    wr_time = zeros(1, length(xanT));
    wh_time = zeros(1, length(xanT));

    % create our windows in time domain
    wr_time(1:L(i)) = 1;
    wh_time(1:L(i)) = 0.5 - 0.5*cos(2*pi*en./(L(i) - 1));

    % element-wise multiply our windows with our sampled signal, x(nT)
    xr(i, :) = wr_time .* xanT;
    xh(i, :) = wh_time .* xanT;
end

% N-point DFT of our windows
xr_fft = fft(xr, N, 2);
xh_fft = fft(xh, N, 2);

% plot with respect to frequency
F = (omegak/(2*pi))*Fs;

% plot and save all of our dft's of xr_i and xh_i
for i = 0:length(L) - 1
    figure(2*i + 1);
    plot(F(1:N/2), abs(xr_fft(i + 1, 1:N/2)));
    title(sprintf("Spectrum of Rect. Window on x(t), (X_h(k)), L = ...
        %d, i = %d", L(i + 1), i + 1));
    xlabel("f (Hz)");
    ylabel("magnitude");
    print(sprintf("3c_rect_%d", i + 1), "-deps");

    figure(2*i + 2);
    plot(F(1:N/2), abs(xh_fft(i + 1, 1:N/2)));
    title(sprintf("Spectrum of Hann Window on x(t), (X_h(k)), L = ...
        %d, i = %d", L(i + 1), i + 1));
    xlabel("f (Hz)");
    ylabel("magnitude");
    print(sprintf("3c_hann_%d", i + 1), "-deps");
end
```

i. $L = 50$ for both rectangular and Hann window

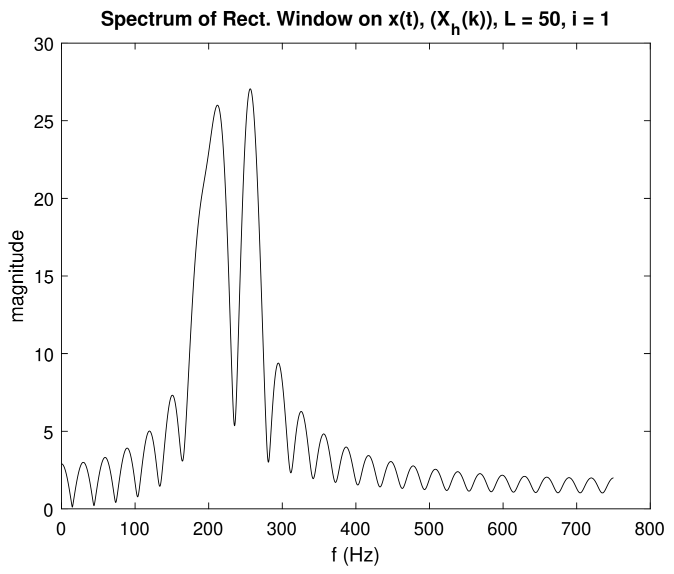


Figure 3: X_{r_1} where $L_1 = 50$

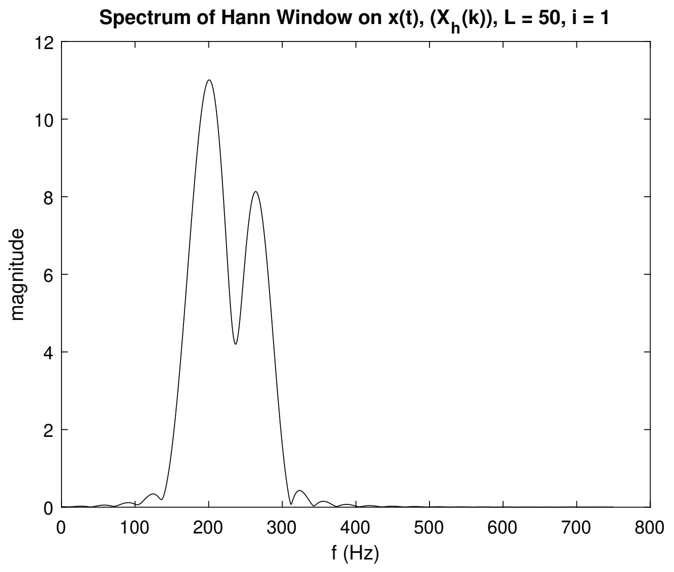


Figure 4: X_{h_1} where $L_1 = 50$

This is a small window length, so the Hann window will suffer greater than the rectangular. As seen, the Hann window has far less magnitude than the rectangular window and less distinct peaks.

ii. $L = 100$ for both rectangular and Hann window

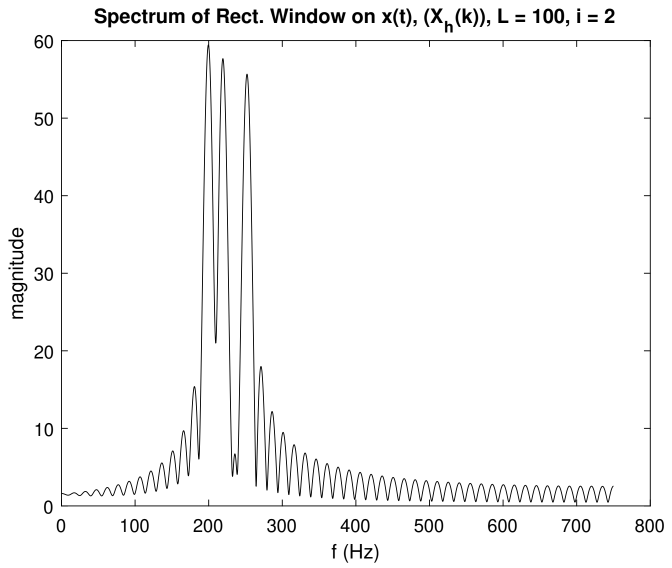


Figure 5: X_{r_2} where $L_2 = 100$

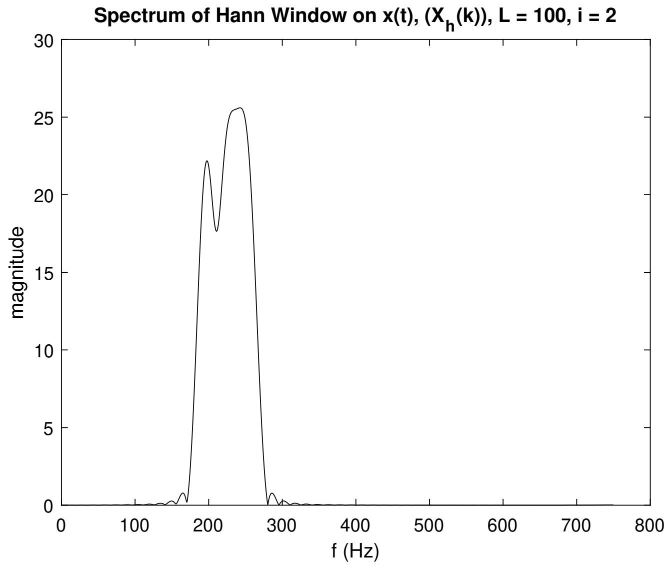


Figure 6: X_{h_2} where $L_2 = 100$

Similar to $L = 50$, the rectangular window has more distinct peaks than the Hann window. In fact, it is here at $L = 100$ where the rectangular window has all three distinct peaks. The Hann window is still struggling, but one upside is the vastly reduced noise on all non-input frequencies.

iii. $L = 150$ for both rectangular and Hann window

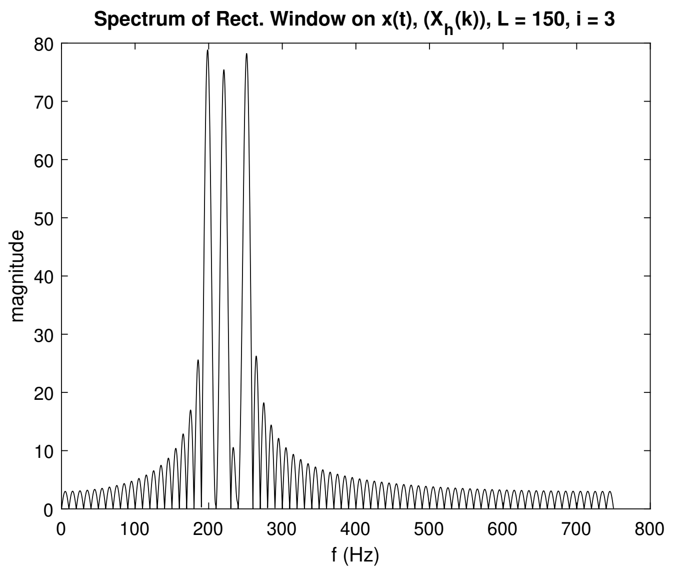


Figure 7: X_{r_3} where $L_3 = 150$

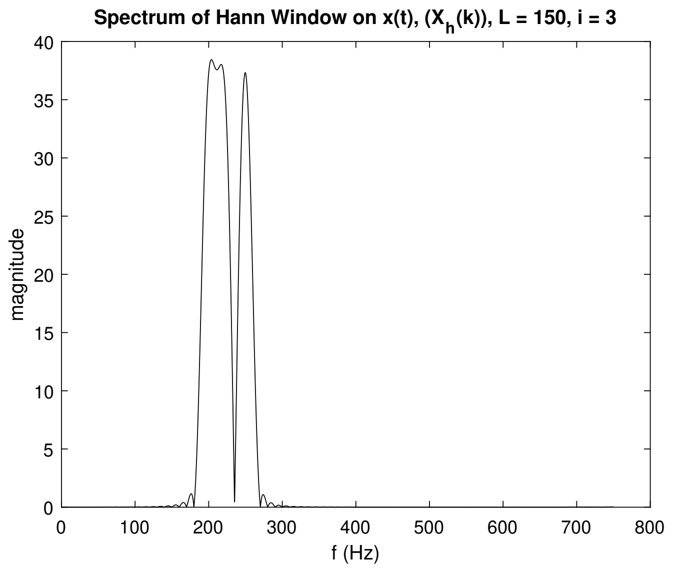


Figure 8: X_{h_3} where $L_3 = 150$

Here, the the rectangular window still has its three peaks, but at the cost of a lot of noise. (roughly 15 in magnitude). The Hann window is just starting to forming peaks, and I hypothesize that a greater window length will enable the Hann window to form its distinct peaks. The Hann window does an incredible job with reducing noise when compared to the rectangular window.

iv. Not necessary for homework, but reasoning from my answer in previous sections. $L = 400$ for both rectangular and Hann window

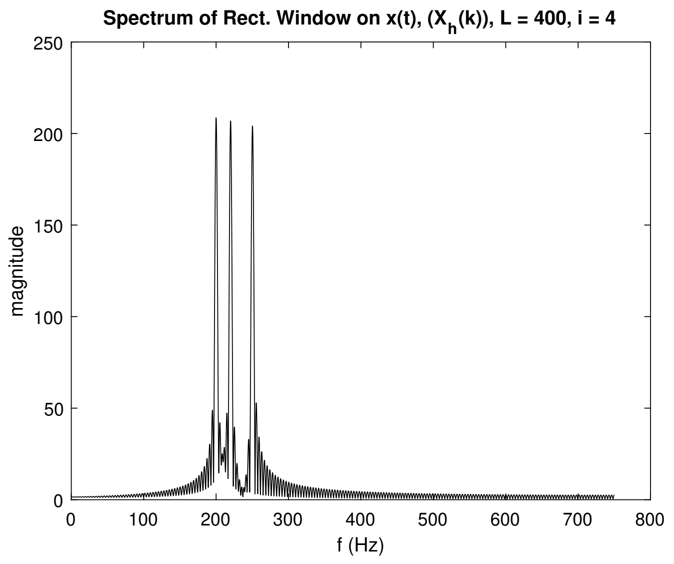


Figure 9: X_{r_4} where $L_4 = 400$

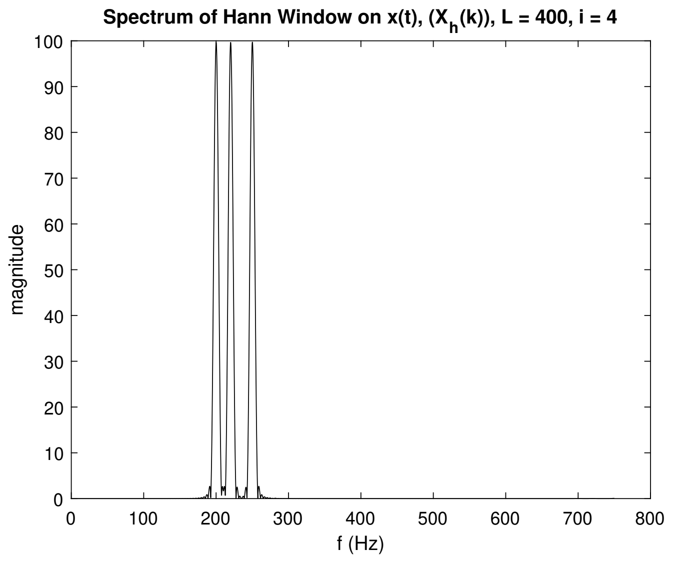


Figure 10: X_{h_4} where $L_4 = 400$

I went ahead and tested a larger window length, where $L = 400$. It is evident that the Hann window has clear, distinct peaks with little to no noise. The absolute difference between the magnitude of the peaks versus the magnitude of the noise is incredible. On the other hand, the rectangular window's three peaks double in magnitude of the Hann windows, but is extremely noisy.

4. Use the `fft` algorithm in MATLAB to compute the DCT of a sequence

   (a) Plot of `sampdata`

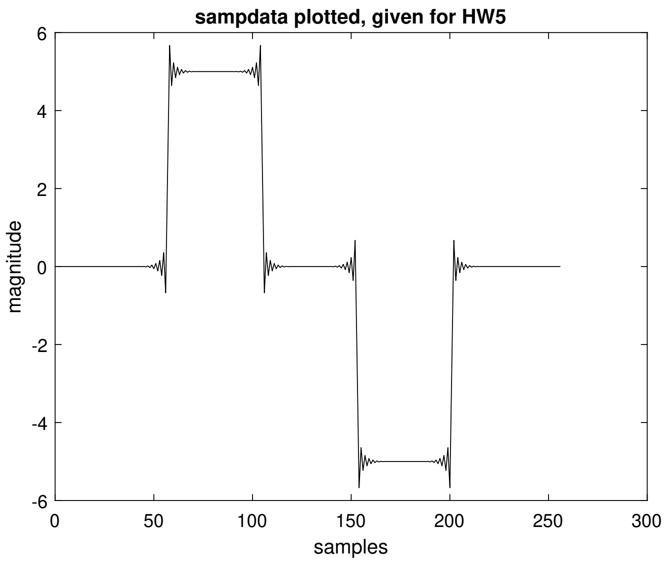


Figure 11: `sampdata`

   (b) Plotting the DCT of `sampdata` using the `dct` function in MATLAB

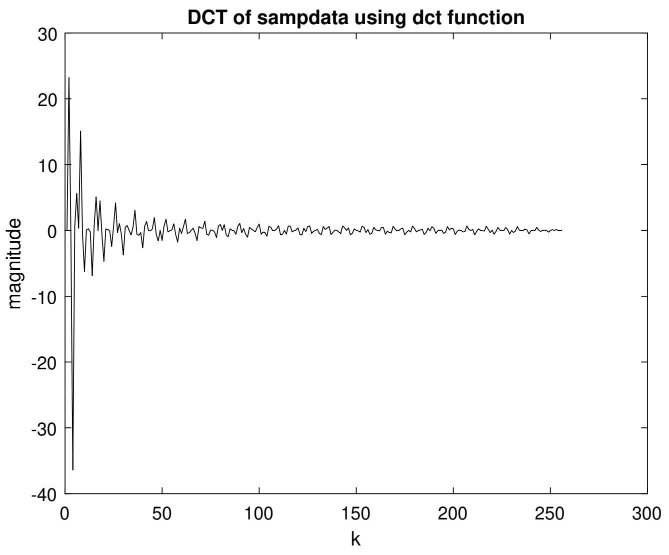


Figure 12: DCT of `sampdata`

As we can see, the signal is completely real.

(c) Plotting the DCT of `sampdata` by creating my own MATLAB routine, `arpad_dct`

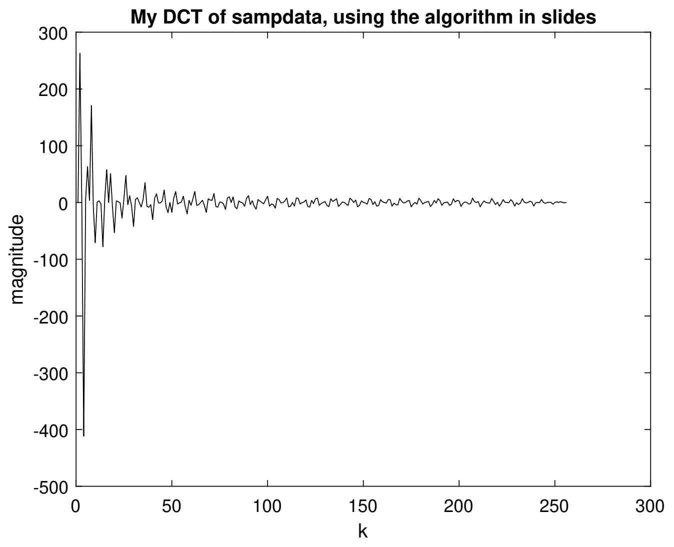


Figure 13: DCT of `sampdata` using `arpad_dct`

As we can see, the plot looks identical to the DCT using MATLAB's built in `dct` function. However, the magnitude is completely off. Therefore, the absolute error will be massive. More in this in part (d). Here is the script:

```
function Sk = arpad_dct(x, varargin)
% arpad_dct() is simply a function which uses the fft()
%   routine to calculate the DCT of a sequence x. limited
%   in terms of N-point DCT; can only compute for 1 value
%   of N
%   INPUTS:     x       sequence
%   OUTPUTS:    result: resulting n-point DCT of sequence x

N = length(x);
% s(n) =       x(n)      ->  0 ≤ n ≤ N - 1
% s(n) = x(2N - 1 - n)   ->  N ≤ n ≤ 2N - 1
sn = [x, flip(x)];
Sk = fft(sn, 2*N);

k = 0:2*N - 1;
Sk = Sk .* exp(-1i * pi * (k / 2) / N);
Sk = Sk(1:N) / 2;

end
```

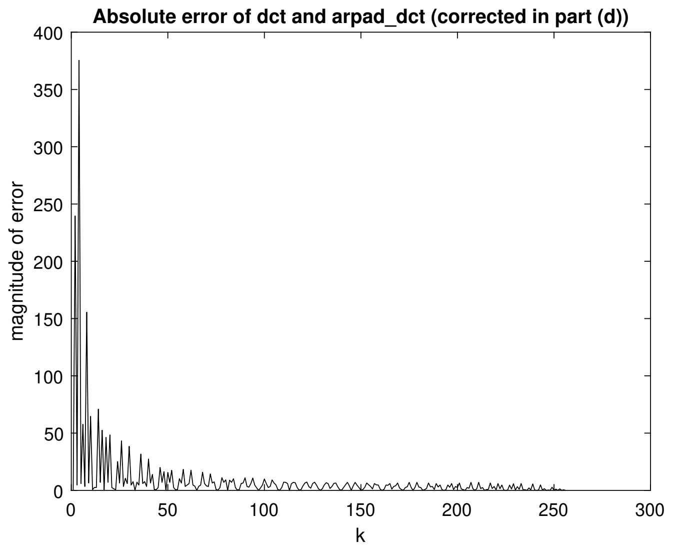


Figure 14: Absolute error of `dct` vs `arpad_dct`, incorrect

Since the magnitudes are so far off, our error is very wrong. In part (d), we correct this to get a negligible absolute error.

(d) By going through the MATLAB documentation on their `dct` function, the following shown in Figure 17 on page 17.

So I have incorporated that into the last line of my `arpad_dct` function

```
% scaled by an extra sqrt(2 / N), as seen in MATLAB documentation
Sk = Sk(1:N) * sqrt(2 / N) / 2;
```

Now the absolute error and the final result should be very similar to what MATLAB has.

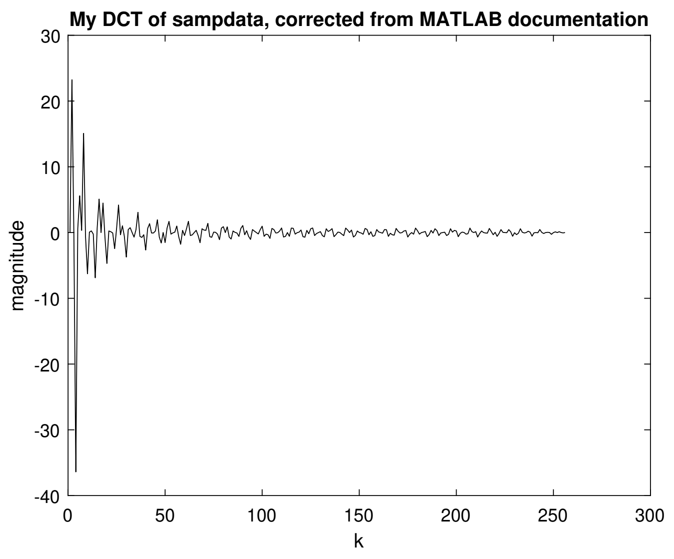


Figure 15: DCT of `sampdata` using `arpad_dct`

As we see, the magnitude is very similar to ideal DCT in Figure 12.

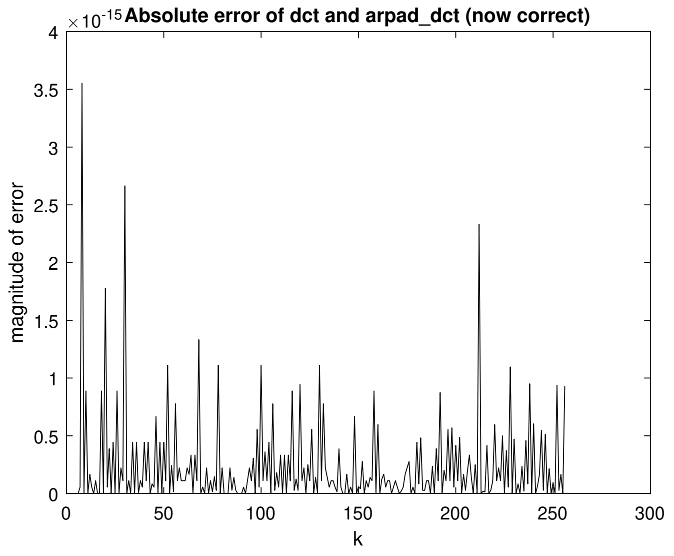
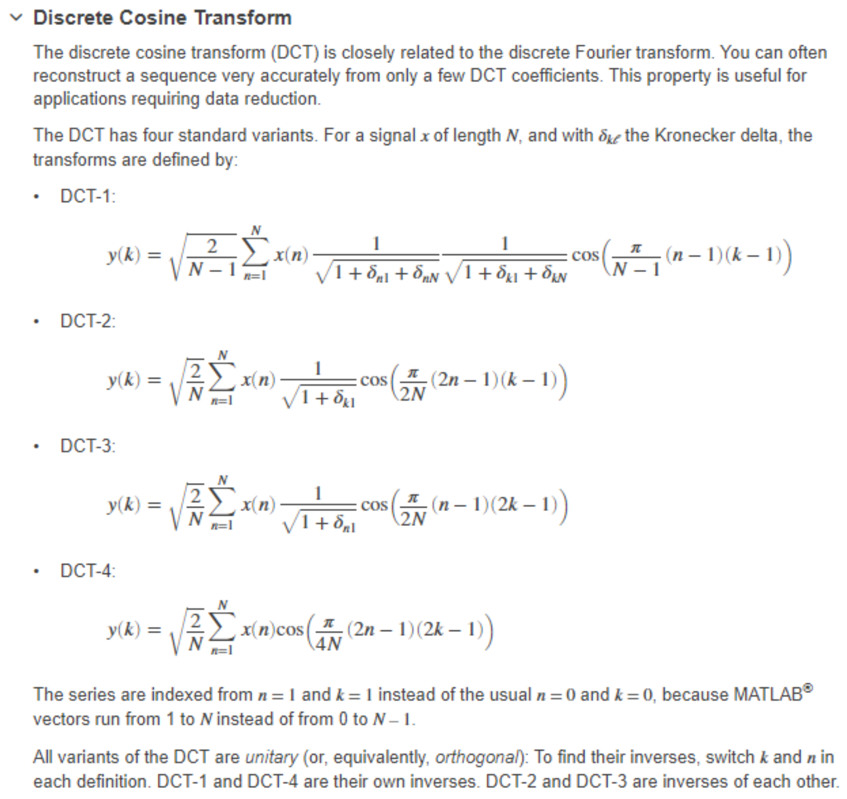


Figure 16: Absolute error of `dct` vs `arpad_dct`, correct

And the absolute error is incredibly small, which is extremely good! We have successfully implemented our own DCT routine using MATLAB's `fft` function.



Discrete Cosine Transform

The discrete cosine transform (DCT) is closely related to the discrete Fourier transform. You can often reconstruct a sequence very accurately from only a few DCT coefficients. This property is useful for applications requiring data reduction.

The DCT has four standard variants. For a signal x of length N, and with $\delta_{k\ell}$ the Kronecker delta, the transforms are defined by:

- DCT-1:

$$y(k) = \sqrt{\frac{2}{N-1}} \sum_{n=1}^{N} x(n) \frac{1}{\sqrt{1+\delta_{n1}+\delta_{nN}}} \frac{1}{\sqrt{1+\delta_{k1}+\delta_{kN}}} \cos\left(\frac{\pi}{N-1}(n-1)(k-1)\right)$$

- DCT-2:

$$y(k) = \sqrt{\frac{2}{N}} \sum_{n=1}^{N} x(n) \frac{1}{\sqrt{1+\delta_{k1}}} \cos\left(\frac{\pi}{2N}(2n-1)(k-1)\right)$$

- DCT-3:

$$y(k) = \sqrt{\frac{2}{N}} \sum_{n=1}^{N} x(n) \frac{1}{\sqrt{1+\delta_{n1}}} \cos\left(\frac{\pi}{2N}(n-1)(2k-1)\right)$$

- DCT-4:

$$y(k) = \sqrt{\frac{2}{N}} \sum_{n=1}^{N} x(n) \cos\left(\frac{\pi}{4N}(2n-1)(2k-1)\right)$$

The series are indexed from $n = 1$ and $k = 1$ instead of the usual $n = 0$ and $k = 0$, because MATLAB® vectors run from 1 to N instead of from 0 to $N-1$.

All variants of the DCT are *unitary* (or, equivalently, *orthogonal*): To find their inverses, switch k and n in each definition. DCT-1 and DCT-4 are their own inverses. DCT-2 and DCT-3 are inverses of each other.

Figure 17: MATLAB scales output by $\sqrt{\frac{2}{N}}$