Adaptive Filtering

Nima Leclerc (nleclerc@seas.upenn.edu)
ESE 531 (Digital Signal Processing)
School of Engineering and Applied Science
University of Pennsylvania

May 2, 2021

1 Implementation of Adaptive Notch Filter

1.1 Simulation of Adaptive Notch Filter

Here we consider an IIR notch filter with two zeros on the unit circle and two poles inside the unit circle. The filter has the following form,

$$H(z) = \frac{(1 - e^{j\omega_0} z^{-1})(1 - e^{-j\omega_0} z^{-1})}{(1 - re^{j\omega_0} z^{-1})(1 - re^{-j\omega_0} z^{-1})} = \frac{1 + az^{-1} + z^{-2}}{1 + raz^{-1} + r^2 z^{-2}}$$

The notch frequency is at ω_0. We can use this filter to reject strongly interfering sinusoids that have some desired frequency. We can write the corresponding impulse response of the filter to get the error $y[n]$ for which we want to minimize.

$$e[n] = x[n] + ax[n-1] + x[n-2]$$

$$y[n] = e[n] - ray[n-1]c - r^2 y[n-2]$$

where $x[n]$ is the input signal. We must employ gradient descent to optimize a which parameterizes our filter. This requires,

$$\nabla_a E[y^2] = 2y[n]\nabla_a y[n] \approx 2y[n]\nabla_a e[n] = 2y[n]x[n-1]$$

Hence, we will take our update step over the course of the adaptation to be,

$$a[n+1] = a[n] - \mu \nabla_a E[y^2]$$

$$\approx a[n] - \mu 2y[n]x[n-1]$$

with $a[n] = -2\cos(\omega_0[n])$. The algorithm now used to optimize a is given by Algorithm 1.

Algorithm 1: Adaptive Notch Filter

```
y = [] ;
a = [] ;
x ← input sequence of length $N$;
a[0] ← 0 ;
r ← initialized $r$ parameter;
$\mu$ ← initialized learning rate;
for  i=3:N do
    $e[i] \leftarrow x[i] + a[i]x[i-1] + x[i-2]$ ;
    $y[i] \leftarrow e[i] - ra[i]y[i-1] - r^2y[i-2]$ ;
    if  $a[i] < 2$ and $a[i] \geq -2$ then
        $a[i+1] \leftarrow a[i] - \mu y[i]x[i-1]$ ;
    else
        a[i] = 0 ;
    end
end
return a
```

We start with the following test input sequence

$$x^1_{test}[n] = \sin(\pi n), -50 \leq n \leq 50$$

$$x^2_{test}[n] = 20\sin(\pi n) + \sin(\pi/2n) + \sin(\pi/4n), -50 \leq n \leq 50$$

In summary, the algorithm takes in an input signal and is then fed into an adaptive filter which is parameterized by a and r. Here, we fix r but allow a to be updated over the course of the optimization. Once convergence is reached, the optimized parameter a can be extracted. From this, we can relate the optimized a to ω_0 from the relation $a = -2\cos\omega_0$. We need to set r and μ, hence we take $r = 0.85, 0.90, 0.97$ and $\mu = 0.001, 0.01, 0.1$. Plotted in Figure 1 are the convergence plots of a over iteration time with $x^1_{test}[n]$ as the input, for μ fixed to 0.001 and $r = 0.85, 0.90, 0.97$ (Figure 1(a)) and for r fixed to 0.85 and $\mu = 0.001, 0.01, 0.1$ (Figure 1(b)). From this, it is clear that the settings for r and μ have a significant impact on the convergence of a over iteration number. For $\mu = 0.001$, a converges to -1.149, -1.145, and -1.1422 for $r = 0.97, 0.9, 0.85$. For r at 0.85, we see see convergence of a to $-1.14, -0.62$ for $\mu = 0.001, 0.01$, however a fails to converge for large $\mu = 0.1$.

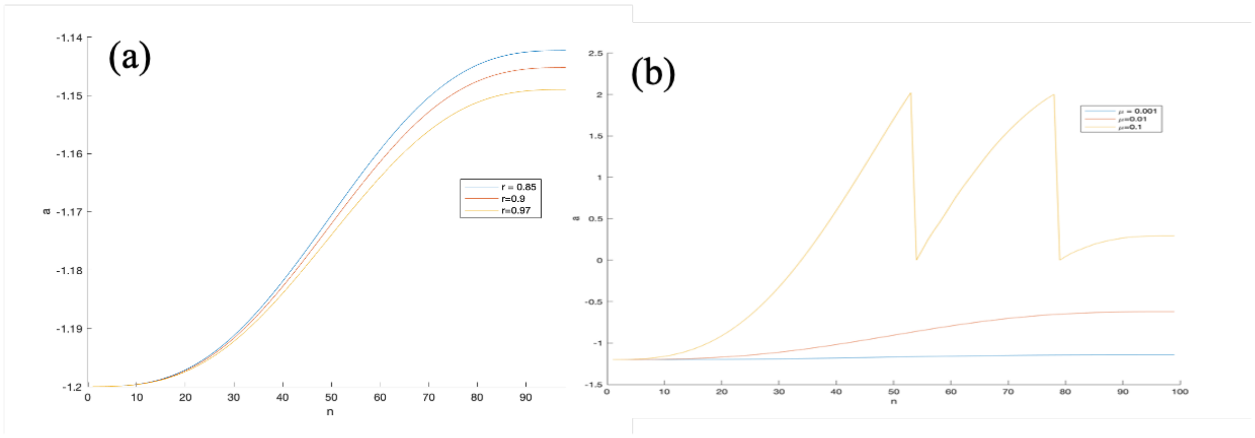


Figure 1: Convergence of adaptive notch filter parameter a. (a) Dependence on r parameter for $r = 0.85$ (blue), $r = 0.90$ (red) , and $r = 0.97$ (yellow). (b) Dependence on learning rate μ parameter for $\mu = 0.001$ (blue), $\mu = 0.01$ (red), $\mu = 0.1$ (yellow).

From this, we pick $\mu = 0.001$ and $r = 0.97$ to be the optimal choice of hyperparameters. Hence, we can proceed to look at the frequency and impulse response of the filter applied on both input signals $x^1_{test}[n]$ and $x^2_{test}[n]$ where $x^2_{test}[n]$ has a strongly interfering signal with frequency $\omega = \pi$. Shown in Figure 2(a) is the magnitude of the frequency response for these set of parameters with $a = -1.149$, where we see that the frequency of the notch is fixed at the value corresponding to our converged parameter. Figure 2(b)/(c) show the impulse response of the input signals $x^1_{test}[n]$ and $x^2_{test}[n]$. Figure 2(d) shows the output of the optimized filter $h[n]$ applied on input signal 1 and likewise applied on signal 2 in figure 2(e). We can see that the optimization of the notch filter missed the frequency of the signal as the output signals are unchanged. However, this problem can be addressed by performing further hyperparameter tuning and improving the training process to achieve lower training errors.

.

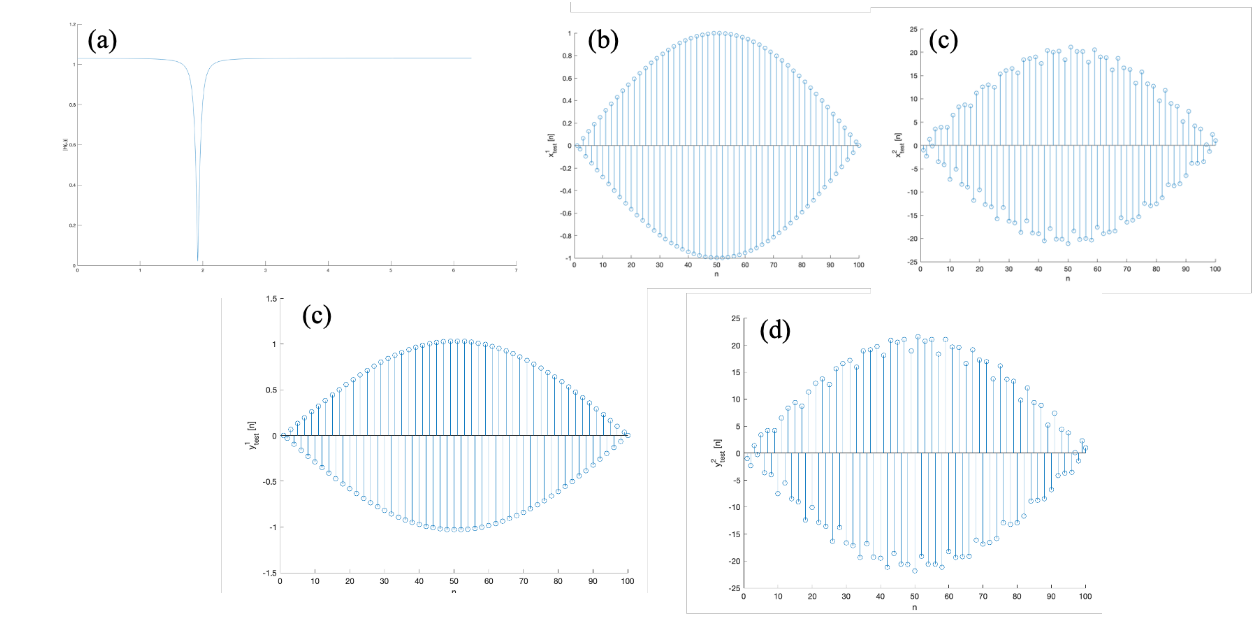


Figure 2: Input/output from adaptive notch filter. (a) Magnitude of frequency response for optimized notch filter. (b) Test input signal $x^1[n]$. (c) Test input signal $x^2[n]$. (d) Channel output from optimized filter with test input signal $x^1[n]$. (e) Channel output from optimized filter with test input signal $x^2[n]$.

1.2 Adaptive filter for linear varying frequency input signal

Here we consider the case where our input signal has a frequency that changes linearly in time. In particular, we consider signals of the form below.

$$x_{in} = \cos(2\pi\phi(t))$$

where $d\phi/dt = ct$ with c as a positive constant. Here, we can take $c = 0.7$. Hence, the input signal will be

$$x_{in}[n] = \cos(2\pi(0.7)n^2)$$

We apply our adaptive notch filter algorithm here to find the optimal a, with $\mu = 0.001$ during training and $r = 0.9$. (initialized $a = 0$). Following the optimization, a converged parameter of $a = -0.0045$ is found. The input signal to this filter is shown in Figure 3(a). The convergence plot of the algorithm is shown in Figure 3(b), illustrating a noisy descent over the course of training for 100 time steps. Following the optimization, the converged parameter a is used to construct the adapted notch filter which has a magnitude frequency response plotted in Figure 3(c). Applying this to the input signal, the output of the channel is plotted in Figure 3(d) illustrating that the filter was not very effective in filtering out the noisy components of the signal. This can be attributed to the fact that the input signal has a continuous range of frequencies with equal magnitude and the algorithm struggled to find the maximally interfering component.

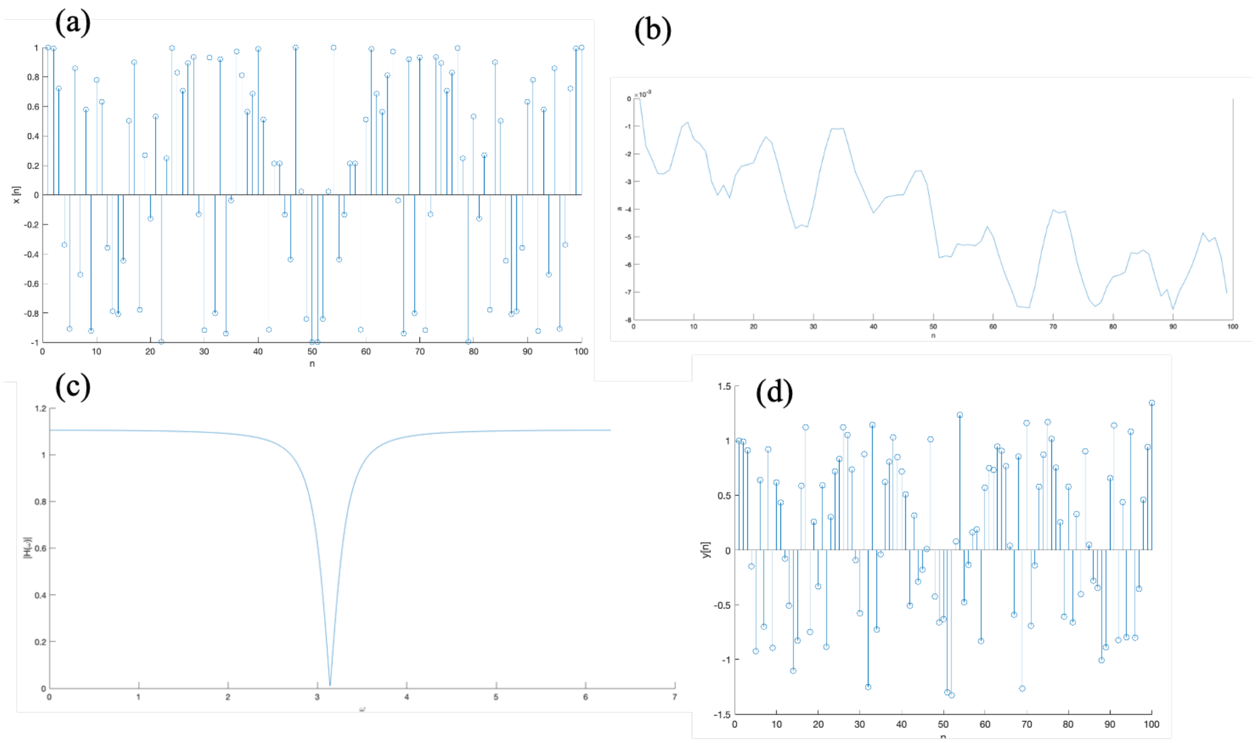


Figure 3: Input/output from adaptive notch filter with linearly varying input signal frequency. (a) Test input signal $x_{in}[n] = \cos(2\pi(0.7)n^2)$. (b) Convergence of a parameter versus time-step. (c) Frequency response of optimized notch filter. (d) Filtered output of channel $y[n]$ with converged filter.

2 Adaptive Equalization

Here we consider an input length sequence sending pulses with amplitudes $\pm A$ which represent logical '1' and '0' bits. Using this we can construct a model for an input signal $s[n]$ entering a noisy channel, yielding an output $x[n] = h[n] * s[n] + w[n]$. $h[n]$ is an LTI system representing the channel and $w[n]$ is added Gaussian white noise. We take the channel order to be L and the channel output of the FIR adaptive filter to be greater than $M + 1$- this yields a delayed version of the input.

In this implementation, an initial random sequence of length 1,000 with samples ± 1 is generated to represent $s[n]$. Next, a filter $h[n]$ is applied which is initially chosen to be $h[n]$ randomly for some specified filter order m. Finally, Gaussian random noise $w[n]$ is added to the output $w[n]$ to give $x[n] = h[n] * x[n] + w[n]$. This implementation is summarized in Algorithm 2.

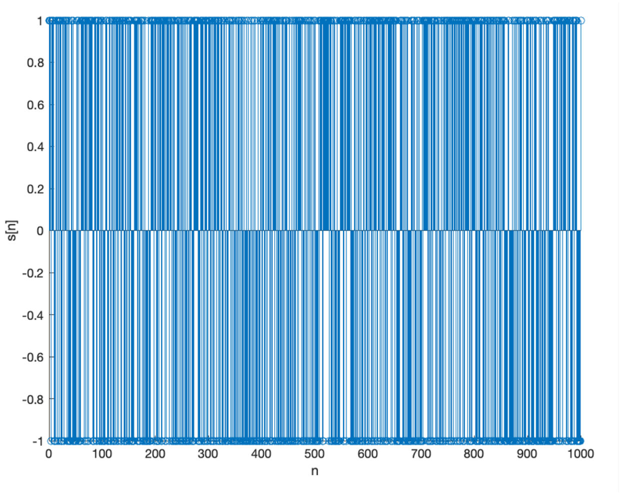


Figure 4: Input binary signal $s[n]$.

For all implementations in this section, we consider the input signal $s[n]$ shown in Figure 4. We then consider how the convergence of the filter parameters vary under three degrees of freedom: (1) step size μ, (2) filter order M, and (3) signal-to-noise ratio (SNR) of added Gaussian white noise SNR. For each of these cases, we evaluate the training performance and testing performance.

Algorithm 2: Adaptive Equalization Filter

```
$N \leftarrow$ size of input signal;
$h_0 \leftarrow$ initialize filter weights;
$\mu \leftarrow$ initialize step size (learning rate) ;
$M$ = length($h_0$) ;
$s \leftarrow$ initialize input sequence of size $N$ as random binary sequence of $\pm 1$ ;
$d \leftarrow h_0 * s + \epsilon(SNR)$ initializes test output sequence with added Gaussian noise
  with SNR ;
$weights = zeros(N, M)$ ;
Initializes weight matrix to 0
for $i$=$M + 1$:$N - M$ do
    $x = s[i - M : i - 1]$ ;
    $err[i] \leftarrow d[i] - weights[i]^T$ x ;
    $weights[i + 1] \leftarrow weights[i] + 2\mu err[i]x$ ;
end
return $weights, err, d, s$
```

μ-dependence:
We start with analyzing the dependence of filter adaptation on step size. We consider settings of $\mu = 0.001$ and $\mu 0.01$, as step sizes beyond this range lead to excessively noisy convergence (we set $M = 4$ and $SNR = -30$dBW. Plotted in Figure 5(a) is the mean squared training error over 1,000 iterations for each setting of μ. From this, it is clear that $\mu = 0.01$ achieves convergence much more quickly and with a lower noise floor than $\mu = 0.001$ indicating that this is an optimal choice. Plotted in Figure 5(b) is the desired signal used to validate the

estimated filter output during training. In Figure (c) we see the impulse response of the LTI filtered output. Figure 5(d) shows the output of the signal going into the converged channel for each setting of μ. The impulse response of the filter is also shown in Figure 5(e), indicating that only the amplitude of the filter response changes with varied μ. Figure 5(f) shows the corresponding magnitude of the frequency response for each setting of μ in log-scale, again revealing similar shapes but different magnitudes. Finally, we have the frequency response of the output $Y(\omega)$ of the LTI system for each μ. Both responses show similar orders of magnitude, but seem to be slightly out of phase in frequency. Collectively, we find that the general filter characteristics after convergence for varied μ remain the same. However, the noise and training performance of each filter is different as $\mu = 0.01$ reveals a relatively low noise-amplitude. For completeness, we plot the pole-zero diagram of the system in Figure 8 for $\mu = 0.001$ (8(a)) and $\mu = 0.01$ (8(b)) revealing 5 zeros for $\mu = 0.001$ with 1 outside the unit circle and 4 zeros with 2 outside the unit circle and 1 pole at the origin for $\mu = 0.01$.

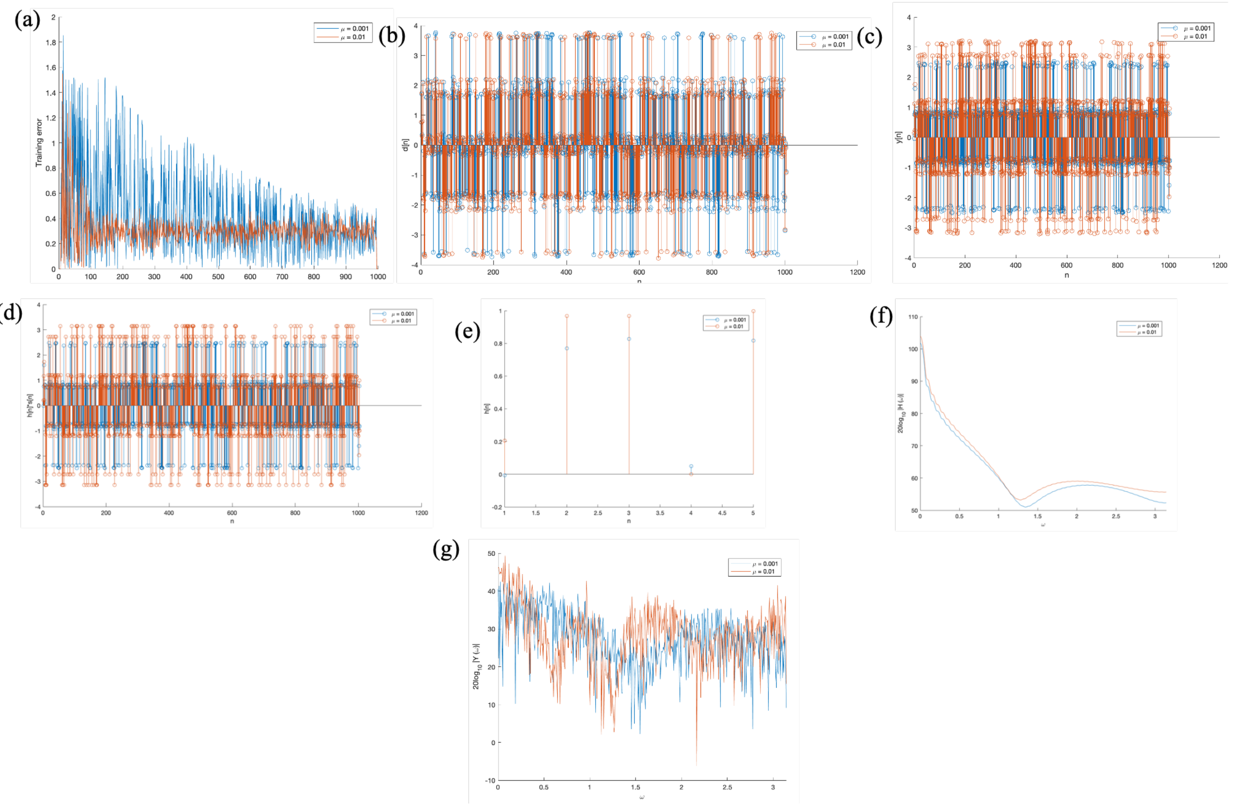


Figure 5: Adaptive equalization filter results for varied step size μ. (a) Training error $e[n]$ with iteration number for $\mu = 0.001$ (blue) and $\mu = 0.01$ (orange) . (b) Desired output from channel $d[n]$ for $\mu = 0.001$ (blue) and $\mu = 0.01$ (orange). (c) Impulse response of output from LTI system with converged filter parameters for $\mu = 0.001$ (blue) and $\mu = 0.01$ (orange). (d) Impulse response of output from channel with converged filter parameters for $\mu = 0.001$ (blue) and $\mu = 0.01$ (orange). (e) Impulse response of filter $h[n]$ for $\mu = 0.001$ (blue) and $\mu = 0.01$ (orange). (f) Magnitude of frequency response of optimized filter in log-scale [dB] for $\mu = 0.001$ (blue) and $\mu = 0.01$. (g) Magnitude of frequency response of optimized LTI system in log-scale [dB] for $\mu = 0.001$ (blue) and $\mu = 0.01$.

Order dependence:
We next look at the dependence on filter order. We consider settings of $M = 5, 10, 20, 30$ (setting $\mu = 0.01$ and $SNR = -30$dBW. Plotted in Figure 6(a) is the mean squared training error over 1,000 iterations for each setting of M. From this, it is clear from Figure 6(a) that the error upon convergence increases monotonically with filter order. Plotted in Figure 6(b) is the desired signal used to validate the estimated filter output during training. In Figure 6(c) we see the impulse response of the LTI filtered output. The impulse response of the filter is also shown in Figure 6(d), indicating that both the amplitude and time instances of the of the filter response changes with varied M which is expected. Figure 6(e) shows the corresponding magnitude of the frequency response for each setting of M indicating a steeper descent near 0 frequency with increased filter order. Finally, we have the frequency response of the output $Y(\omega)$ of the LTI system for each M. Both responses show similar orders of magnitude, but seem to be slightly out of phase in frequency. Collectively, we

find that the general filter characteristics after convergence for varied M differ and show monotonic trends. For completeness, we plot the pole-zero diagram of the system in Figure 9 for each M. Each plot shows similar characteristics, however the number of zeros and their locations differ.

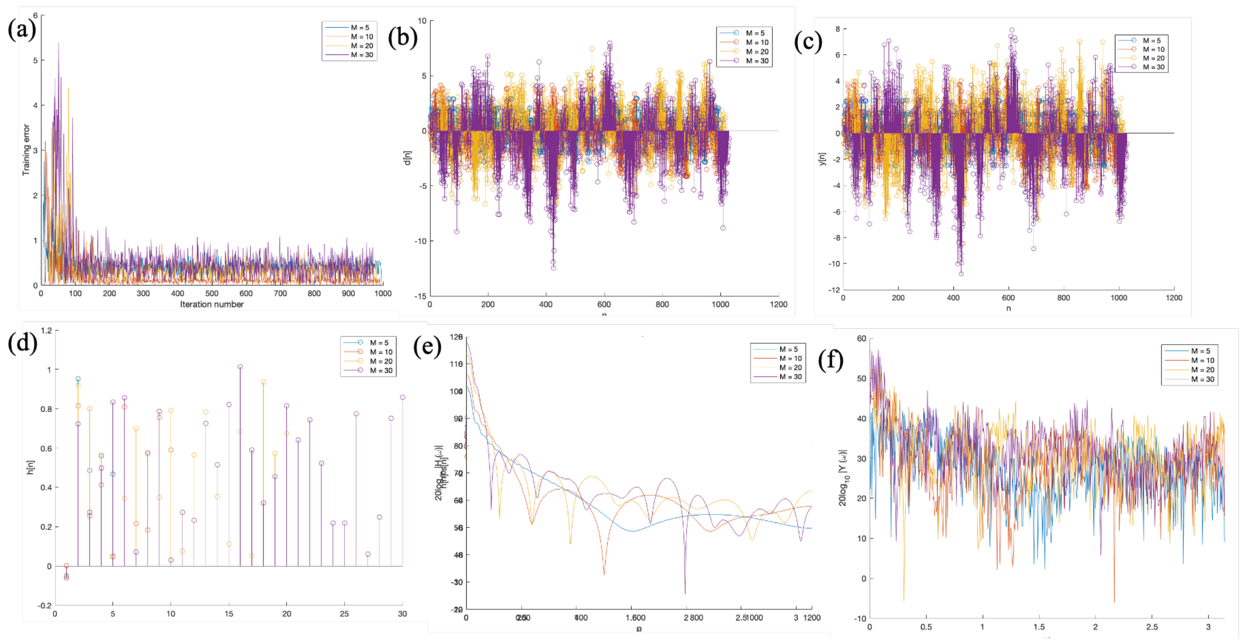


Figure 6: Adaptive equalization filter results for varied filter order M. (a) Training error $e[n]$ with iteration number for $M = 5$ (blue), $M = 10$ (red) , $M = 20$ (yellow) , and $M = 30$ (purple). (b) Desired output from channel $d[n]$ for $M = 5$ (blue), $M = 10$ (red) , $M = 20$ (yellow) , and $M = 30$ (purple). (c) Impulse response of output from LTI system with converged filter parameters for for $M = 5$ (blue), $M = 10$ (red) , $M = 20$ (yellow) , and $M = 30$ (purple). (d) Impulse response of output from channel with converged filter parameters for $M = 5$ (blue), $M = 10$ (red) , $M = 20$ (yellow) , and $M = 30$ (purple). (e) Magnitude of frequency response of optimized filter in log-scale [dB] for for $M = 5$ (blue), $M = 10$ (red) , $M = 20$ (yellow) , and $M = 30$ (purple). (f) Magnitude of frequency response of optimized LTI system in log-scale [dB] for $M = 5$ (blue), $M = 10$ (red) , $M = 20$ (yellow) , and $M = 30$ (purple).

SNR dependence:
Finally, we consider the dependence of filter adaptation on magnitude of SNR for added Gaussian noise. We consider settings of $SNR = -30$dBW and $SNR = -5$ dBW. Plotted in Figure 7(a) is the mean squared training error over 1,000 iterations for each setting of SNR revealing a significantly noisier training performance for the -5 dBW setting which is expected due an inherently higher noise floor. Plotted in Figure 7(b) is the desired signal used to validate the estimated filter output during training. In Figure 7(c) we see the impulse response of the LTI filtered output. Figure 7(d) shows the output of the signal going into the converged channel for each setting of SNR. The impulse response of the filter is also

shown in Figure 7(e), indicating that only the amplitude of the filter response changes with varied SNR. Figure 7(f) shows the corresponding magnitude of the frequency response for each setting of SNR in log-scale, again revealing similar shapes but different magnitudes. Finally, we have the frequency response of the output $Y(\omega)$ of the LTI system for each μ. Both responses show similar orders of magnitude, but seem to be slightly out of phase in frequency. Collectively, we find that the general filter characteristics after convergence for varied SNR remain the same. However, the noise in responses tends to be higher for -5 dBW SNR setting. For completeness, we plot the pole-zero diagram of the system in Figure 10 for each SNR setting.

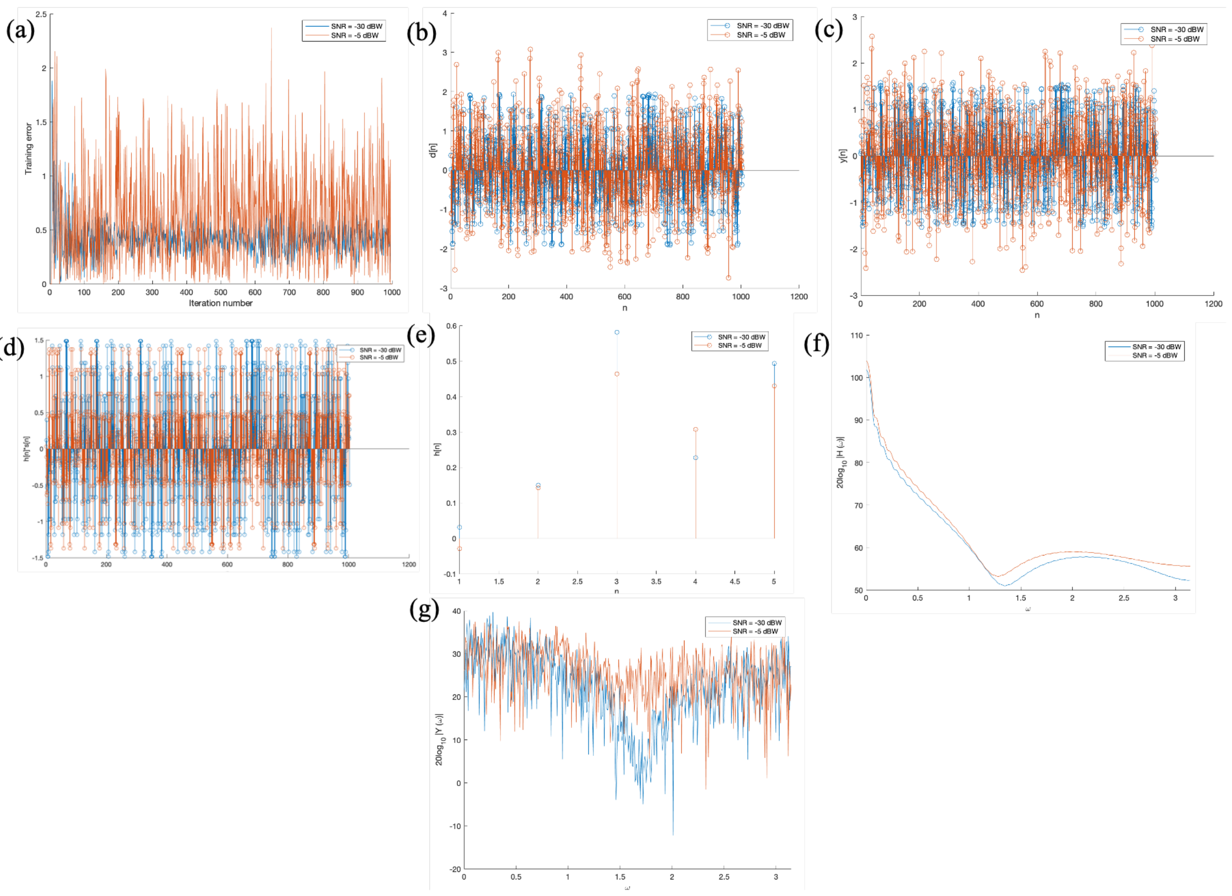


Figure 7: Adaptive equalization filter results for varied SNR. (a) Training error $e[n]$ with iteration number for $SNR = -30$ dBW (blue) and $SNR = -5$dBW (orange) . (b) Desired output from channel $d[n]$ for for $SNR = -30$ dBW (blue) and $SNR = -5$dBW (orange) . (c) Impulse response of output from LTI system with converged filter parameters for $SNR = -30$ dBW (blue) and $SNR = -5$dBW (orange) . (d) Impulse response of output from channel with converged filter parameters for $SNR = -30$ dBW (blue) and $SNR = -5$dBW (orange) (orange). (e) Impulse response of filter $h[n]$ for $SNR = -30$ dBW (blue) and $SNR = -5$dBW (orange) . (f) Magnitude of frequency response of optimized filter in log-scale [dB] for $SNR = -30$ dBW (blue) and $SNR = -5$dBW (orange) . (g) Magnitude of frequency response of optimized LTI system in log-scale [dB] for $SNR = -30$ dBW (blue) and $SNR = -5$dBW (orange)

Among the different cases examined (varied SNR, μ, and M) we observed not only a change in the output signals after equalization with respect to the input signals, but also great variation in the response with respect to these three degrees of freedom. In particular, the choice of filter order and SNR had the greatest impact on the output signals after equalization. In summary, this reveals the immense tuneability that one has in engineering the filter response over a larger hyper-parameter space through adaptive filtering. Here we only investigated three possible degrees of freedom that one can tune, however there exist many others that were not discussed in this work.

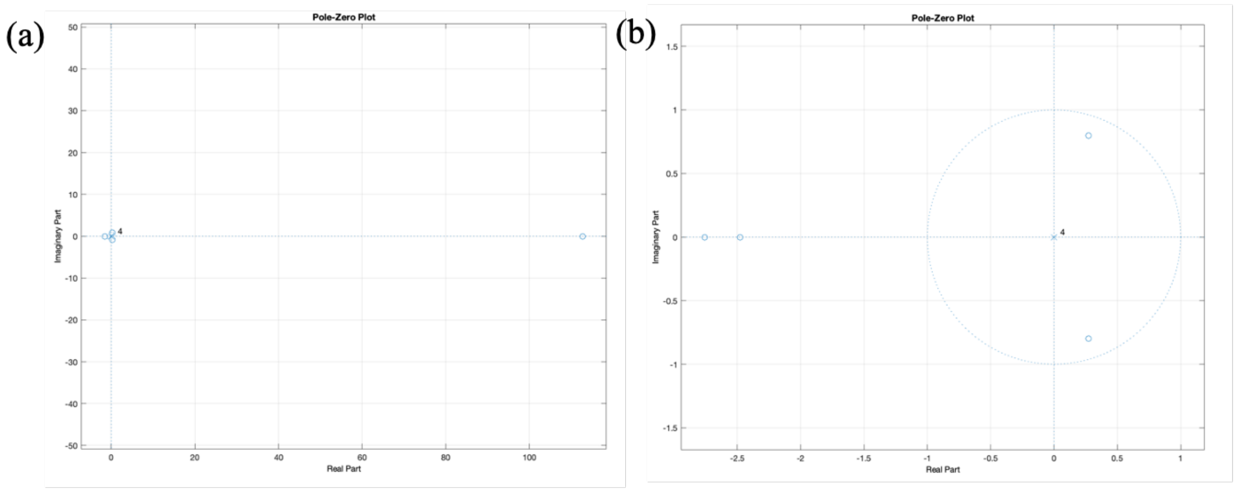


Figure 8: Pole-zero plot of optimized filters with varied μ. (a) $\mu = 0.001$. (b)$\mu = 0.01$.

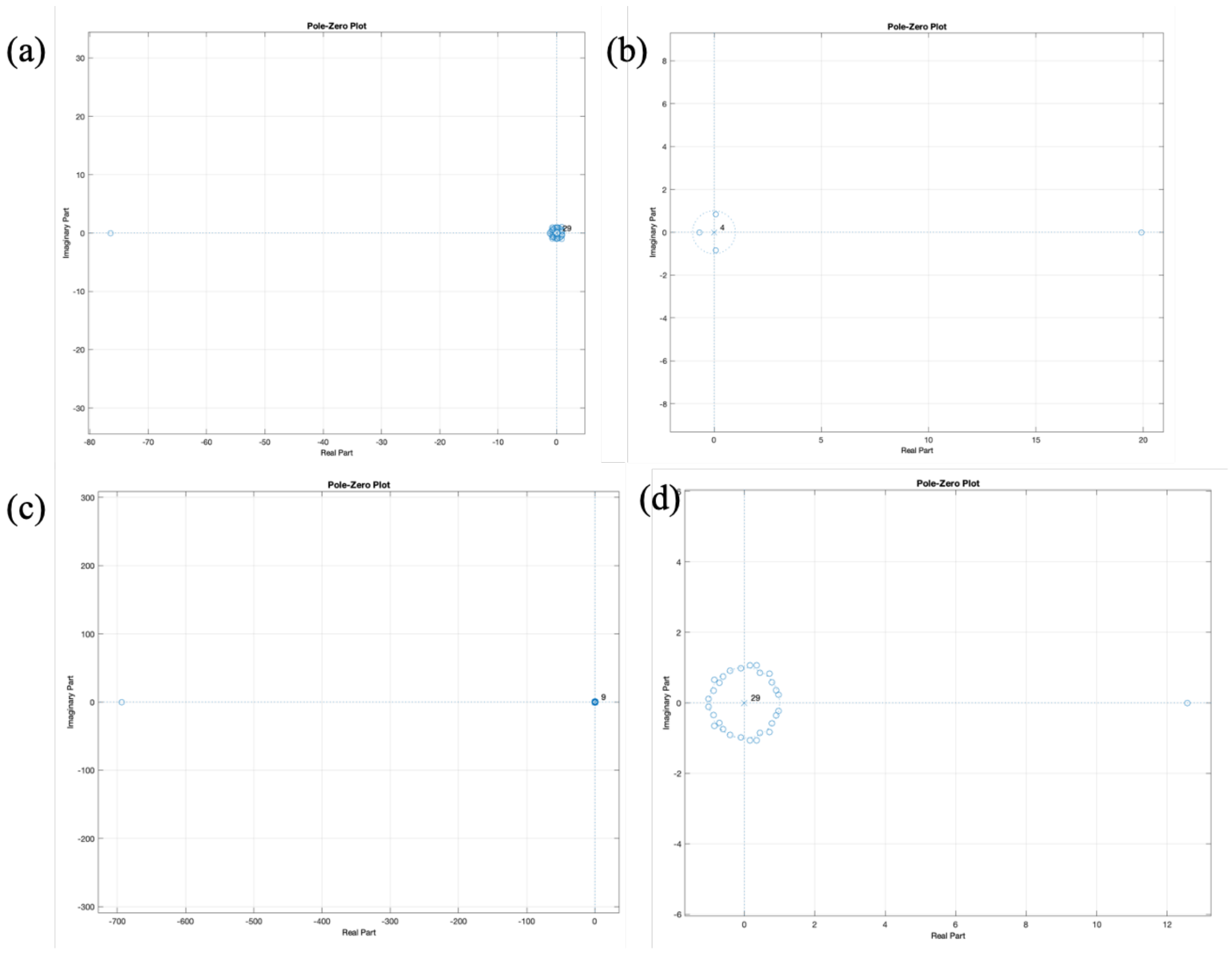


Figure 9: Pole-zero plot of optimized filters with varied filter order M. (a) $M = 5$. (b)$M = 10$. (c)$M = 20$. (d)$M = 30$.

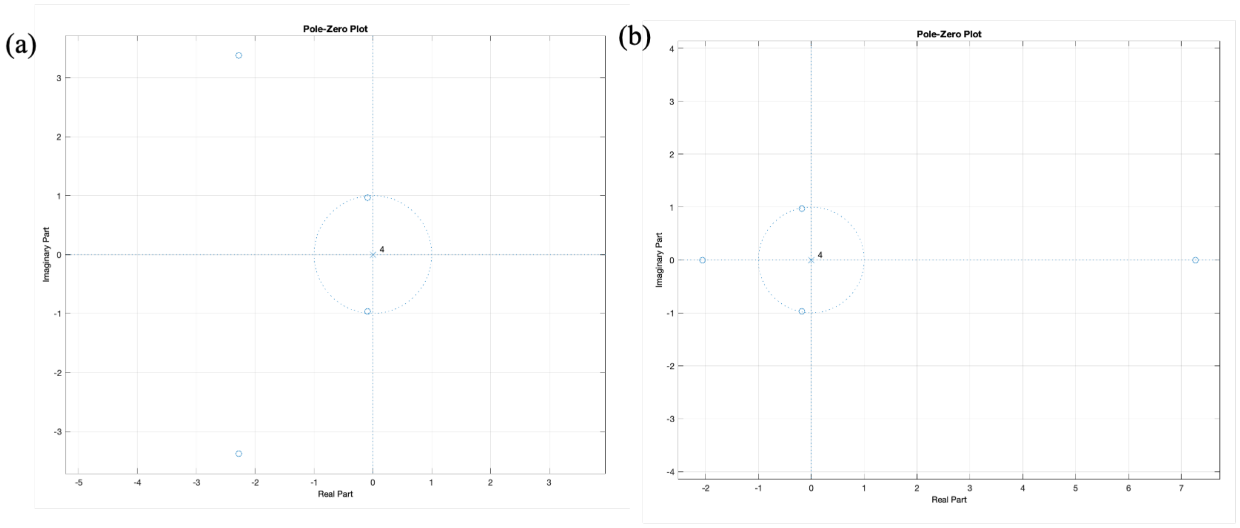


Figure 10: Pole-zero plot of optimized filters with varied SNR. (a) $SNR = -30$ dBW. (b)$SNR = -5$ dBW.

3 Adaptive Blind Equalization

Now we investigate the use of adaptive blind equalization in implementing adaptive filters to noisy binary input signals. Moreover, we will contrast this algorithm with the usual implementation. This algorithm is summarized in Algorithm 3. The key distinction between Algorithm 3 and Algorithm 2 discussed in the previous section is that the error computed involves the error of the norm of the predicted and desired outputs. Hence, we consider the form

$$e_n^2 = (|y[n]|^2 - 1)^2$$

where the desired outputs will have norm 1 since we assume amplitudes of 1. This yields a filter weight update of the form shown below.

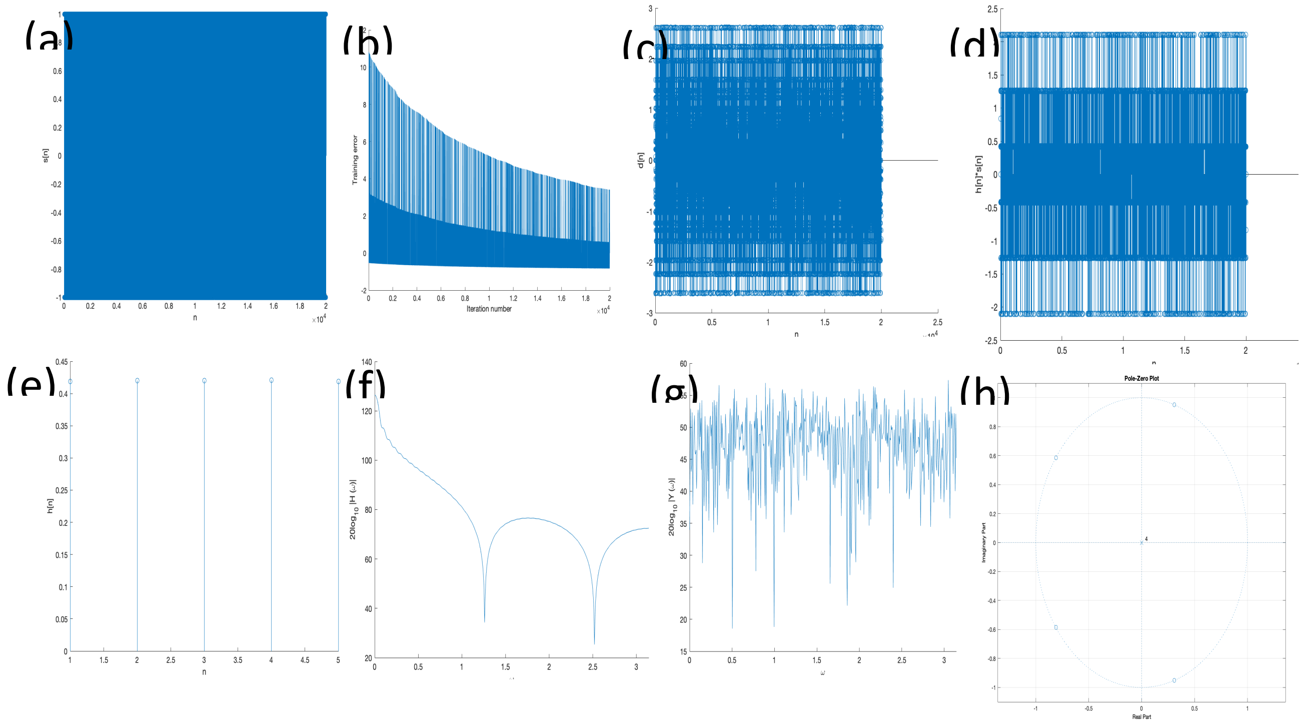
$$g_{n+1} = g_n - \mu(|y[n]|^2 - 1)^2 y[n] x_n$$

with g_n as the filter weights at the nth time step, μ as the step size, x_n as the input, and $y[n]$ as the filtered output.

Algorithm 3: Adaptive Blind Equalization

```
$N \leftarrow$ size of input signal;
$h_0 \leftarrow$ initialize filter weights;
$\mu \leftarrow$ initialize step size (learning rate) ;
$M$ = length($h_0$) ;
$s \leftarrow$ initialize input sequence of size $N$ as random binary sequence of $\pm 1$ ;
$d \leftarrow h_0 * s + \epsilon(SNR)$ initializes test output sequence with added Gaussian noise
  with SNR ;
$weights = zeros(N, M)$ ;
Initializes weight matrix to 0
for $i$=$M+1$:$N-M$ do
    $x = s[i-M : i-1]$ ;
    $y = weights[i] * x$ ;
    $err[i] \leftarrow abs(y)^2 - 1$ ;
    $weights[i+1] \leftarrow weights[i] - \mu err[i] x^T y$ ;
end
return $weights, err, d, s$
```

We implemented this for the same inputs used in section 2 and these results are plotted in Figure 11. Figure 11(a) shows the input signal (1,000 time steps), Figure 11(b) shows the error over the training period indicating convergence behavior with $\mu = 0.0001$, Figure 11(c) shows the desired output, Figure 11(d) shows the output of the LTI system $y[n]$, Figure 11(e) shows the impulse response of the converged filter $h[n]$ with corresponding frequency response in Figure 11(f). Figure 11(g) and 11(h) show the frequency response of the output of the LTI system and pole-zero plot.



(a)
s[n]
n
×10^4
(b)
Training error
Iteration number
×10^4
(c)
d[n]
n
×10^4
(d)
h[n]*s[n]
(e)
h[n]
n
(f)
20log10 |H (ω)|
(g)
20log10 |Y (ω)|
ω
(h)
Pole-Zero Plot
Imaginary Part
Real Part
4

Figure 11

Appendix: Matlab Code Snippits

Part A

```
%% Implementation of adaptive notch filter
function  [a, y] =  adaptive_notch(x, r, mu,a0)
   N = length(x);
   a = zeros(N+4,1);
   a(3) = a0;
   e = zeros(N+4,1);
   y = zeros(N+4,1);
   for n=3:N
       e(n) = x(n) + a(n)*x(n-1) + x(n-2);
       y(n) = e(n) - r*a(n)*y(n-1)-(r^2)*y(n-2);
       if a(n) < 2 && a(n) >= -2
           a(n+1) = a(n) - mu*y(n)*x(n-1);
       else
           a(n+1) = 0;
       end
   end
end

% Project 2, part A
% Part 1
n = linspace(-50,50,100);
x_1 = sin((pi/4)*n) +sin((pi/2)*n) + 20*sin((pi)*n); % first input signal
x_2 = sin((pi)*n);  % second input signal
mu = [0.001, 0.01, 0.1];
r = [0.85, 0.9, 0.97];
a0 = -1.2;

[a_0, y_0] = adaptive_notch(x_2, r(1), mu(1),a0);  %% mu = 0.001, r = 0.85
[a_1, y_1] = adaptive_notch(x_2, r(2), mu(1),a0);  %% mu = 0.001, r = 0.9
[a_2, y_2] = adaptive_notch(x_2, r(3), mu(1),a0);  %% mu = 0.001, r = 0.97

[a_3, y_3] = adaptive_notch(x_2, r(1), mu(1),a0);  %% mu = 0.001, r = 0.85
[a_4, y_4] = adaptive_notch(x_2, r(1), mu(2),a0);  %% mu = 0.01, r = 0.85
[a_5, y_5] = adaptive_notch(x_2, r(1), mu(3),a0);  %% mu = 0.1, r = 0.85


%Plot Convergence of a with respect to r
h1 = figure(1);
ax1 = axes('Parent', h1);
hold on
plot(a_0(3:101))
plot(a_1(3:101))
plot(a_2(3:101))
legend("r = 0.85", "r=0.9", "r=0.97")
xlabel('n');
ylabel('a');

%Plot Convergence of a with respect to mu
h2 = figure(2);
ax2 = axes('Parent', h2);
hold on
plot(a_3(3:101))
plot(a_4(3:101))
plot(a_5(3:101))
legend("\mu = 0.001", "\mu=0.01", "\mu=0.1")
xlabel('n');
ylabel('a');

%Plot of frequency response
a = -1.149; r=0.97;
omeg = linspace(0,2*pi,1000);
numer = [1, a, 1];
denom = [1, r*a, r^2];
n = 1000;
[h,w] = freqz(numer,denom,n);


h1 = figure(1);
ax1 = axes('Parent', h1);
hold on
plot(omeg,abs(h))
xlabel('\omega');
ylabel('|H(\omega)|');

%Plot of impulse response

% Test signal 1 impulse response
h1 = figure(1);
ax1 = axes('Parent', h1);
hold on
stem(x_2)
xlabel('n');
ylabel('x_{test}^1_[n]');

% Test signal 2 impulse response
h2 = figure(2);
```

```
ax2 = axes('Parent', h2);
hold on
stem(x_1)
xlabel('n');
ylabel('x_{test}^2 [n]');


% Filter applied to test signal 1  impulse response
y_1 = filter(numer,denom, x_2);
h1 = figure(1);
ax1 = axes('Parent', h1);
hold on
stem(y_1)
xlabel('n');
ylabel('y_{test}^1 [n]');

% Filter applied to test signal 2  impulse response
y_2 = filter(numer,denom, x_1);
h1 = figure(1);
ax1 = axes('Parent', h1);
hold on
stem(y_2)
xlabel('n');
ylabel('y_{test}^2 [n]');

%Part 2: moving interference
n = linspace(-50,50,100);
phi = 2*n.^2;
x_in = cos(2*pi*phi);

% Plot Input signal
h1 = figure(1);
ax1 = axes('Parent', h1);
hold on
stem(x_in)
xlabel('n');
ylabel('x [n]');

% Apply adaptive notch
a0 = 0;
r0 = 0.9;
mu0 = 0.001;
[a_0, y_0] = adaptive_notch(x_in, r0, mu0,a0);

h1 = figure(1);
```

```
ax1 = axes('Parent', h1);
hold on
plot(a_0(3:101))
xlabel('n');
ylabel('a');

h1 = figure(1);
ax1 = axes('Parent', h1);
hold on
plot(a_0(3:101))
xlabel('n');
ylabel('a');

a = -0.0045; r=0.9;
omeg = linspace(0,2*pi,1000);
numer = [1, a, 1];
denom = [1, r*a, r^2];
n = 1000;
[h,w] = freqz(numer,denom,n);


h1 = figure(1);
ax1 = axes('Parent', h1);
hold on
plot(omeg,abs(h))
xlabel('\omega');
ylabel('|H(\omega)|');

y = filter(numer, denom, x_in);
h1 = figure(1);
ax1 = axes('Parent', h1);
hold on
stem(y)
xlabel('n');
ylabel('y[n]');
```

Part B

```
%% Implementation of LMS adaptive filter
function [weights, err, d, s] = lms_adapt(N, h0, mu, SNR)
   M = length(h0);
    s = randi([0 1], N, 1);
    s(s == 1) = 1;
    s(s == 0) = -1;
    y = conv(h0,s);
    w = wgn(N+M-1,1,SNR);
    d = y + w;
    err = zeros(N,1);
    weights = zeros(N,M);

    for i=M+1:N-M
        x = s(i-M:i-1);
        err(i) = d(i) - (weights(i,:)*x);
        weights(i+1,:) = weights(i,:) + 2*mu * err(i) * x';
    end
end

% Project 2, part B

% Dependence on filter-order
SNR = -30;
mu = 0.01;
N = 1000;
M = [5,10,20,30];
h_0 = rand(M(1),1);
h_1 = rand(M(2),1);
h_2 = rand(M(3),1);
h_3 = rand(M(4),1);

[weight_0, err_0, d_0, s_0] = lms_adapt(N, h_0, mu, SNR);
[weight_1, err_1, d_1, s_1] = lms_adapt(N, h_1, mu, SNR);
[weight_2, err_2, d_2, s_2] = lms_adapt(N, h_2, mu, SNR);
[weight_3, err_3, d_3, s_3] = lms_adapt(N, h_3, mu, SNR);

h_0 = weight_0(N-M(1)-1,:);
h_1 = weight_1(N-M(2)-1,:);
h_2 = weight_2(N-M(3)-1,:);
h_3 = weight_3(N-M(4)-1,:);

%Plot of error with respect to iteration number
h1 = figure(1);
ax1 = axes('Parent', h1);
hold on
plot(sqrt(err_0.*err_0))
plot(sqrt(err_1.*err_1))
plot(sqrt(err_2.*err_2))
plot(sqrt(err_3.*err_3))
xlabel("Iteration number")
ylabel("Training error")
legend("M = 5", "M = 10", "M = 20", "M = 30")
```

```
% Plot of impulse response
h2 = figure(2);
ax2 = axes('Parent', h2);
hold on
stem(h_0)
stem(h_1)
stem(h_2)
stem(h_3)
xlabel("n")
ylabel("h[n]")
legend("M = 5", "M = 10", "M = 20", "M = 30")

% Plot of input
h3 = figure(3);
ax3 = axes('Parent', h3);
hold on
stem(s_0)
xlabel("n")
ylabel("s[n]")

% Plot of output from channel
h4 = figure(4);
ax4 = axes('Parent', h4);
hold on
stem(d_0)
stem(d_1)
stem(d_2)
stem(d_3)
xlabel("n")
ylabel("d[n]")
legend("M = 5", "M = 10", "M = 20", "M = 30")

% Plot of output from convolved signal
h5 = figure(5);
ax5 = axes('Parent', h5);
hold on
stem(conv(h_0, s_0))
stem(conv(h_1, s_1))
stem(conv(h_2, s_2))
stem(conv(h_3, s_3))
xlabel("n")
ylabel("h[n]*s[n]")
legend("M = 5", "M = 10", "M = 20", "M = 30")
```

```
% Plot of frequency response of filter
unit_input = linspace(N,1);
omeg = linspace(0,pi,500);
H0 = fft(conv(h_0,unit_input),1000);
H1 = fft(conv(h_1,unit_input),1000);
H2 = fft(conv(h_2,unit_input),1000);
H3 = fft(conv(h_3,unit_input),1000);

h5 = figure(5);
ax5 = axes('Parent', h5);
hold on
xlim([0 pi])
plot(omeg,20*log10(abs(H0(1:500))))
plot(omeg,20*log10(abs(H1(1:500))))
plot(omeg,20*log10(abs(H2(1:500))))
plot(omeg,20*log10(abs(H3(1:500))))
xlabel("\omega")
ylabel("20log_{10} |H (\omega)|")
legend("M = 5", "M = 10", "M = 20", "M = 30")


% LTI system Impulse response plot
N = 1000;
y_0 = conv(h_0, s_0) + wgn(length(conv(h_0, s_0)),1,SNR);
y_1 = conv(h_1, s_1) + wgn(length(conv(h_1, s_1)),1,SNR);
y_2 = conv(h_2, s_2) + wgn(length(conv(h_2, s_2)),1,SNR);
y_3 = conv(h_3, s_3) + wgn(length(conv(h_3, s_3)),1,SNR);

h6 = figure(6);
ax6 = axes('Parent', h6);
hold on
stem(y_0)
stem(y_1)
stem(y_2)
stem(y_3)
xlabel("n")
ylabel("y[n]")
legend("M = 5", "M = 10", "M = 20", "M = 30")


Y0 = fft(y_0);
Y1 = fft(y_1);
Y2 = fft(y_2);
Y3 = fft(y_3);
```

```
% LTI system frequency response plot
h7 = figure(7);
ax7 = axes('Parent', h7);
hold on
xlim([0 pi])
plot(omeg,20*log10(abs(Y0(1:500))))
plot(omeg,20*log10(abs(Y1(1:500))))
plot(omeg,20*log10(abs(Y2(1:500))))
plot(omeg,20*log10(abs(Y3(1:500))))
xlabel("\omega")
ylabel("20log_{10} |Y (\omega)|")
legend("M = 5", "M = 10", "M = 20", "M = 30")

% Pole-Zero frequency plot
%%h_0
h8 = figure(8);
fvtool(h_0,1,'Analysis','polezero')

%%h_1
h9 = figure(9);
fvtool(h_1,1,'Analysis','polezero')

h10 = figure(10);
%%h_2
fvtool(h_2,1,'Analysis','polezero')

%%h_3
h11 = figure(11);
fvtool(h_3,1,'Analysis','polezero')

% Dependence on step-size
SNR = -30;
mu = [0.001, 0.01] ;
N = 1000;
M = 5;
h0 = rand(M,1);

[weight_0, err_0, d_0, s_0] = lms_adapt(N, h0, mu(1), SNR);
[weight_1, err_1, d_1, s_1] = lms_adapt(N, h0, mu(2), SNR);


h_0 = weight_0(N-M-1,:);
h_1 = weight_1(N-M-1,:);

% Plot of error
```

```
h1 = figure(1);
ax1 = axes('Parent', h1);
hold on
plot(sqrt(err_0.*err_0))
plot(sqrt(err_1.*err_1))
xlabel("Iteration number")
ylabel("Training error")
legend("\mu = 0.001", "\mu = 0.01")

% Plot of impulse response
h2 = figure(2);
ax2 = axes('Parent', h2);
hold on
stem(h_0)
stem(h_1)
xlabel("n")
ylabel("h[n]")
legend("\mu = 0.001", "\mu = 0.01")

% Plot of input
h3 = figure(3);
ax3 = axes('Parent', h3);
hold on
stem(s_0)
xlabel("n")
ylabel("s[n]")

% Plot of output from channel
h4 = figure(4);
ax4 = axes('Parent', h4);
hold on
stem(d_0)
stem(d_1)
xlabel("n")
ylabel("d[n]")
legend("\mu = 0.001", "\mu = 0.01")

% Plot of output from convolved signal
h5 = figure(5);
ax5 = axes('Parent', h5);
hold on
stem(conv(h_0, s_0))
stem(conv(h_1, s_1))
xlabel("n")
ylabel("h[n]*s[n]")
```

```
legend("\mu = 0.001", "\mu = 0.01")

% Plot of frequency response of filter
unit_input = linspace(N,1);
omeg = linspace(0,pi,500);
H0 = fft(conv(h_0,unit_input),1000);
H1 = fft(conv(h_1,unit_input),1000);

h5 = figure(5);
ax5 = axes('Parent', h5);
hold on
xlim([0 pi])
plot(omeg,20*log10(abs(H0(1:500))))
plot(omeg,20*log10(abs(H1(1:500))))
xlabel("\omega")
ylabel("20log_{10} |H (\omega)|")
legend("\mu = 0.001", "\mu = 0.01")


% LTI system Impulse response plot
N = 1000;
y_0 = conv(h_0, s_0) + wgn(length(conv(h_0, s_0)),1,SNR);
y_1 = conv(h_1, s_1) + wgn(length(conv(h_1, s_1)),1,SNR);


h6 = figure(6);
ax6 = axes('Parent', h6);
hold on
stem(y_0)
stem(y_1)
xlabel("n")
ylabel("y[n]")
legend("\mu = 0.001", "\mu = 0.01")


Y0 = fft(y_0);
Y1 = fft(y_1);
Y2 = fft(y_2);
Y3 = fft(y_3);

% LTI system frequency response plot
h7 = figure(7);
ax7 = axes('Parent', h7);
hold on
xlim([0 pi])
```

```
plot(omeg,20*log10(abs(Y0(1:500))))
plot(omeg,20*log10(abs(Y1(1:500))))
xlabel("\omega")
ylabel("20log_{10} |Y (\omega)|")
legend("\mu = 0.001", "\mu = 0.01")

Pole-Zero frequency plot
%h_0
h8 = figure(8);
fvtool(h_0,1,'Analysis','polezero')

%%h_1
h9 = figure(9);
fvtool(h_1,1,'Analysis','polezero')


% Dependence on SNR
SNR = [-30, -5] ;
mu = 0.01;
N = 1000;
M = 5;
h0 = rand(M,1);

[weight_0, err_0, d_0, s_0] = lms_adapt(N, h0, mu, SNR(1));
[weight_1, err_1, d_1, s_1] = lms_adapt(N, h0, mu, SNR(2));


h_0 = weight_0(N-M-1,:);
h_1 = weight_1(N-M-1,:);

Plot of error
h1 = figure(1);
ax1 = axes('Parent', h1);
hold on
plot(sqrt(err_0.*err_0))
plot(sqrt(err_1.*err_1))
xlabel("Iteration number")
ylabel("Training error")
legend("SNR = -30 dBW", "SNR = -5 dBW")

% Plot of impulse response
h2 = figure(2);
ax2 = axes('Parent', h2);
hold on
stem(h_0)
stem(h_1)
xlabel("n")
ylabel("h[n]")
legend("SNR = -30 dBW", "SNR = -5 dBW")
%
Plot of input
h3 = figure(3);
ax3 = axes('Parent', h3);
hold on
stem(s_0)
xlabel("n")
ylabel("s[n]")

% Plot of output from channel
h4 = figure(4);
ax4 = axes('Parent', h4);
hold on
stem(d_0)
stem(d_1)
xlabel("n")
ylabel("d[n]")
legend("SNR = -30 dBW", "SNR = -5 dBW")

% Plot of output from convolved signal
h5 = figure(5);
ax5 = axes('Parent', h5);
hold on
stem(conv(h_0, s_0))
stem(conv(h_1, s_1))
xlabel("n")
ylabel("h[n]*s[n]")
legend("SNR = -30 dBW", "SNR = -5 dBW")

% Plot of frequency response of filter
unit_input = linspace(N,1);
omeg = linspace(0,pi,500);
H0 = fft(conv(h_0,unit_input),1000);
H1 = fft(conv(h_1,unit_input),1000);

h5 = figure(5);
ax5 = axes('Parent', h5);
hold on
xlim([0 pi])
plot(omeg,20*log10(abs(H0(1:500))))
plot(omeg,20*log10(abs(H1(1:500))))
```

```
xlabel("\omega")
ylabel("20log_{10} |H (\omega)|")
legend("SNR = −30 dBW", "SNR = −5 dBW")


% LTI system Impulse response plot
N = 1000;
y_0 = conv(h_0, s_0) + wgn(length(conv(h_0, s_0)),1,SNR(1));
y_1 = conv(h_1, s_1) + wgn(length(conv(h_1, s_1)),1,SNR(2));


h6 = figure(6);
ax6 = axes('Parent', h6);
hold on
stem(y_0)
stem(y_1)
xlabel("n")
ylabel("y[n]")
legend("SNR = −30 dBW", "SNR = −5 dBW")
%

Y0 = fft(y_0);
Y1 = fft(y_1);


%%LTI system frequency response plot
h7 = figure(7);
ax7 = axes('Parent', h7);
hold on
xlim([0 pi])
plot(omeg,20*log10(abs(Y0(1:500))))
plot(omeg,20*log10(abs(Y1(1:500))))
xlabel("\omega")
ylabel("20log_{10} |Y (\omega)|")
legend("SNR = −30 dBW", "SNR = −5 dBW")


Pole−Zero frequency plot
%%h_0
h8 = figure(8);
fvtool(h_0,1,'Analysis','polezero')

%%h_1
h9 = figure(9);
fvtool(h_1,1,'Analysis','polezero')
```

Part C

```
%% Implementation of adaptive blind equalizaiton filter
function [weights, err, d, s] = adaptive_blind(N, h0, mu, SNR)
   M = length(h0);
    s = randi([0 1], N, 1);
    s(s == 1) = 1;
    s(s == 0) = -1;
    y = conv(h0,s);
    w = wgn(N+M-1,1,SNR);
    d = y + w;
    err = zeros(N,1);
    weights = repmat(h0,N,M);
    disp(size(weights))


    for i=M+1:N-M
        x = s(i-M:i-1);
        y = weights(i,:)*x;
        err(i) = abs(y).^2 - 1;
        weights(i+1,:) = weights(i,:) - mu * err(i)*y* x';
    end
end

%%Project 2, part C

%%Adaptive Blind Equalization
SNR = -50;
mu = 0.00001;
N = 20000;
M = 5;
h0 = rand(M,1);
[weights, err, d, s] = adaptive_blind(N, h0, mu, SNR);

h_0 = weights(N-M-1,:);
%%Plot of error with respect to iteration number
h1 = figure(1);
ax1 = axes('Parent', h1);
hold on
plot(err)
xlabel("Iteration number")
ylabel("Training error")

%%Plot of impulse response
h2 = figure(2);
```

```
ax2 = axes('Parent', h2);
hold on
stem(h_0)
xlabel("n")
ylabel("h[n]")


% Plot of input
h3 = figure(3);
ax3 = axes('Parent', h3);
hold on
stem(s)
xlabel("n")
ylabel("s[n]")


%%Plot of output from channel
h4 = figure(4);
ax4 = axes('Parent', h4);
hold on
stem(d)
xlabel("n")
ylabel("d[n]")

%%Plot of output from convolved signal
h5 = figure(5);
ax5 = axes('Parent', h5);
hold on
stem(conv(h_0, s))
xlabel("n")
ylabel("h[n]*s[n]")

%%Plot of frequency response of filter
unit_input = linspace(N,1);
omeg = linspace(0,pi,500);
H0 = fft(conv(h_0,unit_input),1000);

h5 = figure(5);
ax5 = axes('Parent', h5);
hold on
xlim([0 pi])
plot(omeg,20*log10(abs(H0(1:500))))
xlabel("\omega")
ylabel("20log_{10} |H (\omega)|")
```

```
% LTI system Impulse response plot
y_0 = conv(h_0, s) + wgn(length(conv(h_0, s)),1,SNR);

h6 = figure(6);
ax6 = axes('Parent', h6);
hold on
stem(y_0)
xlabel("n")
ylabel("y[n]")

Y0 = fft(y_0);
%%LTI system frequency response plot
h7 = figure(7);
ax7 = axes('Parent', h7);
hold on
xlim([0 pi])
plot(omeg,20*log10(abs(Y0(1:500))))
xlabel("\omega")
ylabel("20log_{10} |Y (\omega)|")

%h_0
h8 = figure(8);
fvtool(h_0,1,'Analysis','polezero')
```