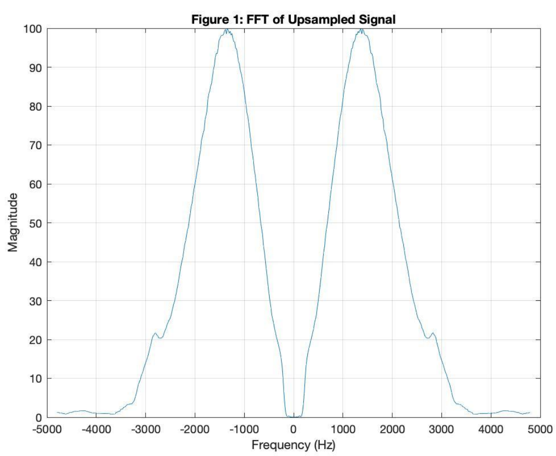
ECE 113 Channel Project Report, Fall 2021

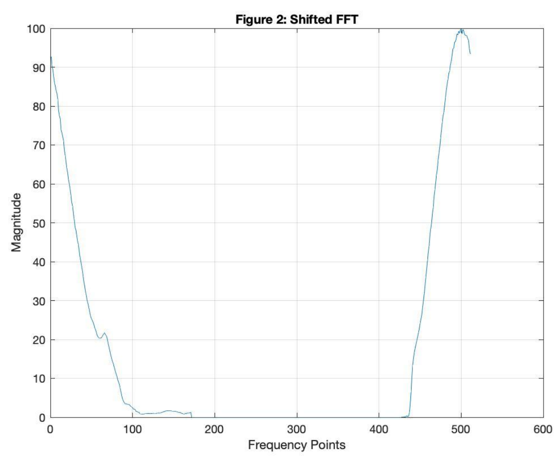
Yu-Wei Vincent Yeh (UID: 005123289)
(Partner: Kellen Cheng (UID: 905155544))

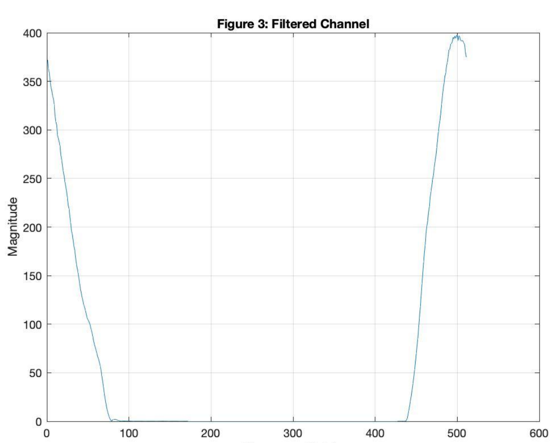
12/03/2021

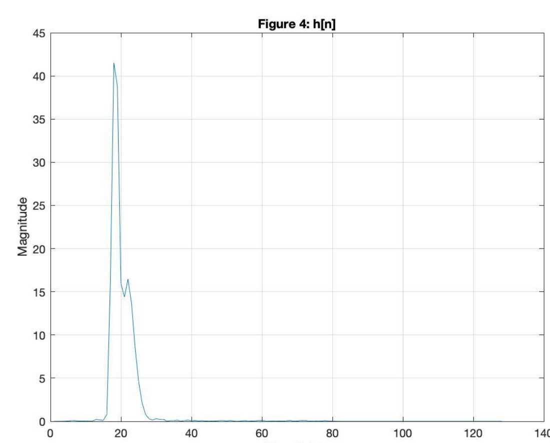
Part 1:

a. For the spline interpolation, please refer to the code attached at the end of this report.
b. Please refer to Figure 1, the upsampled fourier domain representation, notice that there are peaks that occur at approximately 1500Hz, which is in close proximity to 1600Hz.
c. Please refer to the code for zeroing out the negative frequencies.
d. Please refer to Figure 2, the cyclically shifted spectrum, with the # of steps to shift determined by the method taught in the lecture. Notice that the “peak” isn’t exactly at 1, which is consistent with our observation in b.
e. Please refer to the code attached at the end of this report for the SRRC implementation, with truncations.
f. Please refer to Figure 3, the multiplication of srrc and upsampled array in the frequency domain.
g. Please refer to Figure 4, the 4-times downsampled signal. The sum of squares of the absolute values (hpower) is 4.51e+3.





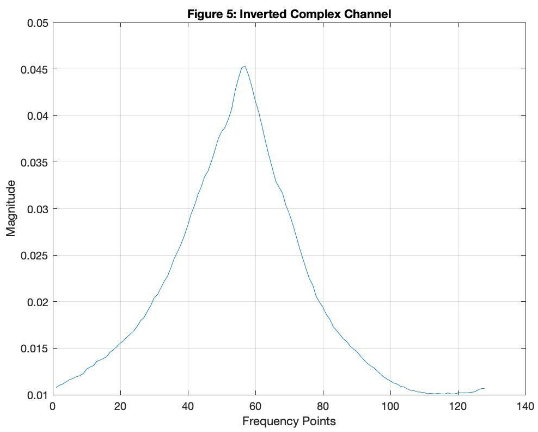


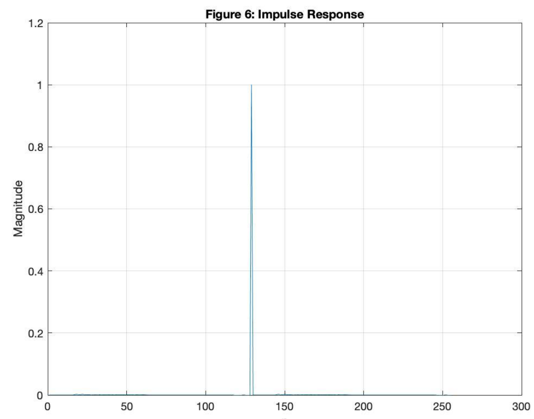


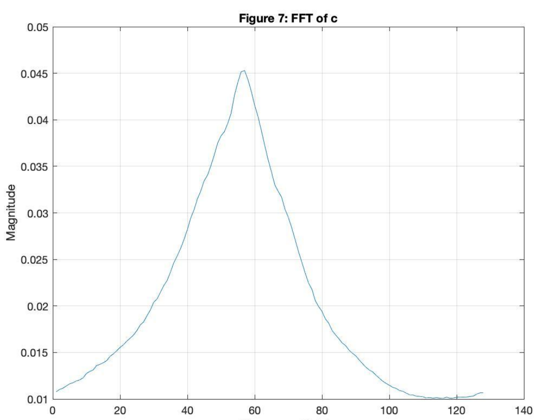
Part 2:

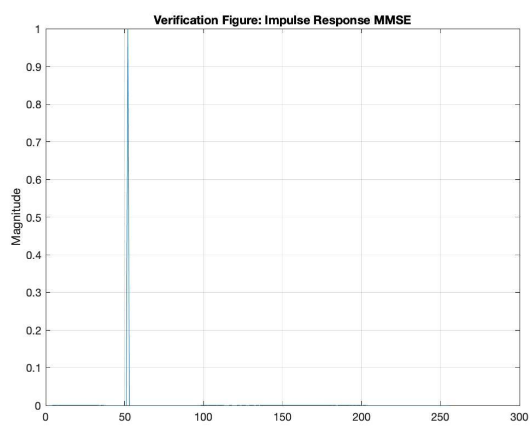
a. Please refer to the code attached for the fft of the complex channel, inversion in the frequency, and convolution. Please refer to Figure 5 and 6 for the respective plots. The maximum squared value is 1.0000, and the SIR is 1.03e+4. From the graph, we see a delay of 129, and compared to part 1, where Figure 4 shows a “wide impulse,” figure 6 showed a clean and narrow impulse with a magnitude of one. Thus, the degree of interference suppression is much more significant in this part.

b. Please refer to the code for the implementation of the “R matrix,” equalizer, and delay calculation. Please refer to Figure 7 for the fft of c with the best delay, which according to our calculation, turned out to be 52. With this delay, we get an SIR of 1.54e+9. Compared to the ZF equalizer, we see MMSE has an even better improvement in interference suppression, and has a delay that is much smaller. ***A plot of impulse for verification purposes is also provided below.







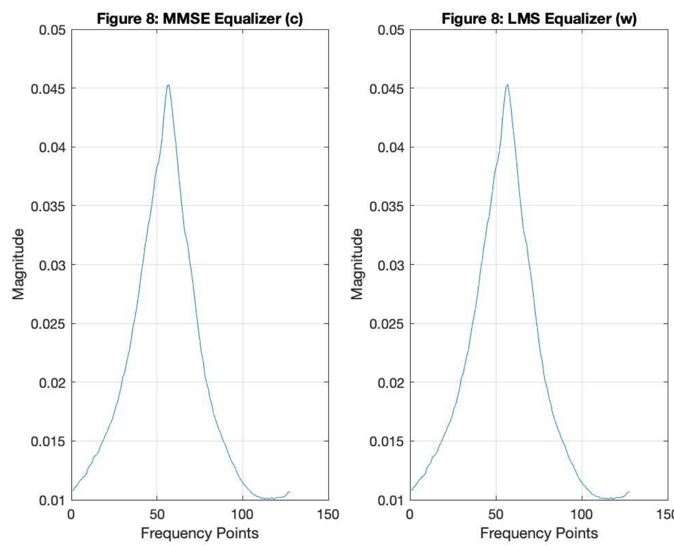


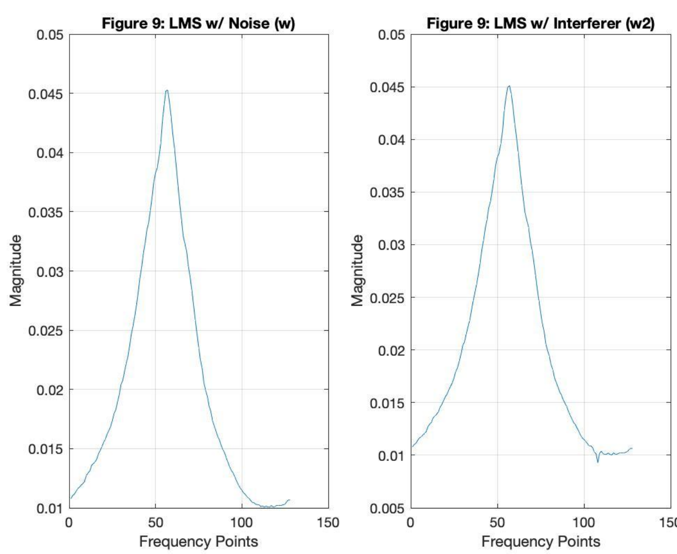
Part 3:

*****For adaptive equalizer implementation steps a-e, please refer to the attached code**.

a. See Figure 8 for the side-by-side comparison of c and w. For SINR, we got 1.41e+9, which is in the same ballpark as 2b, as they have a similar order of magnitude.

b. See the attached code for the interferer implementation, and refer to Figure 9 for a side-by-side comparison. We see that the SINR is 1.07e+3 with 5 million iterations, which is comparably smaller. However, even with this SINR, a good amount of suppression still occurred, since the two plots are quite similar to one another. The advantage of this approach over an equalizer or cascade notch filter is that it does not produce amplification at interference, like those that occur when we try to invert the notch filter, which is non-ideal.

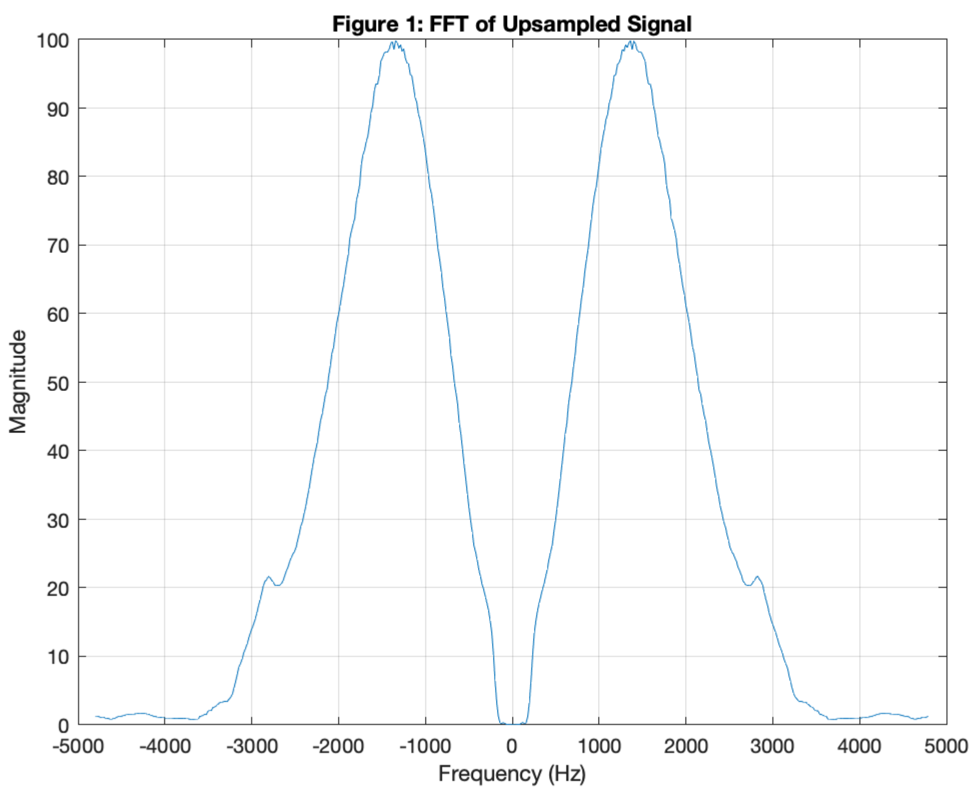




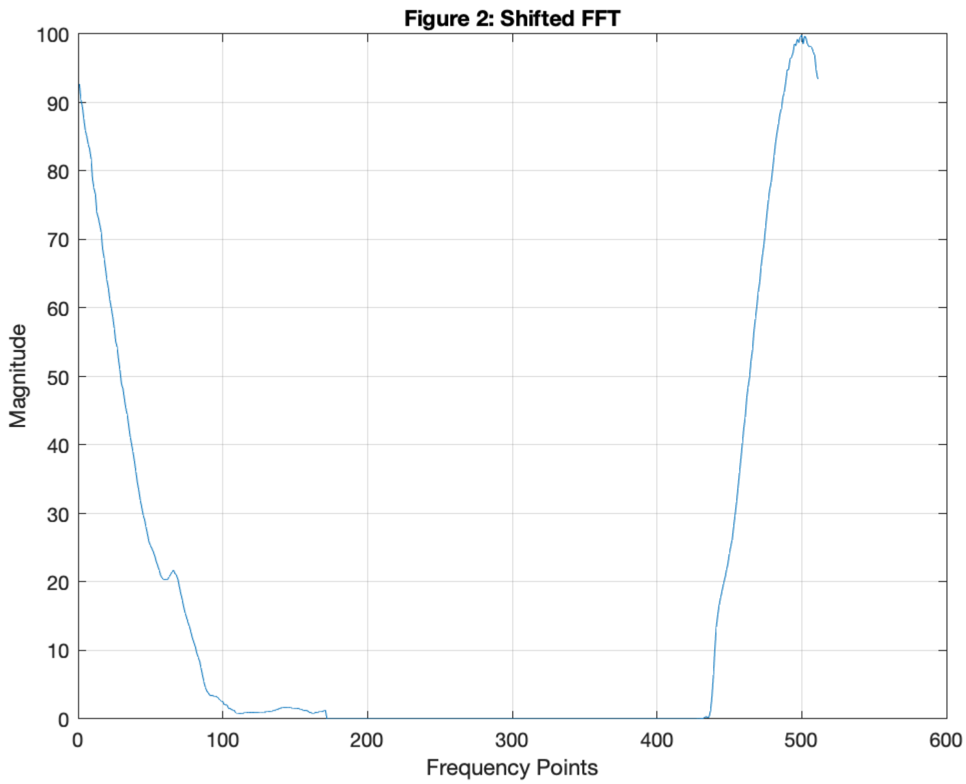
Appendix: Matlab Code for Each Parts (Live Script)

Part 1:

```
M = readmatrix("c0.txt")';
M = M(1:end-2)'; D = 512; L = 256;
upsampled = interp1(1:L, M(:, 1), linspace(1, L, 299), "spline");
upsampled = [upsampled zeros(1, (D - length(upsampled)))];

% Take the FFT and plot magnitude
Upsampled = (fftshift(fft(upsampled)));
tmp = linspace(-4800, 4800, 512);
figure();plot(tmp,abs(Upsampled)); title("Figure 1: FFT of Upsampled Signal");
xlabel("Frequency (Hz)"); ylabel("Magnitude"); grid on;
```

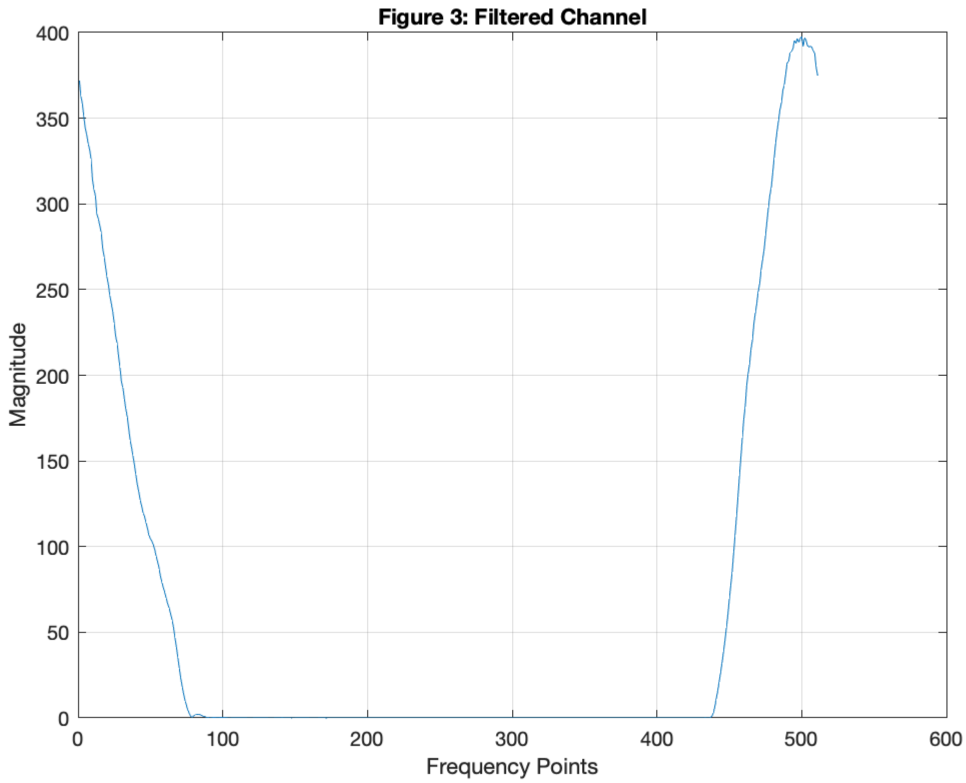



```
% Zero the negative frequencies and perform a cyclically shift
Upsampled(1:256) = zeros(1, 256);
Upsampled = circshift(Upsampled, -341); % Shift so that "max" is at sample 1
figure(); plot(abs(Upsampled)); title("Figure 2: Shifted FFT"); grid on;
xlabel("Frequency Points"); ylabel("Magnitude"); grid on;
```

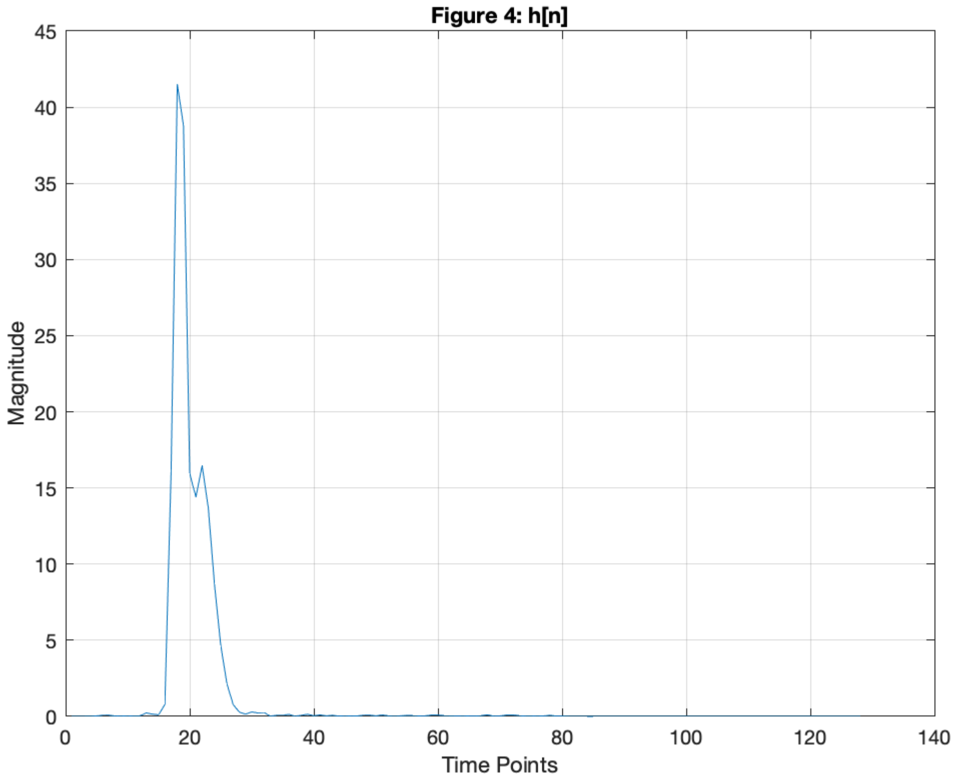



```
%generate srrc response and determine the index in which srrc dips below
%30db
N = 512; beta = 0.15;
srrc_signal = zeros(1, N);
for i=-256:255
    srrc_signal(i+257) = SRRC(beta, i);
end
thirty_dB = (srrc_signal(257) / sqrt(1000));
idx = 1;
for i=1:257
    if abs(srrc_signal(i)) > thirty_dB
        break;
    else
        idx = i;
    end
end

% Obtain the truncated SRRC filter and zero stuff the response
idx = idx + 1; M2 = 2*(257 - idx) + 1;
signal_new = zeros(1, M2);
for i=(-floor(M2 / 2)):(floor(M2 / 2))
    signal_new(i + (floor(M2 / 2)) + 1) = SRRC(beta,i);
end
signal_new = [signal_new zeros(1, (D - length(signal_new)))];
srrc_filter = (fft(signal_new));

% filtered channel response, and plot the filtered channel
channel = srrc_filter .* Upsampled;
```

```
figure(); plot(abs(channel)); title("Figure 3: Filtered Channel"); grid on;
xlabel("Frequency Points"); ylabel("Magnitude"); grid on;
```




```
% compute ifft and down sample
time_representation = ifft(channel, D); % 512-point IFFT of channel
impulse_response = time_representation(1:4:end);
figure(); plot(abs(impulse_response)); title("Figure 4: h[n]"); grid on;
xlabel("Time Points"); ylabel("Magnitude"); grid on;
```

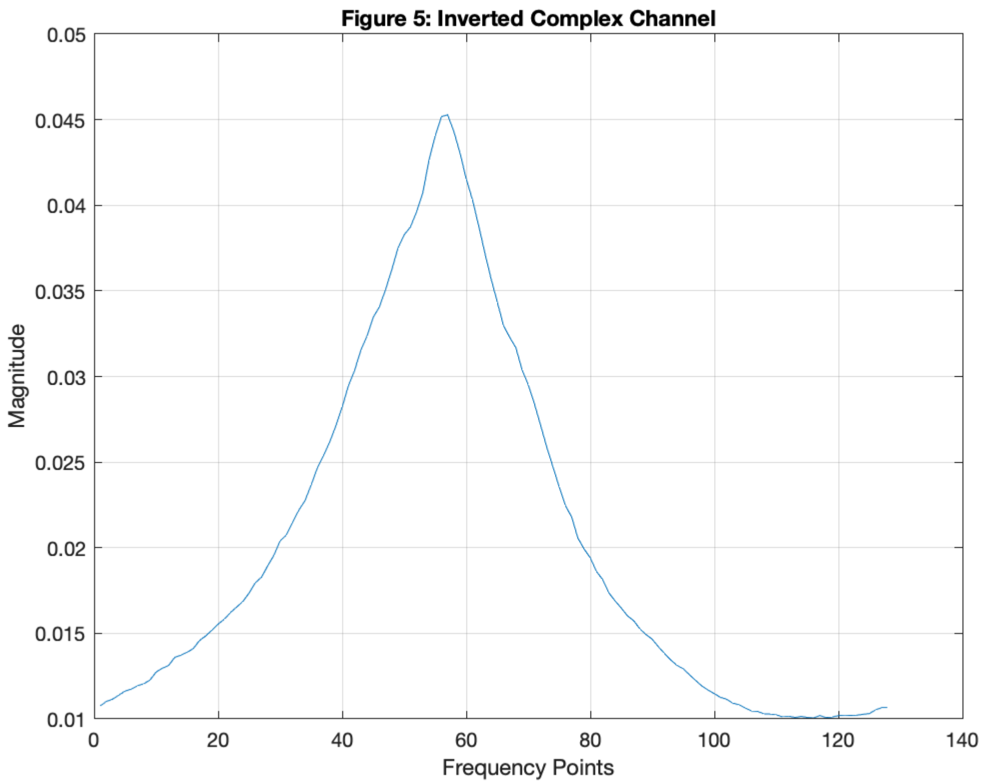



```
% sum of squares for power
hpower = 0;
for i=1:128
    hpower = hpower + abs(impulse_response(i))^2;
end

hpower
```

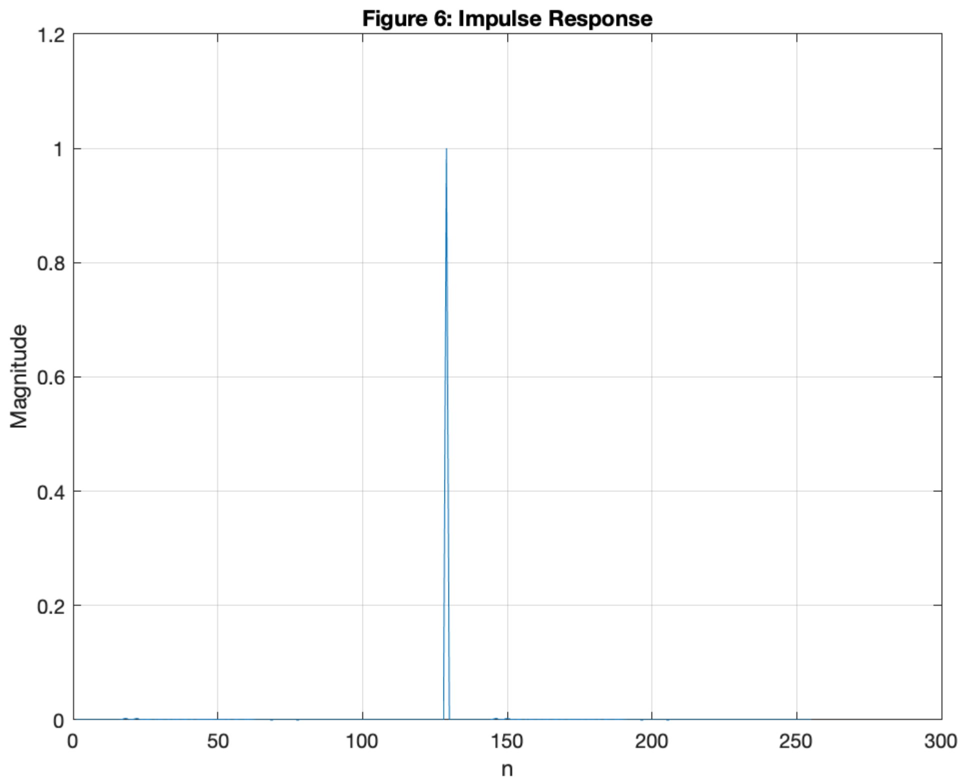
```
hpower = 4.5070e+03
```

Part 2:

```
impulse_response_fft = (fft(impulse_response));
%invert the equalizer
inv_imp_fft = 1./impulse_response_fft;
figure(); plot(abs(inv_imp_fft)); title("Figure 5: Inverted Complex Channel"); grid on;
xlabel("Frequency Points"); ylabel("Magnitude"); grid on;
```




```
%obtain time domain representation of the inverted equalizer
inv_imp_time_domain = ifft(inv_imp_fft);
%figure(); plot(abs(inv_imp_time_domain));
inv_conv_channel = conv(inv_imp_time_domain,impulse_response);
figure(); plot(abs(inv_conv_channel)); title("Figure 6: Impulse Response"); grid on;
xlabel("n"); ylabel("Magnitude"); grid on;
```




```
%sum of squared absolute values for the four peaks
f_peak = max(abs(inv_conv_channel).^2);
sq_ab = sum(abs(inv_conv_channel).^2);
abs_max = f_peak.^2;
SIR = abs_max/(sq_ab-abs_max)
```

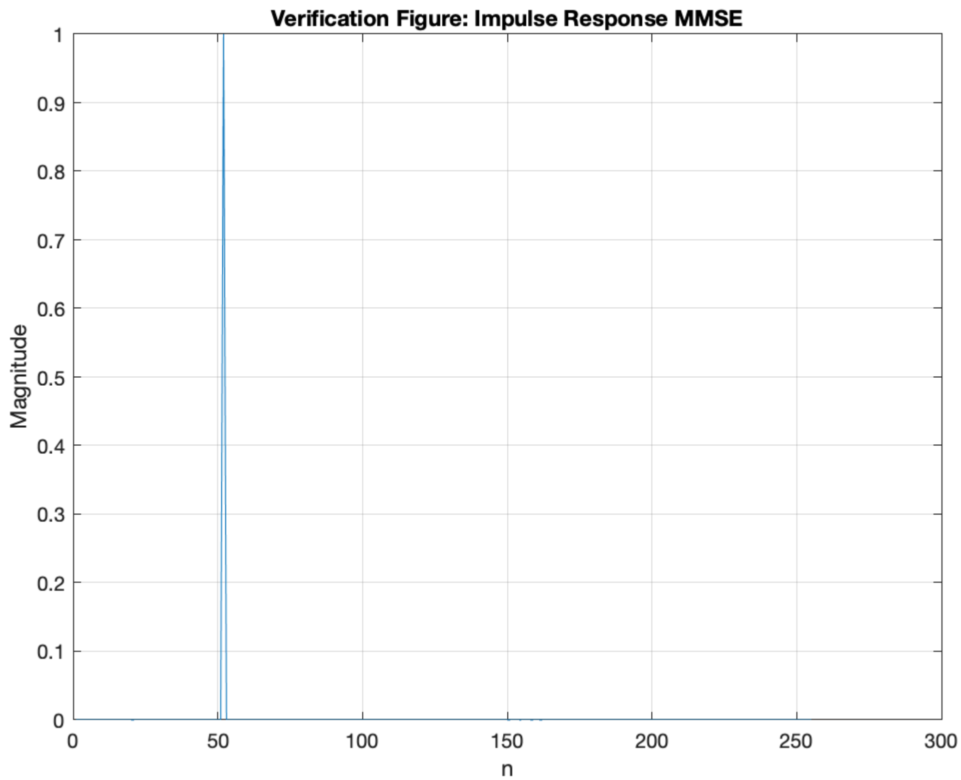
```
SIR = 1.0341e+04
```

```
nvar = 1/(sum(abs(impulse_response).^2) * 2000);

%compute the R matrix
R = zeros(128,128);
for i = 1:128
    for j = 1:128
        for k = 1:256
            con_h = conj(impulse_response);
            % make sure it's within the boundary
            if (k+1-i > 0) && (k+1-i <= 128) && (k+1-j > 0) && (k+1-j <= 128)
                R(i,j) = R(i,j) + con_h(1,k+1-i) * impulse_response(1,k+1-j);
            end
            if i == j
                R(i,j) = R(i,j) + 1/(sum(abs(impulse_response).^2) * 2000);
            end
        end
    end

end
```

```
%find the optimal d in which the SIR is minimized
%hd is the shifted h, re_delay is the optimal delay
hd=zeros(128,1);
re_delay = 0;
SIR = 0;
for d = 1:128
    hd_temp=zeros(128,1);
    for i = 1:128
        if d+1-i > 0
            hd_temp(i)=impulse_response(d+1-i);
        end
    end
    c_temp = R \ conj(hd_temp);
    temp_conv = conv(c_temp,impulse_response);
    m = max(abs(temp_conv));
    SIR_temp = m.^2/(sum(abs(temp_conv).^2)-m.^2);
    if SIR_temp > SIR
        SIR = SIR_temp;
        re_delay = d;
        hd = hd_temp;
        mmse = temp_conv;
    end
end

re_delay
```

```
re_delay = 52
```

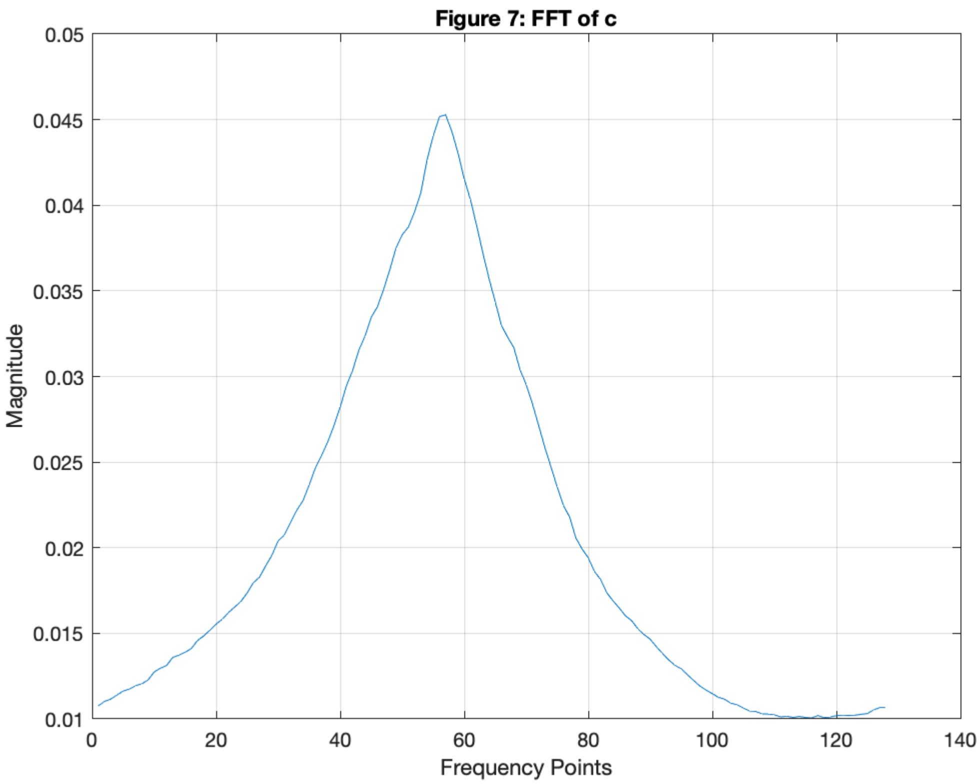
```
%compute c and convolve with h
c=R\conj(hd);
c_con_chan = conv(c,impulse_response);
%visualize the end product
figure();plot(abs(c_con_chan)); title("Verification Figure: Impulse Response MMSE"); gr
xlabel("n"); ylabel("Magnitude"); grid on;
```




```
%compute SIR
SIR_MMSE = max(abs(c_con_chan).^2)/(sum(abs(c_con_chan).^2)-abs(max(abs(c_con_chan))).^
```

```
SIR_MMSE = 1.5453e+09
```

```
%plot the fft of the equalizer
figure();plot(abs(fft(c))); title("Figure 7: FFT of c"); grid on;
xlabel("Frequency Points"); ylabel("Magnitude"); grid on;
```



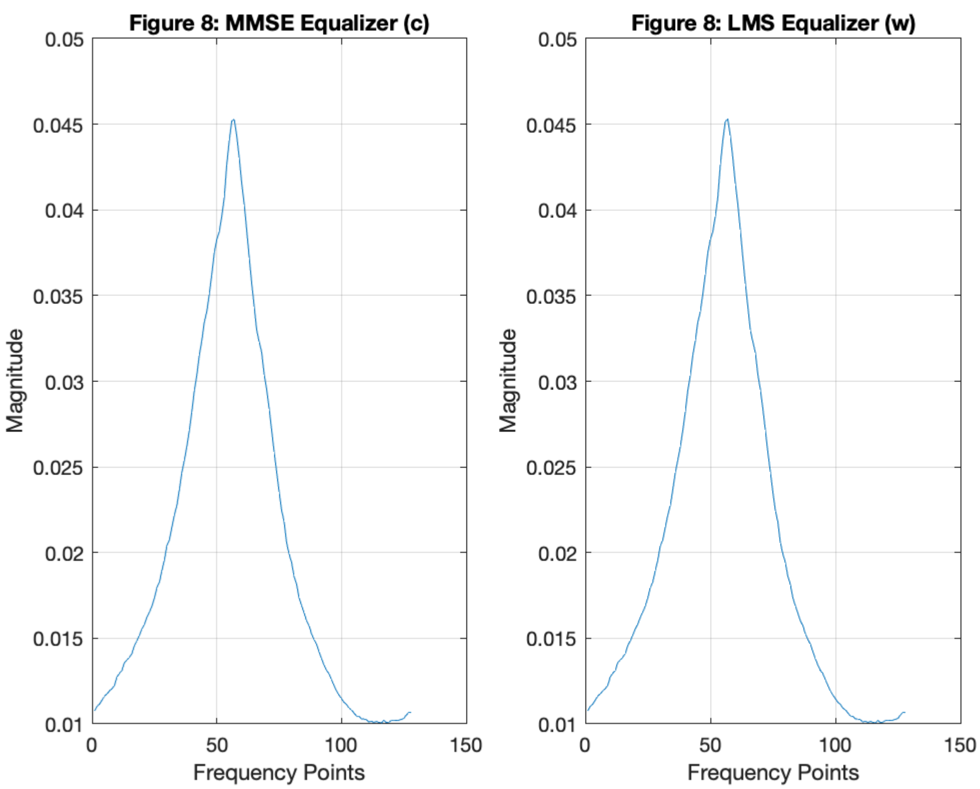
Part 3:

```
w = zeros(128, 1); w(d) = 1; del = 2 / (128 * hpower * 100); j = sqrt(-1);
data = zeros(128, 1); u = zeros(128, 1); squared_error = 0; lim = 3000000;

for iter=1:lim
    % Randomly compute sampled data
    val = randsample([-1, 1], 1) * (1/sqrt(2)) + randsample([-1, 1], 1) * (j/sqrt(2));
    data = circshift(data, 1); data(1) = val;
    % zero mean complex Gaussian noise
    z = (normrnd(0, sqrt(nvar)) + j*normrnd(0, sqrt(nvar))) / sqrt(2);

    % Compute channel output
    val = 0;
    for i=1:128
        val = val + impulse_response(i) * data(i);
    end

    % Cyclically shift channel output into output array, and add val in front
    u = circshift(u, 1); u(1) = val + z;

    % Calculate the error
    y = 0;
    for i=1:128
        y = y + u(i) * w(i);
    end
    e = data(re_delay) - y;
```

```
    % Calculate squared error when iterations surpasses 95 percent
    % benchmark
    if iter >= 0.95 * lim
        squared_error = squared_error + abs(e)^2;
    end
    % Adjust the step size and update the w in gradient descent
    if iter > 1000000
        del = 2 / (128 * hpower * 1000);
    end
    w = w + del * e * conj(u);
end
%calculate the SINR
SINR = 0.05 * lim / squared_error
```

```
SINR = 1.4180e+09
```

```
% Plot comparison using subplot, since overlapping would result in a
% non-discernable graph
figure();
subplot(1, 2, 1); plot(abs(fft(c))); title("Figure 8: MMSE Equalizer (c)");
xlabel("Frequency Points"); ylabel("Magnitude"); grid on;
subplot(1, 2, 2); plot(abs(fft(w))); title("Figure 8: LMS Equalizer (w)");
xlabel("Frequency Points"); ylabel("Magnitude"); grid on;
```




```
% LMS Adaptive with Interference Equalizer
w2 = zeros(128, 1); w2(re_delay) = 1; data2 = zeros(128, 1); u2 = zeros(128, 1);
```

```
j = sqrt(-1); squared_error2 = 0; lim = 5000000;

for iter=1:lim
    % Randomly compute sampled data
    val = randsample([-1, 1], 1) * (1/sqrt(2)) + randsample([-1, 1], 1) * (j/sqrt(2));

    % Cyclically shift channel output into output array, and add val in front
    data2 = circshift(data2, 1); data2(1) = val;
    % channel output
    val = 0;
    for i=1:128
        val = val + impulse_response(i) * data2(i);
    end
    val = val + sqrt(10) * exp(-sqrt(-1) * pi * iter / 3);

    % Cyclically shift channel output into output array, and add val in front
    u2 = circshift(u2, 1); u2(1) = val;

    % Calculate the error
    y = 0;
    for i=1:128
        y = y + u2(i) * w2(i);
    end
    e = data2(re_delay) - y;
    % Adjust the step size and update the w in gradient descent
    if iter >= 0.95 * lim
        squared_error2 = squared_error2 + abs(e)^2;
    end
    w2 = w2 + del * e * conj(u2);
end

SINR2 = 0.05 * lim / squared_error2
```

```
SINR2 = 1.0591e+03
```

```
% Plot comparison between noise and interference
figure();
subplot(1, 2, 1); plot(abs(fft(w))); title("Figure 9: LMS w/ Noise (w)");
xlabel("Frequency Points"); ylabel("Magnitude"); grid on;
subplot(1, 2, 2); plot(abs(fft(w2))); title("Figure 9: LMS w/ Interferer (w2)");
xlabel("Frequency Points"); ylabel("Magnitude"); grid on;
```

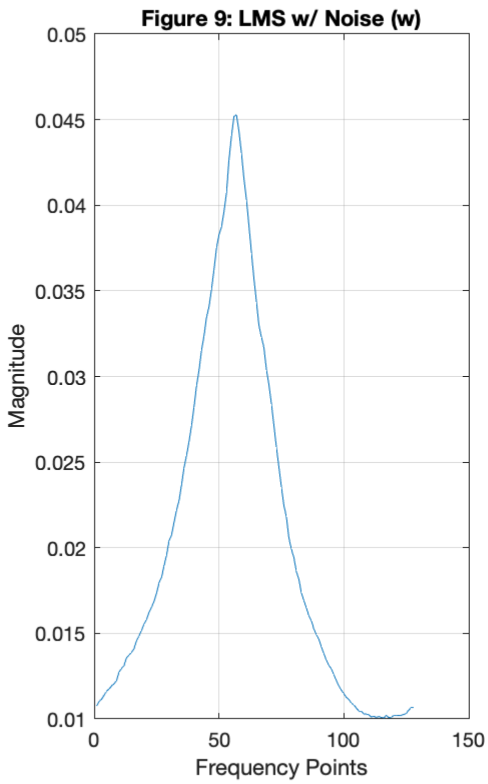


Figure 9: LMS w/ Noise (w)
Magnitude
Frequency Points
0.05
0.045
0.04
0.035
0.03
0.025
0.02
0.015
0.01
0
50
100
150

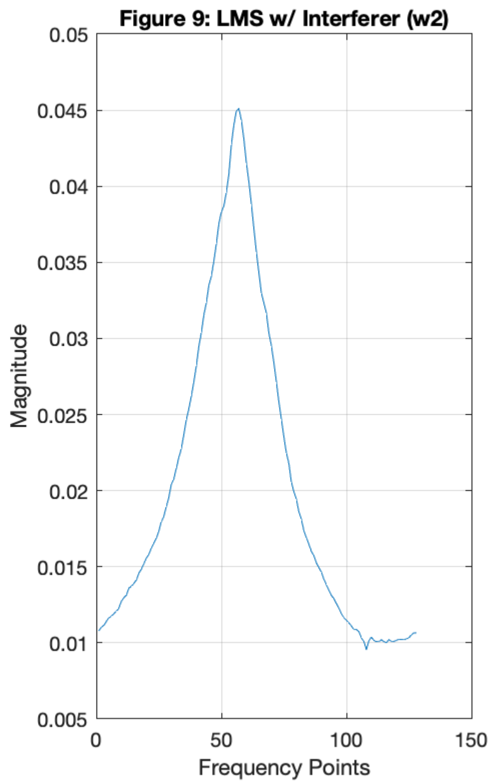


Figure 9: LMS w/ Interferer (w2)
Magnitude
Frequency Points
0.05
0.045
0.04
0.035
0.03
0.025
0.02
0.015
0.01
0.005
0
50
100
150