ELEC531 HW4 Adaptive Channel Equalization

Ender Erkaya

December 11, 2021

Contents

Listings

Adaptive Channel Equalization

```
clearvars
%
% Ender Erkaya
% December 2021
%
```

a) Statistics of the LTI system and FIR MMSE Equalization Function

To implement FIR MMSE Equalizer function of a known stochastic system, first we need to derive statistics $R_{\mathbf{y}x_{i-d}}$ (cross correlation matrix btw observations $\mathbf{y}$ and the desired signal x_{i-d}) and $R_{\mathbf{yy}}$ (autocorrelation matrix of observations $\mathbf{y}$). The system is a known linear system in the form $y_i = h^T x$ and $\mathbf{y} = \mathbf{Hx} + \mathbf{v}$ where $\mathbf{H}$ is covariance matrix of the channel impulse response h and $\mathbf{v}$ is iid white noise samples vector. The cross covariance matrix is:

$$R_{\mathbf{y}x_{i-d}} = E(\mathbf{y}x^*_{i-d}) = E((\mathbf{Hx} + \mathbf{v})x^*_{i-d})$$

$$= \mathbf{H}R_{\mathbf{x}x_{i-d}} = \mathbf{H}\sigma^2_{\mathbf{x}_i} e_{d+1}$$

$$R_{\mathbf{y}x_{i-d}} = \sigma^2_{\mathbf{x}}\mathbf{H}_{:,d+1}$$

where $\mathbf{H}_{:,d+1} = [h_d \; h_{d-1} \; ...h_0 \; zeros(1 \times (N - d - 1)]^T$ The covariance matrix of the observations sequence:

$$R_{\mathbf{yy}} = E(\mathbf{yy^*})$$

$$= E((\mathbf{Hx}+\mathbf{v})(\mathbf{Hx}+\mathbf{v})^*)$$
$$= E((\mathbf{Hx}+\mathbf{v})(\mathbf{x^*H^*}+\mathbf{v^*})$$
$$= \mathbf{H}E(\mathbf{xx^*})\mathbf{H^*}+\mathbf{v}\ \mathbf{v^*}$$

since the signal **x** and the noise **v** are uncorrelated

$$\mathbf{H}R_{\mathbf{xx}}\mathbf{H^*}+R_{\mathbf{vv}}$$

$$R_{\mathbf{yy}} = \sigma^2_{\mathbf{x}_i}\mathbf{H}\ \mathbf{H^*}+\sigma^2_{\mathbf{v}_i}\mathbf{I}$$

Putting the values $\sigma^2_{\mathbf{x}_i} = 1$ and $\sigma^2_{\mathbf{v}_i} = 10^{-4}$ into the equations:

$$R_{\mathbf{y}x_{i-d}} = \mathbf{H}_{:,d+1}$$

$$R_{\mathbf{yy}} = \mathbf{H}\ \mathbf{H^*}+10^{-4}\mathbf{I}$$

b) FIR MMSE Equalizer for delays d=0,...,8

```
h = [1 1.5]; % channel impulse response
N = 10; % equalizer length
d = 0:8; % delays vector
r = 1e-4; % noise variance

% Find the mmse equalizer filter and mmse
[w, mmse] = findmmsefireq(h,r,N,d);
wopt = w;
figure
semilogy(d, mmse, 'k');
grid on;
title('MMSE vs Delay', 'Interpreter', 'latex');
xlabel('Delay', 'Interpreter', 'latex');
ylabel('mmse', 'Interpreter', 'latex');
```

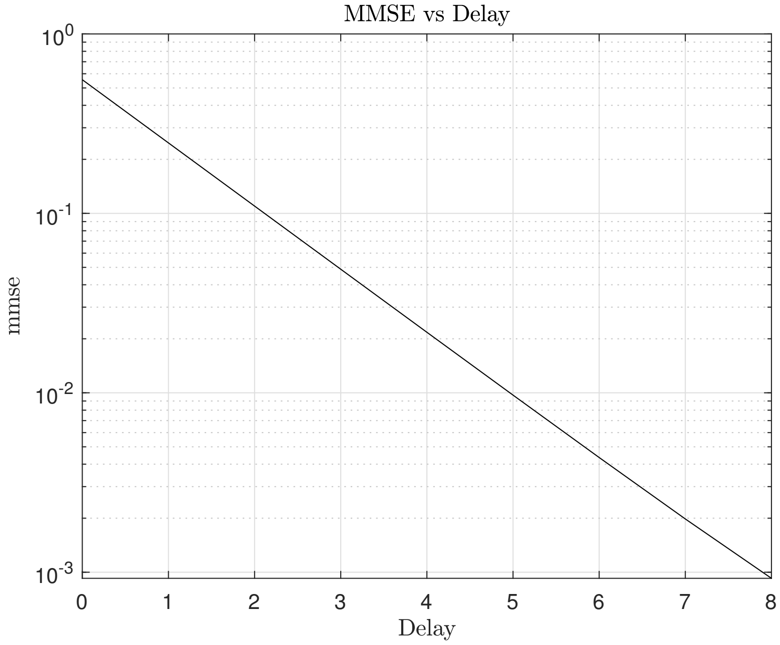

Figure 1:

log(mmse) function decreases almost linearly by delay parameter d for $d < N$, MMSE ˜ α^d, for $SNR = 10 * log10(1e4) = 40dB$. As SNR decreases, the almost linear behavior of log(mmse) changes. Optimum delay parameter depends on SNR, $10 * log10(1/r)$. For $SNR = 40dB$, optimum delay is 8

c) Equalization of BPSK-2QAM sequence on awgn channel

Settings

```
d            = 8;
mu           = 0.03;

sq_error1  = simulateLMSerrorcurve(h,r,N,d,mu);
sigma_e1   = std(sq_error1, 1);
mean_e1    = mean(sq_error1, 1);

% Display mean square error convergence curves
figure
semilogy(1:length(mean_e1), mean_e1 + sigma_e1, '--k', 'LineWidth',
    1.5);
hold on
semilogy(1:length(mean_e1), mean_e1, 'r', 'LineWidth', 2);
grid on
xlabel('time','Interpreter','latex');
ylabel('averaged squared error ($(x_{k-d}-\hat{x_{k-d}})^2$)','
    Interpreter','latex');
title('Average square error convergence curve','Interpreter','latex');
legend('Mean + Sigma','Mean')
```

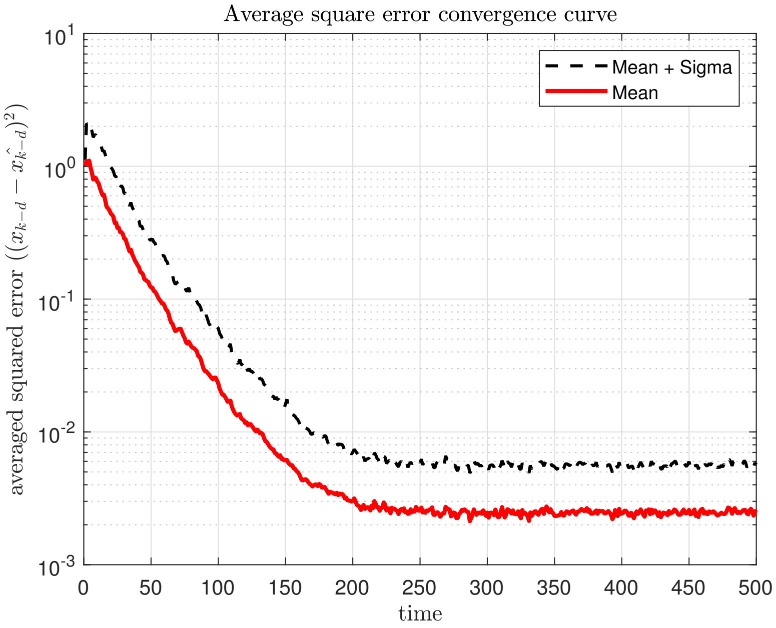

Figure 2:

LMS algorithm learns the deconvolution filter by approximating a gradient descent step for each observation. Since it corresponds to an approximate gradient descent method, the convergence behavior is slow, ie it takes a few hundred steps to learn the linear system. It is an online algorithm, which updates the equalization filter at each received observation. Due to its online behavior, the update cost per iteration is low.

d) $\mu = 0.03$ vs $\mu = 0.02$ mse convergence curves

Settings

```
mu         = 0.02;

sq_error2  = simulateLMSerrorcurve(h,r,N,d,mu);
% sigma_e2 = std(sq_error2,1);
mean_e2    = mean(sq_error2,1);

% Display mean square error convergence curves
figure
plot(1:length(mean_e2), 10*log10(mean_e2), 'b', 'LineWidth', 1.5);
hold on
plot(1:length(mean_e1), 10*log10(mean_e1), 'r', 'LineWidth', 1.5);
hold on
yline(10*log10(mmse(d+1)), 'k', 'LineWidth', 1.5)
grid on
xlabel('time','Interpreter','latex');
ylabel('mean squared error (dB)','Interpreter','latex');
title('MSE convergence curves','Interpreter','latex');
legend('$\mu=0.02$','$\mu=0.03$','mmse(dB)','Interpreter','latex');
```

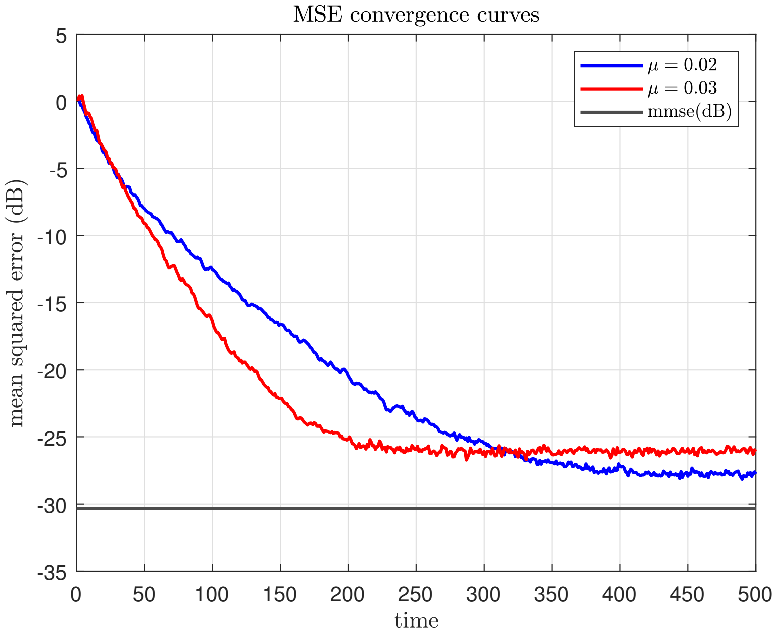

Figure 3:

The convergence of $\mu = 0.03$ is faster than that of $\mu = 0.02$. mmse $\mu = 0.03$ converges at around 250 steps. $\mu = 0.02$ converges at around 450 steps. However, $\mu = 0.02$ achieves better mse performance, ~2dB distance to optimal mse. $\mu = 0.03$ achieves worse mse performance, less than ~4dB from mmse. Excess mmse levels are about $2.3$dB and $4.3$dB FOR $\mu = 0.02$ and $\mu = 0.03$ accordingly. This may due to that although higher step size converges faster, it may be due to that the step size $\mu = 0.03$ would be slightly too high for the curvature as the the equalizer w approaches to, around its optimal value.

e) Behavior of $\mu = 0.5$

Settings

```
mu       = 0.5;

sq_error3  = simulateLMSerrorcurve(h,r,N,d,mu);
mean_e3    = mean(sq_error3,1);

% Display mean square error convergence curves
figure
plot(1:length(mean_e3), 10*log10(mean_e3), 'm', 'LineWidth', 2);
hold on
yline(10*log10(mmse(d+1)), 'k', 'LineWidth', 1.5)
grid on
xlabel('time','Interpreter','latex');
ylabel('Mean squared error ($(x_{k-d}-\hat{x_{k-d}})^2$)','Interpreter'
    ,'latex');
title('MSE convergence curve','Interpreter','latex');
legend('MSE curve','mmse','Interpreter','latex');
```

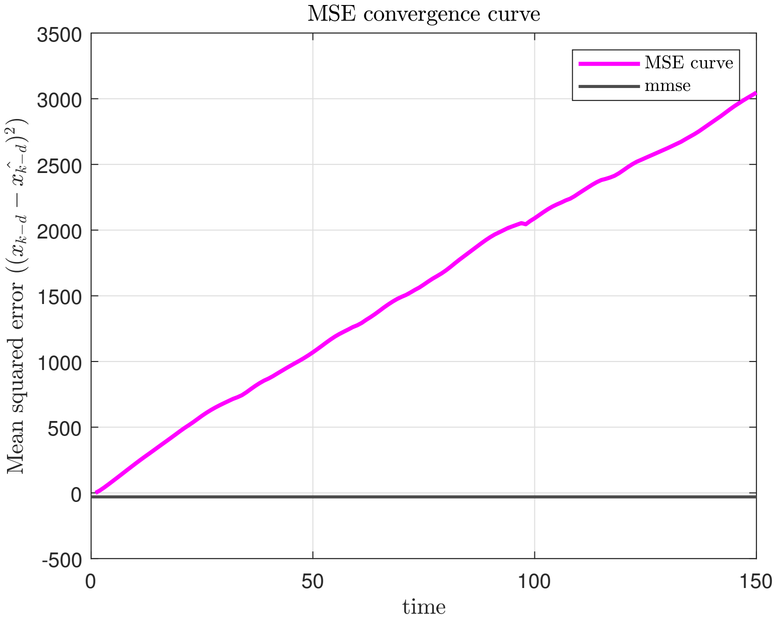

Figure 4:

The reason behind the divergence behavior should be investigated behind the condition and eigenvalues of the curvature, hessian matrix of the objective function.

$$\nabla^2(J(w)) = R_{\mathbf{yy}}$$

Hence, the spectral coefficients of $R_{\mathbf{yy}}$ is analyzed as:

```
hn    = horzcat(reshape(h, [1 length(h)]), zeros(1,N-1));
vn    = vertcat(h(1), zeros(N-1,1));
H     = toeplitz(vn, hn); % convolution matrix N * (N+L-1)
Ryy   = H * H' + r * eye(N);
lambdas = sort(eig(Ryy),'descend');
L     = lambdas(1);
m     = lambdas(end);
mu_max = 2/L;
fprintf('The eigenvalues of the hessian matrix are: [');
fprintf('%.3g ', lambdas);
fprintf(']\n');
fprintf('Lipschitz constant is %.2f \n', L);
fprintf('Strong convexity parameter is %.2f \n ', m);
fprintf('The condition number is %.2f \n', L/m);
```

```
The eigenvalues of the hessian matrix are: [6.13 5.77 5.21 4.5 3.68 2.82 2 1.29 0.726 0.372 ]
Lipschitz constant is 6.13
Strong convexity parameter is 0.37
 The condition number is 16.49
```

As seen from the spectrum of the hessian matrix, the step size for an gradient descent method must be less than maximum eigenvalue λ_{max} of $R_{\mathbf{yy}}$. For the guaranteed convergence, following condition must hold for the exact gradient descent algorithm:

$$\mu \leq 2/L$$

Since LMS algorithm is an approximated online gradient descent, the optimum value for exact gradient descent is not observed. In implemented LMS scheme, different step size values are experimented and below graph is obtained in order to decide the maximum step size value.

LMS for different step sizes on the convergence boundary

```
mus = [0.04:0.001:0.045];

figure
for i = 1:length(mus)
    mu = mus(i);
    sq_error4 = simulateLMSerrorcurve(h,r,N,d,mu);
    mean_e4    = mean(sq_error4,1);
    hold on
    plot(1:length(mean_e4), 10*log10(mean_e4));
    grid on
end
legend('$\mu=0.04$','$\mu=0.041$','$\mu=0.042$','$\mu=0.043$','$\mu
   =0.044$','$\mu=0.045$','Interpreter','latex');
```

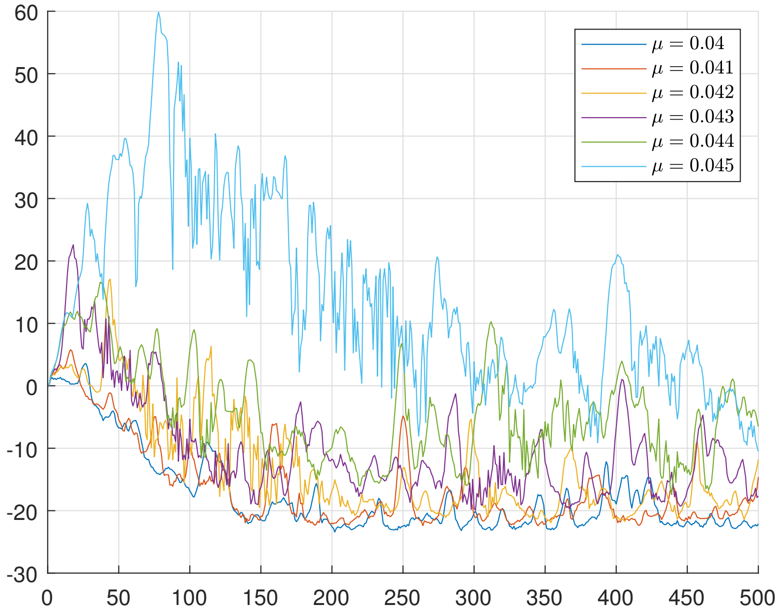

Figure 5:

According to the obtained figure maximum step size is around

$$\mu = 0.42$$

Gear Shifted LMS

Due to the difference in convergence characteristics of $\mu = 0.03$ and $\mu = 0.02$, a gear shifted LMS algorithm is suggested to utilize their convergence characteristics. The algorithm is a version of LMS where step size is reduced to $\mu = 0.02$ after some phase. Below is the obtained convergence plot when change is applied at $k = 150$. After k == 150 step size changes to 0.02 $\mu = 0.03$ for $k < 150$, $\mu = 0.02$ for $k \geq 150$

```
sq_errorgs = simulategearshiftedLMSerrorcurve(h,r,N,d);
mean_egs  = mean(sq_errorgs,1);
```

Continuous Gear Shifted LMS

Further develop the convergence a continuous gear shifted scheme is simulated below. The scheme updates the step size continuous as

$$\mu_k = \frac{\mu_0}{1 + \frac{k}{200}}$$

The initial step size is adjusted as

$$\mu_0 = 0.4$$

around experimentally found maximum $ \mu_{max}$ for a convergent LMS algorithm.

```
mu0 = 0.04;
sq_errorcgs = simulatecontgearshiftedLMSerrorcurve(h,r,N,d,mu0);
mean_ecgs  = mean(sq_errorcgs,1);

figure
plot(1:length(mean_ecgs), 10*log10(mean_ecgs), 'c', 'LineWidth', 1.5);
hold on
plot(1:length(mean_egs), 10*log10(mean_egs), 'm', 'LineWidth', 1.5);
hold on
plot(1:length(mean_e2), 10*log10(mean_e2), 'b', 'LineWidth', 1.5);
hold on
plot(1:length(mean_e1), 10*log10(mean_e1), 'r', 'LineWidth', 1.5);
hold on
yline(10*log10(mmse(d+1)), 'k', 'LineWidth', 1.5)
grid on
xlabel('time','Interpreter','latex');
ylabel('mean squared error (dB)','Interpreter','latex');
title('MSE convergence curves','Interpreter','latex');
legend('continuous gear shifted($\mu_k= 0.04(1/(1+k/200))$)','gear-
   shifted,$k=150$','$\mu=0.02$','$\mu=0.03$','mmse(dB)','Interpreter',
   'latex');
```

The convergence behavior is improved by both a gear shifted LMS scheme and a continuously gear shifted LMS. Both modified LMS schemes outperforms constant step LMS versions simulated before. The suggested continuous gear shifted LMS version performs best in all in case of both convergence speed and final mse performance.

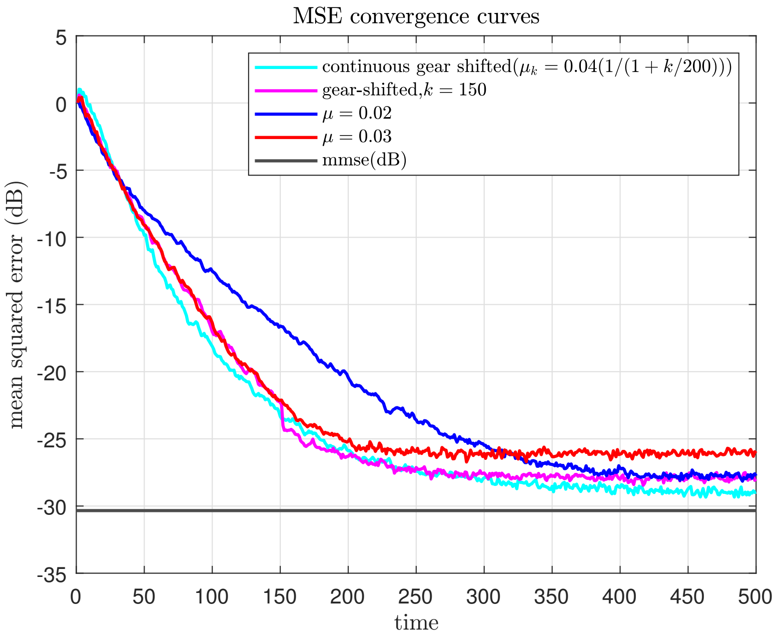

Figure 6:

Transform LMS

Prewhitening applied assumed R_yy is known a priori in order to observe the effect of the prewhitening.

```
[V, D] = eig(Ryy);
Wpre = D^(-1/2)* V';
mu   = 0.1;
sq_errorpw = simulateLMSerrorcurvePW(h,r, N, d, mu, Wpre);
mean_epw   = mean(sq_errorpw,1);

figure
plot(1:length(mean_ecgs), 10*log10(mean_ecgs), 'b', 'LineWidth', 1.5);
hold on
plot(1:length(mean_epw), 10*log10(mean_epw), 'r', 'LineWidth', 1.5);
hold on
yline(10*log10(mmse(d+1)), 'k', 'LineWidth', 1.5)
grid on
xlabel('time','Interpreter','latex');
ylabel('mean squared error (dB)','Interpreter','latex');
title('MSE convergence curves','Interpreter','latex');
legend('continuous gear shifted($\mu_k=0.04(1/(1+k/200))$)','Transform
   LMS with $\mu_k=0.1 (1/(1+k/100))$ ','Interpreter','latex');
```

Having assumed R_{yy} is known a priori, a prewhitening scheme is applied together with continuously gear shifted step size. The combination of Transform LMS with continously gear shifted LMS reveals better result. The learning rate is chosen as:

$$\mu_k = \frac{\mu_0}{1 + \frac{k}{100}}$$

where μ_0 is adjusted as 0.1 The used prewhitener for Transform LMS is given below:

$$W_{pre} = D^{(} - 1/2) * V^*$$

where

$$R_{yy} = V * D * V^*$$

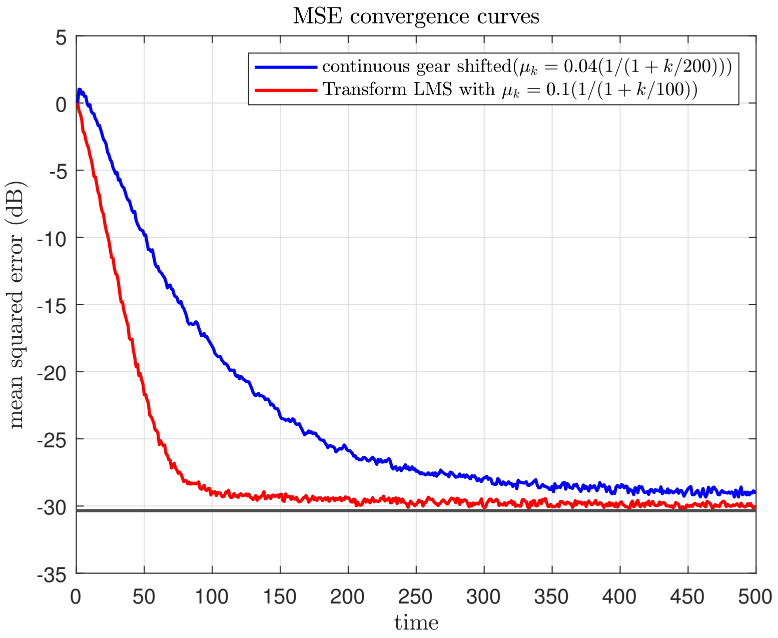

Figure 7:

f) RLS Algorithm

Settings

```
lambda    = 0.99; % forgetting factor
num_sim   = 1000; %number of process realizations
samples_len = 500; %number of time steps
% d = 8;
delta = 1;

% Initializations

num_samples   = samples_len + N-1;
error_RLS     = zeros(num_sim, samples_len);
sq_error_RLS  = zeros(num_sim, samples_len);

for s = 1:num_sim
    % generate binary data
    data = randi([0 1], num_samples+length(h)-1, 1);
    % bpsk constelltions
    x    = 2 * data - 1;
    % white gaussian noise with variance r
    v    = sqrt(r) * randn(length(x)+length(h)-1, 1);
    % v = wgn(num_samples + length(h)-1,1, 10*log10(1e-4));
    % generate channel output
    y    = conv(x, h) + v;
    xc   = x(length(h):end); % constellations, info signal
    yc   = y(length(h):end-length(h)+1); % observations, channel output
    wRLS = 0.01 * randn(N,1); % initial equalizer random guess
    PK   = delta * eye(N); % initialization of PK := (1/K) * Ryinv(K)
%     Ryyinv = inv(Ryy); % to compare

    % RLS Algorithm
    for K = d+1:samples_len+d
        yk = [yc(K:-1:max(1,K-N+1)) ; zeros(N-K,1)]; % current
```

```
           observation vector
        xk = xc(K-d);
        error_RLS(s,K-d)   = xk - (wRLS' * yk); % current error

        % Update equalizer
        temp = (lambda) + (yk' * PK * yk);
        wRLS = wRLS + (error_RLS(s,K-d)/temp) * PK * yk;
        %     deltaw(:,K) = norm(wopt-wRMS(:,K+1));

        % PK Update
        PKold = PK;
        PK    = (1/lambda)*(PK - ((PK* yk * yk' *PK)/temp));
        %     PKC(K)=norm(PK-(1/K)*inv(Ryy));
    end
    sq_error_RLS(s,:)=error_RLS(s,:).^2;
end

mean_eRLS  = mean(sq_error_RLS ,1);

figure
plot(1:length(mean_ecgs), 10*log10(mean_ecgs), 'r', 'LineWidth', 1.5);
hold on
plot(1:length(mean_eRLS), 10*log10(mean_eRLS), 'm', 'LineWidth', 1.5);
hold on
yline(10*log10(mmse(d+1)), 'k', 'LineWidth', 1.5)
grid on
xlabel('time','Interpreter','latex');
ylabel('mean squared error (dB)','Interpreter','latex');
title('MSE convergence curves($d=8)$','Interpreter','latex');
legend('continuous gear shifted($\mu_k=0.04(1/(1+k/200))$)','RLS','
   Interpreter','latex');
```

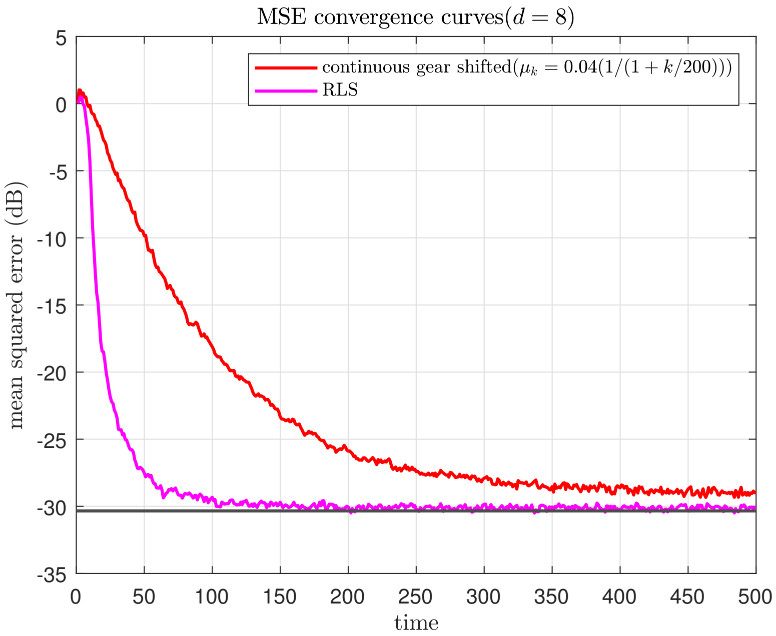

Figure 8:

Normalized version of RLS algorithm is implemented above. Delay parameter is chosen same with before, $d = 8$ to compare the performances. It is observed that the equalization performance is dependent on the delay parameter as observed before on mmse vs delay plot. The convergence behavior resembles to Newton-like second order algorithms as expected. Since, RLS algorithm behaves as second order algorithm by recursively estimating the curvature, ie autocorrelation matrix R_{yy}, and it applies curvature correction by R_{yy}^{-1} with recursive autocorrelation matrix estimation. One of the observed unwanted behavior in the above code is that it highly depends on the initialization of $P(k)$ which was chosen as:

$$P(0) = \delta * I_N$$

$\delta = 1$ reveals the above performance, setting $\delta = 0.01$ reduces the convergence speed significantly. In order to observe the effect of the forgetting factor λ, RLS algorithm performance is observed below with differen choices of λ

Forgetting Factor λ

```
lambdas = [1, 1-0.1.^[4:-1:1], 0.8, 0.7, 0.6, 0.5];
mse_RLS_lambdas = zeros(length(lambdas),samples_len);

for it = 1:length(lambdas)
    lambda = lambdas(it);
    mse_RLS_lambdas(it,:) = simulateRLSerrorcurve(h,r,N,d,lambda);
end

figure
yline(10*log10(mmse(d+1)), 'k', 'LineWidth', 1.2)
for it = 1:length(lambdas)
hold on
plot(1:samples_len,10*log10(mse_RLS_lambdas(it,:)), 'LineWidth', 1.2);
hold on
grid on
end
hold on
yline(10*log10(mmse(d+1)), 'k', 'LineWidth', 1.2)
legend('$\lambda=1$','$\lambda=1-1e-4$','$\lambda=0.999$','$\lambda
    =0.99$','$\lambda=0.9$',...
     '$\lambda=0.8$','$\lambda=0.7$','$\lambda=0.6$','$\lambda=0.5$','
        Interpreter','latex');
title('RLS vs $\lambda$','Interpreter','latex');
xlabel('time','Interpreter','latex');
ylabel('mean squared error (dB)','Interpreter','latex');
title('MSE convergence curves($d=8)$','Interpreter','latex');
```

The performance of the above algorithm at this settings reduces significantly after $\lambda < 0.9$. The simulated system is highly stationary, high performance at large λ values is expected. As lowering λ creates robustness over nonstationarity, it reduces performance for highly stationary data. Hence, it should be adjusted properly.

Appendix-1 findmmsefireq.m

```
function [w, mmse] = findmmsefireq(h,r,N,d)
% MATLAB function that calculates the MMSE FIR equalizer
% h : Impulse response of the channel to be equalized
% r : Variance of the iid noise
% N : Length of the FIR equalizer
% d : Equalization delay
% w : Equalizer coefficients as output
% mmse: Minimum Mean Square Error as output
% This function assumes that transmission sequence has variance 1.
% Initializations
hn    = horzcat(reshape(h, [1 length(h)]), zeros(1,N-1));
vn    = vertcat(h(1), zeros(N-1,1));
H     = toeplitz(vn, hn); % convolution matrix N * (N+L-1)
Ryy   = H * H' + r * eye(N);
[L,U]= lu(Ryy);
w     = zeros(N,length(d)); % equalizer
mmse = zeros(length(d),1); % mmse values

for k = 1:length(d)
    delay = d(k);

    % Normal Equations
    Ryx = H(:, delay + 1);

    % Solve FIR MMSE Equalizer
    w(:,k)  = U \ (L \ Ryx);
    mmse(k) = 1 - Ryx' * w(:,k); % sigma2x - Ryx' * Ry^(-1) * Ryx
end
end
```

Appendix-2 simulateLMSerrorcurve.m

```
function [sq_error] = simulateLMSerrorcurve(h,r,N,d,mu)
%
% Ender Erkaya
% December 2021
%
% simulates convergence mse curve for LMS update of a generated bpsk
% constellation in awgn channel
% Inputs:
% channel impulse response h,
% noise variance r,
% equalizer length N,
% equalization delay d,
% LMS step size mu,
% Outputs

% Settings
num_sim   = 1000; %number of process realizations
samples_len = 500; %number of time steps

% Initializations
num_samples = samples_len + N-1;
error     = zeros(num_sim, samples_len);
sq_error = zeros(num_sim, samples_len);

% Simulations of LMS Equalization Updates
for s = 1:num_sim

    % generate binary data
    data = randi([0 1], num_samples+length(h)-1, 1);
    % bpsk constelltions
    x    = 2 * data - 1;
    % white gaussian noise with variance r
    v    = sqrt(r) * randn(length(x)+length(h)-1, 1);
    % v = wgn(num_samples + length(h)-1,1, 10*log10(1e-4));
    % generate channel output
    y     = conv(x, h) + v;
    y     = y(length(h):end-length(h)+1); % observations, channel output
    x     = x(length(h):end); % constellations, info signal
    w_i   = 0.01 * randn(N,1); % initial equalizer random guess
    w_LMS      = zeros(N, samples_len+1); % equalizer
    w_LMS(:,1)= w_i;

    % LMS Updates
    for k = N : num_samples % samples_len = 500
        it = k-N+1;
        error(s,it)    = x(k-d) - w_LMS(:,it)' * y(k:-1:it);
        w_LMS(:,it+1) = w_LMS(:,it) + mu * error(s,it) * y(k:-1:it);
    end
    sq_error(s,:) = error(s,:).^2;
end
end
```

Appendix 3 simulategearshiftedLMSerrorcurve.m

```
function [sq_error] = simulategearshiftedLMSerrorcurve(h,r,N,d)
% Settings
num_sim   = 1000; %number of process realizations
samples_len = 500; %number of time steps

% Initializations
num_samples = samples_len + N-1;
error     = zeros(num_sim, samples_len);
sq_error = zeros(num_sim, samples_len);

% Simulations of LMS Equalization Updates
for s = 1:num_sim

    % generate binary data
    data = randi([0 1], num_samples+length(h)-1, 1);
    % bpsk constelltions
    x     = 2 * data - 1;
    % white gaussian noise with variance r
    v     = sqrt(r) * randn(length(x)+length(h)-1, 1);
    % v = wgn(num_samples + length(h)-1,1, 10*log10(1e-4));
    % generate channel output
    y     = conv(x, h) + v;
    y     = y(length(h):end-length(h)+1); % observations, channel output
    x     = x(length(h):end); % constellations, info signal
    w_i   = 0.01 * randn(N,1); % initial equalizer random guess
    w_LMS      = zeros(N, samples_len+1); % equalizer
    w_LMS(:,1)= w_i;

    mu = 0.03; % step size
    % LMS Updates
    for k = N : num_samples % samples_len = 500
        if k == N + 150
            mu = 0.02;
        end
        it = k-N+1;
        error(s,it)    = x(k-d) - w_LMS(:,it)' * y(k:-1:it);
        w_LMS(:,it+1) = w_LMS(:,it) + mu * error(s,it) * y(k:-1:it);
    end
    sq_error(s,:) = error(s,:).^2;
end
end
```

Appendix-4 simulatecontgearshiftedLMSerrorcurve.m

```
function [sq_error] = simulatecontgearshiftedLMSerrorcurve(h,r,N,d, mu0
   )
% Settings
num_sim  = 1000; %number of process realizations
samples_len = 500; %number of time steps

% Initializations
num_samples = samples_len + N-1;
error    = zeros(num_sim, samples_len);
sq_error = zeros(num_sim, samples_len);
% mu0 = 0.03; % step size

% Simulations of LMS Equalization Updates
for s = 1:num_sim

    % generate binary data
    data = randi([0 1], num_samples+length(h)-1, 1);
    % bpsk constelltions
    x     = 2 * data - 1;
    % white gaussian noise with variance r
    v     = sqrt(r) * randn(length(x)+length(h)-1, 1);
    % v = wgn(num_samples + length(h)-1,1, 10*log10(1e-4));
    % generate channel output
    y     = conv(x, h) + v;
    y     = y(length(h):end-length(h)+1); % observations, channel output
    x     = x(length(h):end); % constellations, info signal
    w_i  = 0.01 * randn(N,1); % initial equalizer random guess
    w_LMS      = zeros(N, samples_len+1); % equalizer
    w_LMS(:,1)= w_i;

    % LMS Updates
    for k = N : num_samples % samples_len = 500
        it = k-N+1;
        mu = mu0/(1+k/200);
        error(s,it)    = x(k-d) - w_LMS(:,it)' * y(k:-1:it);
        w_LMS(:,it+1) = w_LMS(:,it) + mu * error(s,it) * y(k:-1:it);
    end
    sq_error(s,:) = error(s,:).^2;
end
end
```

Appendix-5 simulateLMSerrorcurvePW.m

```
function [sq_error] = simulateLMSerrorcurvePW(h,r, N, d, mu, Wpre)
%
% Prewhitening added
% Outputs
% Settings
num_sim  = 1000; %number of process realizations
samples_len = 500; %number of time steps

% Initializations
num_samples = samples_len + N-1;
error    = zeros(num_sim, samples_len);
sq_error = zeros(num_sim, samples_len);

% Simulations of LMS Equalization Updates
for s = 1:num_sim

    % generate binary data
    data = randi([0 1], num_samples+length(h)-1, 1);
    % bpsk constelltions
    x    = 2 * data - 1;
    % white gaussian noise with variance r
    v    = sqrt(r) * randn(length(x)+length(h)-1, 1);
    % v = wgn(num_samples + length(h)-1,1, 10*log10(1e-4));
    % generate channel output
    y    = conv(x, h) + v;
    y    = y(length(h):end-length(h)+1); % observations, channel output
    x    = x(length(h):end); % constellations, info signal
    w_i  = 0.01 * randn(N,1); % initial equalizer random guess
    w_LMS     = zeros(N, samples_len+1); % equalizer
    w_LMS(:,1)= w_i;

    % LMS Updates
    for k = N : num_samples % samples_len = 500
        it = k-N+1;
        yw = Wpre * y(k:-1:it);
        error(s,it)   = x(k-d) - w_LMS(:,it)' * yw;
        w_LMS(:,it+1) = w_LMS(:,it) + mu/(1+it/100) * error(s,it) * yw;
    end
    sq_error(s,:) = error(s,:).^2;
end
end
```

Appendix-6 simulateRLSerrorcurve.m

```
function [mean_eRLS] = simulateRLSerrorcurve(h,r,N,d,lambda)

num_sim  = 1000; %number of process realizations
samples_len = 500; %number of time steps
% d = 8;
delta = 1;

% Initializations

num_samples  = samples_len + N-1;
error_RLS    = zeros(num_sim, samples_len);
sq_error_RLS = zeros(num_sim, samples_len);

for s = 1:num_sim
    % generate binary data
    data = randi([0 1], num_samples+length(h)-1, 1);
    % bpsk constelltions
    x     = 2 * data - 1;
    % white gaussian noise with variance r
    v     = sqrt(r) * randn(length(x)+length(h)-1, 1);
    % v = wgn(num_samples + length(h)-1,1, 10*log10(1e-4));
    % generate channel output
    y     = conv(x, h) + v;
    xc    = x(length(h):end); % constellations, info signal
    yc    = y(length(h):end-length(h)+1); % observations, channel output
    wRLS = 0.01 * randn(N,1); % initial equalizer random guess
    PK    = delta * eye(N); % initialization of PK := (1/K) * Ryinv(K)
%     Ryyinv = inv(Ryy); % to compare

    % RLS Algorithm
    for K = d+1:samples_len+d
        yk = [yc(K:-1:max(1,K-N+1)) ; zeros(N-K,1)]; % current
            observation vector
        xk = xc(K-d);
        error_RLS(s,K-d)   = xk - (wRLS' * yk); % current error

        % Update equalizer
        temp = (lambda) + (yk' * PK * yk);
        wRLS = wRLS + (error_RLS(s,K-d)/temp) * PK * yk;
        %     deltaw(:,K) = norm(wopt-wRMS(:,K+1));

        % PK Update
        PKold = PK;
        PK    = (1/lambda)*(PK - ((PK* yk * yk' *PK)/temp));
        %     PKC(K)=norm(PK-(1/K)*inv(Ryy));
    end
    sq_error_RLS(s,:)=error_RLS(s,:).^2;
end
mean_eRLS = mean(sq_error_RLS ,1);
end
```

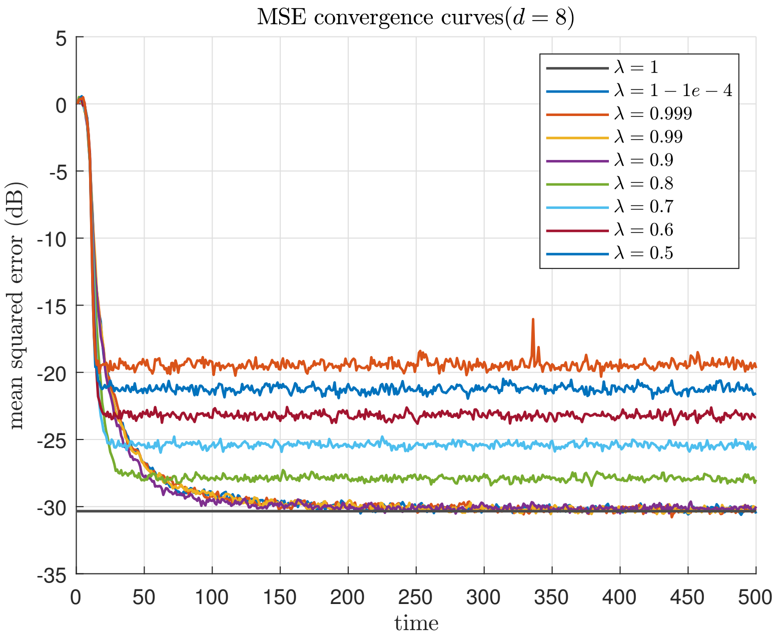

Figure 9: