# English Vowel Phonemes Identification

Submitted to: Dr. Qadri Mayyala

Department of Electrical and Computer Engineering

Birzeit University

Group members:
Yafa Naji 1200708
Nour Rahib , 1202853
Dunia Hamada 1201001

**The project abstract on system identification methods and techniques from traditional to modern and offline to online rapid analysis during the previous years to the present days. It takes into account part of AI technologies (soft computing technologies), primarily artificial neural network methods, for off-line system data analysis and real-time system identification, including linear and nonlinear issues. This study will provide a very brief survey of the researchers on the development of the definition of the system, its applications, and the techniques that were used.**

## INTRODUCTION

**The process of identifying the mathematical model that explains the behavior of a physical system based on a collection of input-output data is known as system identification. The goal of this project is to create a system identification algorithm capable of properly identifying the underlying dynamics of a physical system from a collection of measurable inputs and outputs.**

## EVALUATION CRITERIA

**Project evaluation mainly depends on the accuracy of the result. Accuracy of our project results is developed by working on the project. The results were fairly accurate, after working more on the project and studying more MATLAB and DSP techniques, the results were revised to give better accuracy. Results are developed and filter performance is evaluated by comparing the filtered output with the desired output, using metrics such as mean squared error, signal-to-noise ratio, or other performance criteria.**

**By working on the second part, we used a filter, and it was noticed that there were not many differences.**

## .PROBLEM SPECIFICATION

**The problem addressed in this project is to develop a system identification algorithm that can accurately identify the underlying dynamics of a physical system from a set of measured inputs and outputs. The algorithm should be able to handle nonlinear, time-varying systems with varying levels of noise and uncertainty.**

## DATA

**This project's data will be a mix of real-world signals and synthetic signals created via a physics-based simulation tool. The synthetic signals will be created using a physics-based simulation tool, while the real-world signals will be gathered from physical systems such as mechanical, electrical, and fluid systems.**

## APPROACH

The model was built in two different ways, but the results are similar and differ in complexity and speed.

An adaptive filter generally performs a spatial operation to identify pixels affected by pulse noise.The first method is applied "LMSFilter"() in Matlab. Using the lms() function.The second method is applied "RLSFilter" using Matlab's rls() function. We used desired And we got the value of the error in each of them

## RESULTS AND ANALYSIS

In general, the results from using LMS or RLS filters can include improved signal quality, reduced noise and interference, enhanced system performance, and increased accuracy of the estimated parameters. However, these results can also be affected by factors such as the convergence rate, the initialization of the filter parameters, and the presence of nonlinearities in the system.

It is important to carefully design and evaluate the performance of the LMS or RLS filter in the specific context of each application, taking into account the trade-offs between computational complexity, convergence speed, and accuracy of the estimated parameters.

## DEVELOPMENT

Enormous progress has been accomplished

It is still the major task in any advanced control project

Major challenge today: reduce the cost of identification

- Make it more user friendly and data-driven
- Optimize the experiment: less time, less energy
- Tune the experiment towards the application: don’t waste

Many research challenges remain, new ones have appeared:

- Structured systems
- Nonlinear systems
- Large distributed and network controlled systems

## CONCLUSION

The main operations of this project (System Identification) have included many concepts and basics of the DSP. The project has included dealing with filters main concept. And for sure many other concepts of DSP. Also, many MATLAB skills were gained after working on such a project. After finish working on the project, very satisfying results were obtained. But still, many more developments and features can be added to the project.

## REFERENCES

https://ch.mathworks.com/help/ident/gs/about-system-identification.html

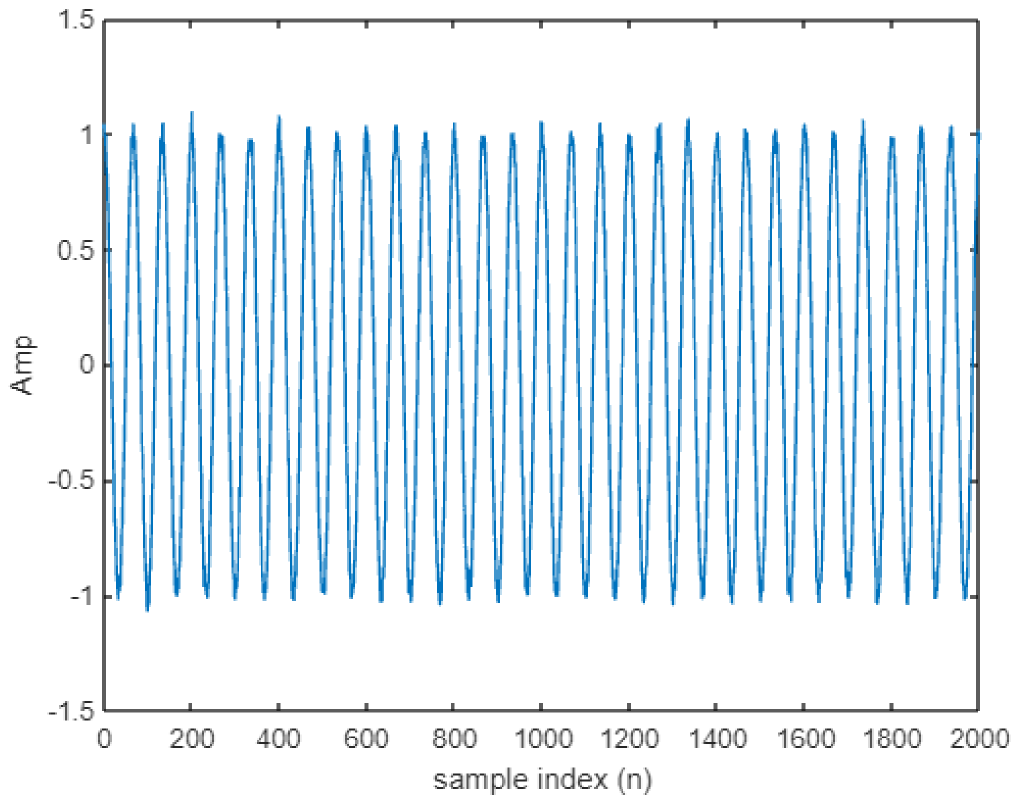
https://ch.mathworks.com/help/dsp/ref/dsp.lmsfilter-system-object.html

https://ch.mathworks.com/help/dsp/ref/dsp.rlsfilter-system-object.html

## Part1

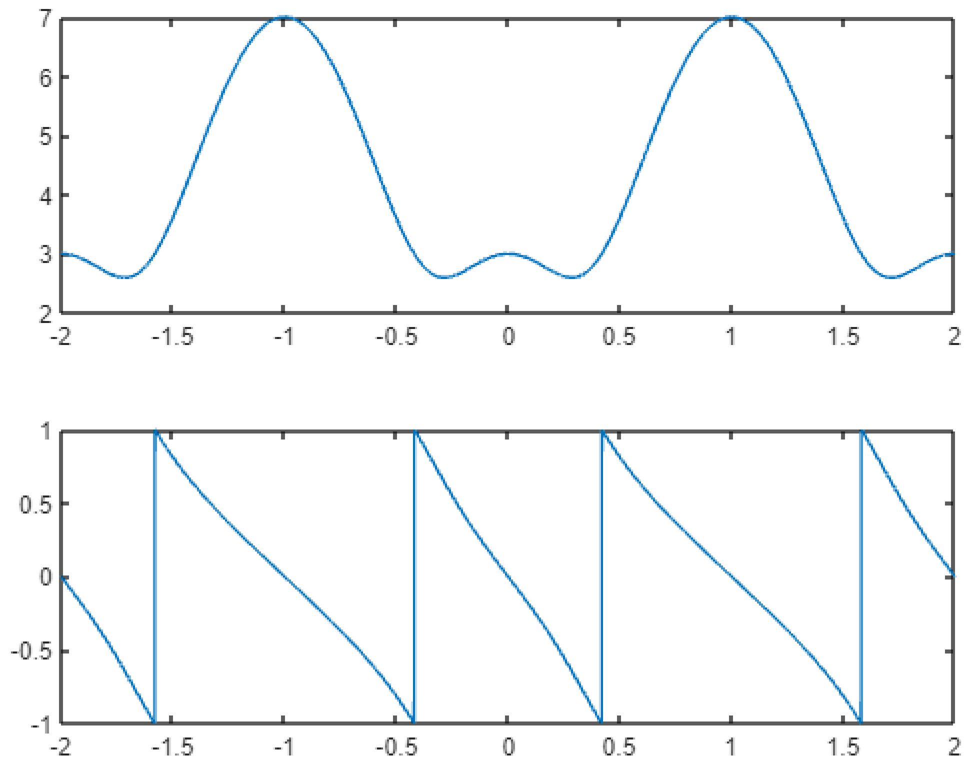
### A.

```
N=2000;
n=0:N-1;
x = cos(0.03*pi*n);
xn=awgn(x,30)
plot(n,xn)
xlabel("sample index (n)");
ylabel("Amp");
```



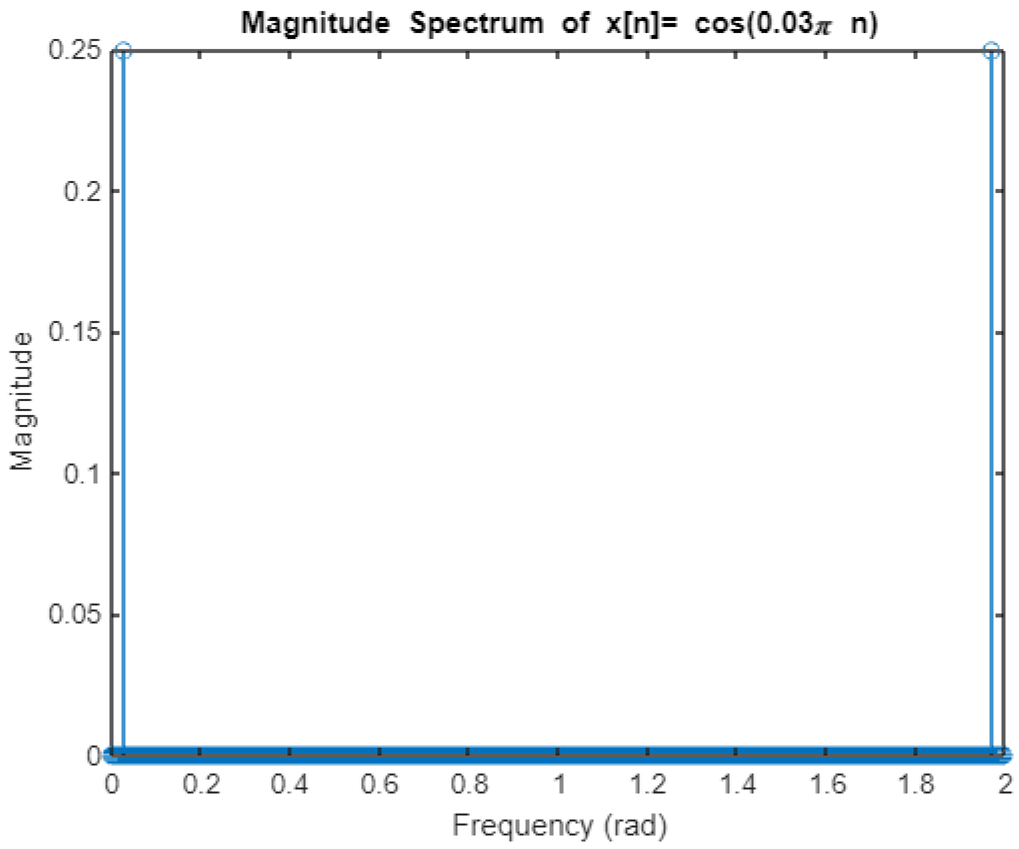
**B.**

```
w=-2:0.001:2;
w=w*pi;
H=1-2*exp(-j*w) +4*exp(-j*2*w);
figure
subplot(211),plot(w/pi,abs(H))
subplot(212),plot(w/pi,angle(H)/pi)
```

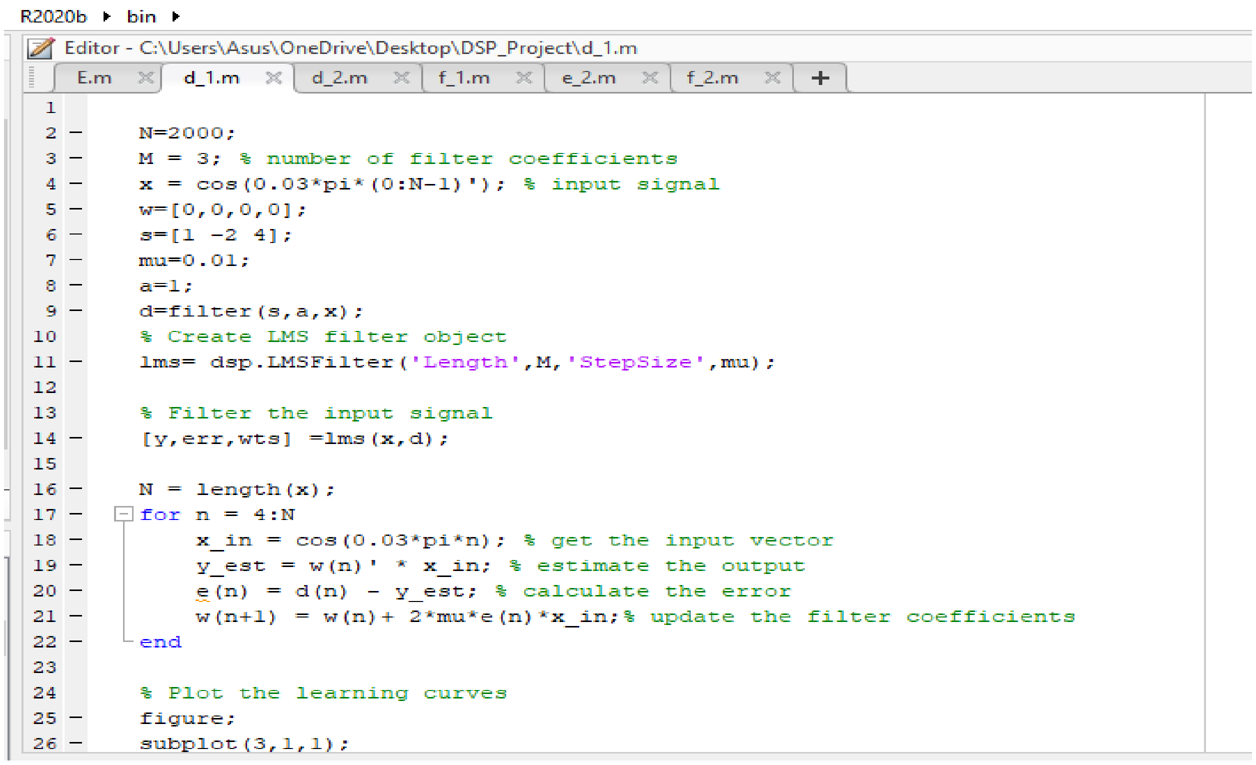


**C.**

```
N = 2000;
n = 0:N-1;
x = cos(0.03*pi*n);
stem(0:2/N:2-1/N, abs(fft(x)/N).^2);
xlabel('Frequency (rad)')
ylabel('Magnitude')
title('Magnitude Spectrum of x[n]= cos(0.03\pi n)')
```



**D**

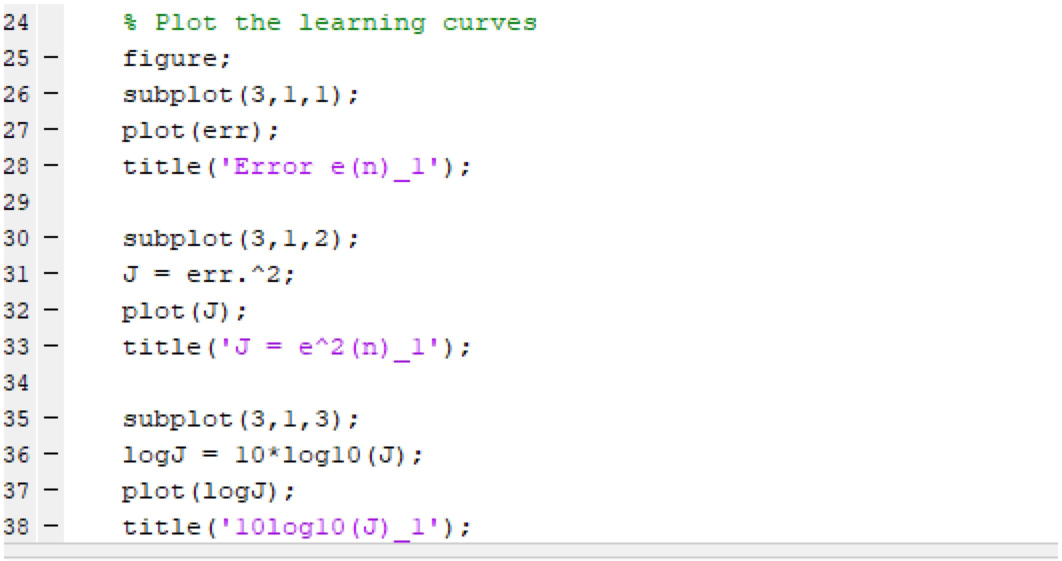


R2020b ▸ bin ▸

Editor - C:\Users\Asus\OneDrive\Desktop\DSP_Project\d_1.m

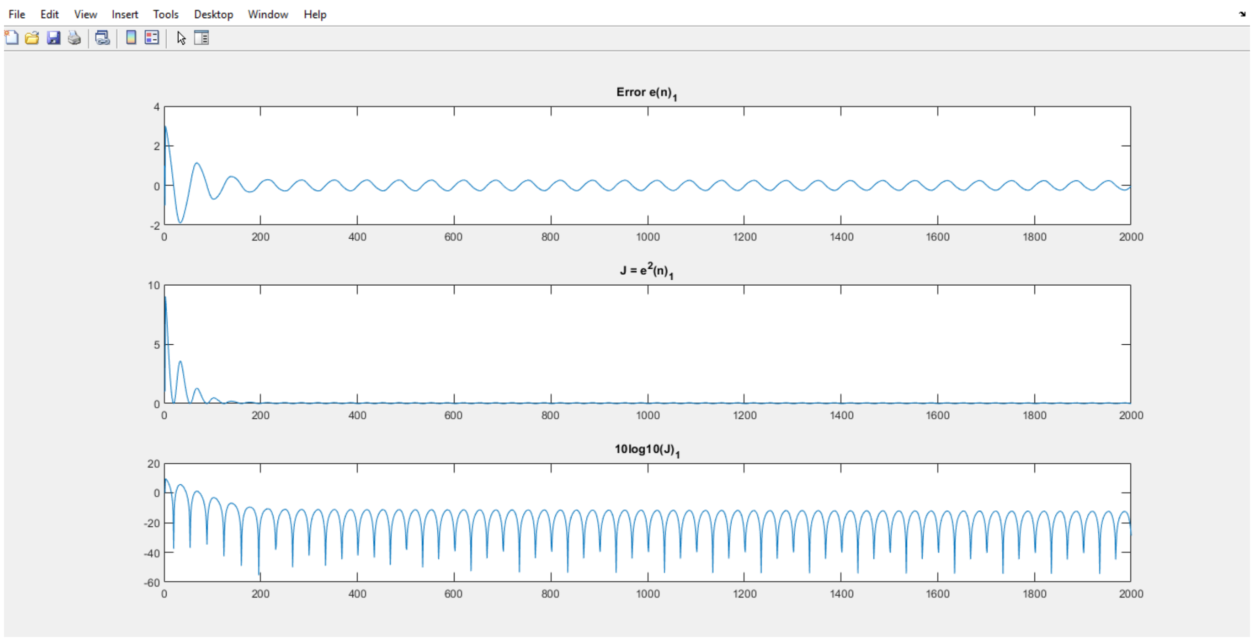
E.m | d_1.m | d_2.m | f_1.m | e_2.m | f_2.m | +

```

N=2000;
M = 3; % number of filter coefficients
x = cos(0.03*pi*(0:N-1)'); % input signal
w=[0,0,0,0];
s=[1 -2 4];
mu=0.01;
a=1;
d=filter(s,a,x);
% Create LMS filter object
lms= dsp.LMSFilter('Length',M,'StepSize',mu);

% Filter the input signal
[y,err,wts] =lms(x,d);

N = length(x);
for n = 4:N
    x_in = cos(0.03*pi*n); % get the input vector
    y_est = w(n)' * x_in; % estimate the output
    e(n) = d(n) - y_est; % calculate the error
    w(n+1) = w(n)+ 2*mu*e(n)*x_in;% update the filter coefficients
end

% Plot the learning curves
figure;
subplot(3,1,1);
```

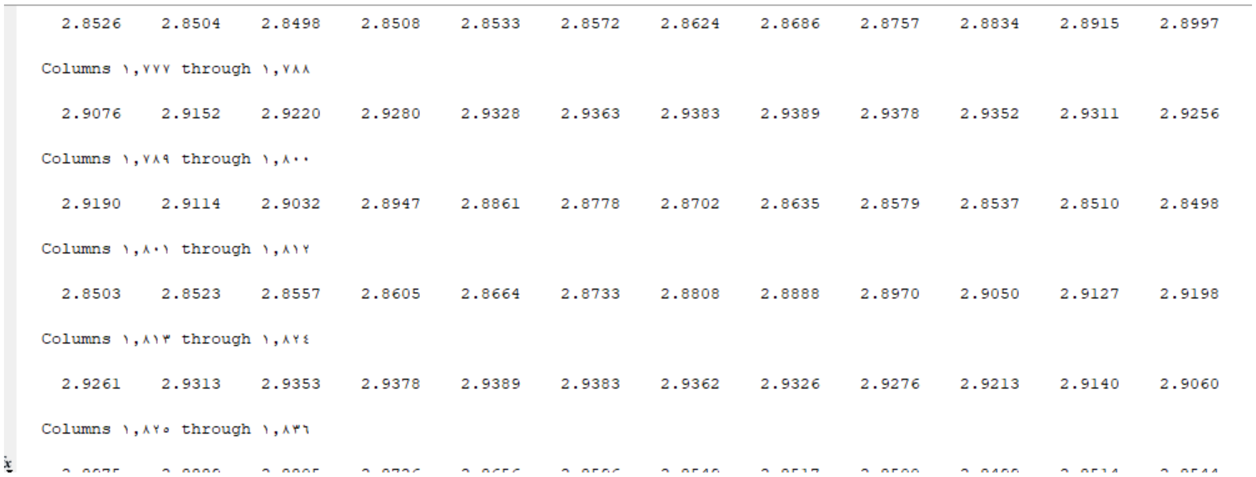


```
% Plot the learning curves
figure;
subplot(3,1,1);
plot(err);
title('Error e(n)_1');

subplot(3,1,2);
J = err.^2;
plot(J);
title('J = e^2(n)_1');

subplot(3,1,3);
logJ = 10*log10(J);
plot(logJ);
title('10log10(J)_1');
```

T



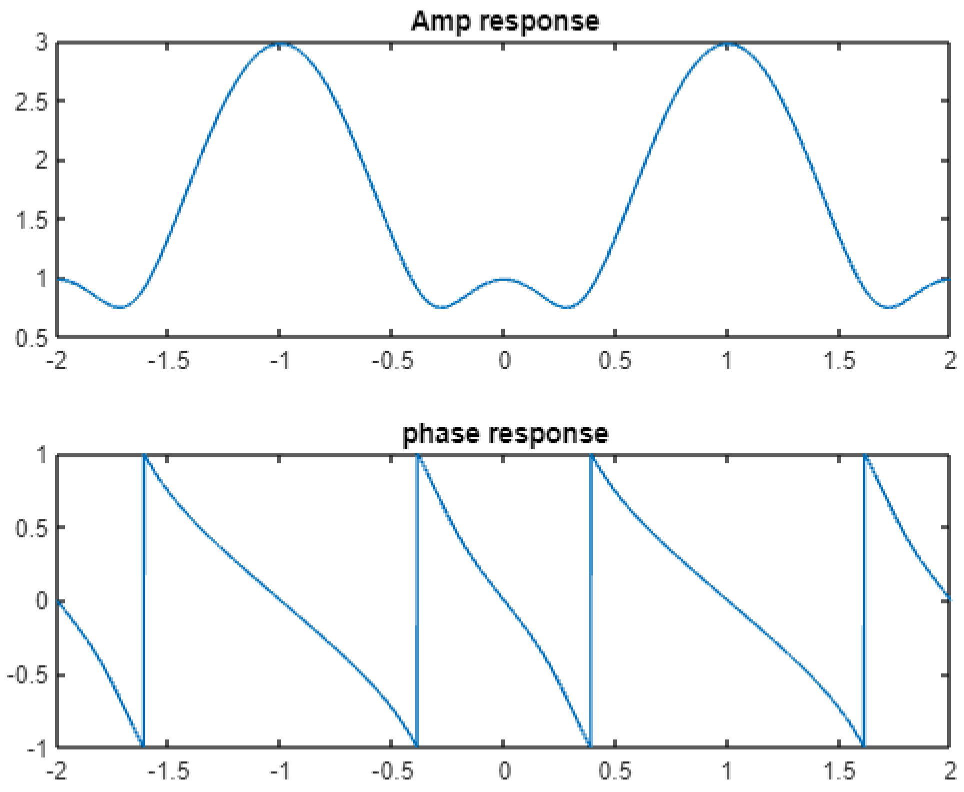
## The estimate of W



```
    2.8526    2.8504    2.8498    2.8508    2.8533    2.8572    2.8624    2.8686    2.8757    2.8834    2.8915    2.8997

  Columns ١,٧٧٧ through ١,٧٨٨

    2.9076    2.9152    2.9220    2.9280    2.9328    2.9363    2.9383    2.9389    2.9378    2.9352    2.9311    2.9256

  Columns ١,٧٨٩ through ١,٨٠٠

    2.9190    2.9114    2.9032    2.8947    2.8861    2.8778    2.8702    2.8635    2.8579    2.8537    2.8510    2.8498

  Columns ١,٨٠١ through ١,٨١٢

    2.8503    2.8523    2.8557    2.8605    2.8664    2.8733    2.8808    2.8888    2.8970    2.9050    2.9127    2.9198

  Columns ١,٨١٣ through ١,٨٢٤

    2.9261    2.9313    2.9353    2.9378    2.9389    2.9383    2.9362    2.9326    2.9276    2.9213    2.9140    2.9060

  Columns ١,٨٢٥ through ١,٨٣٦
```

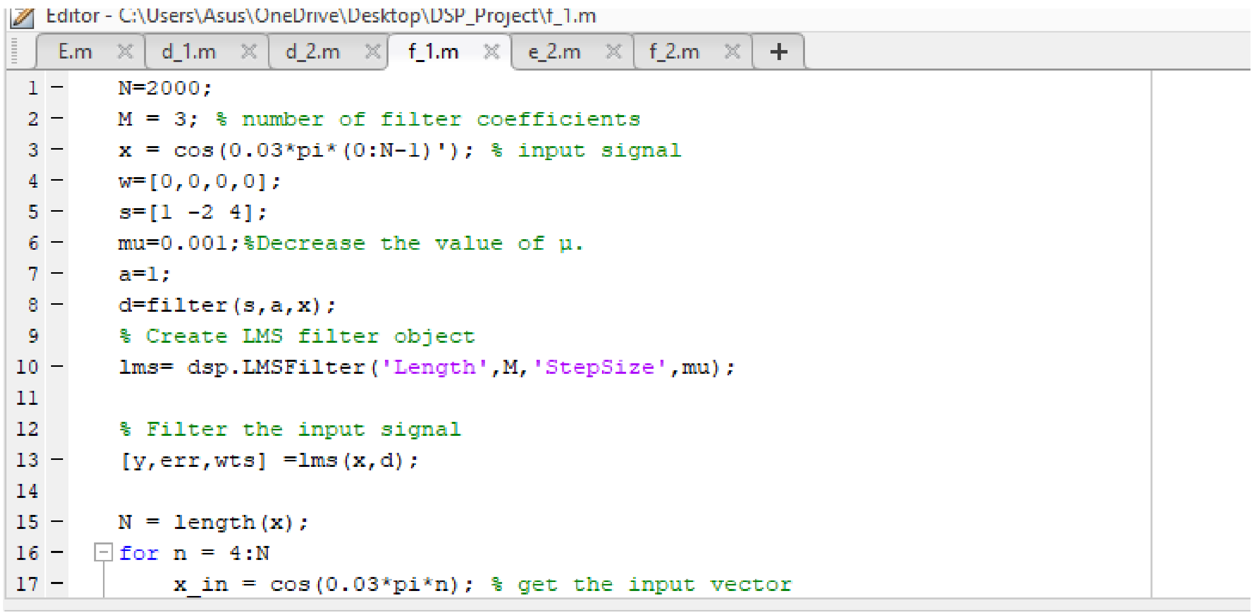
## E

```
w=-2:0.001:2;
w=w*pi;
H1=0.5357-0.9933*exp(-j*w) +1.4421*exp(-j*2*w);
figure
subplot(211);
plot(w/pi,abs(H1));
title("Amp response");
subplot(212);
plot(w/pi,angle(H1)/pi);
title("phase response");
```

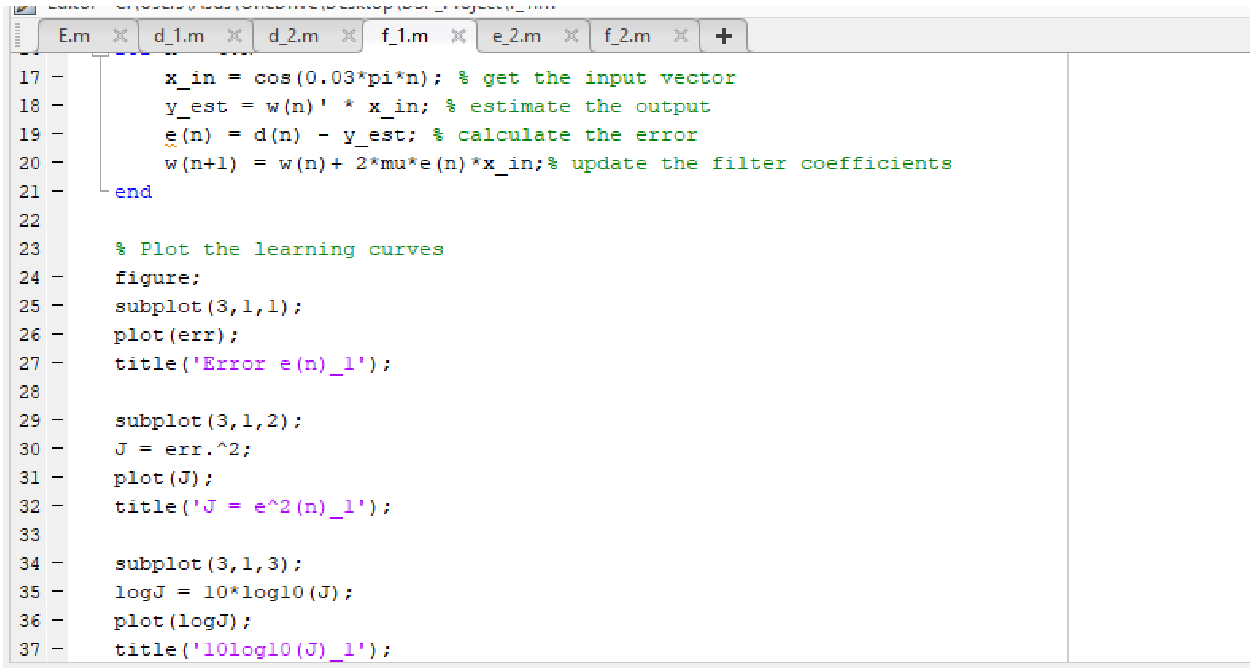


**The difference between it and branch B is the change the amplitude due to the change in the coefficients**

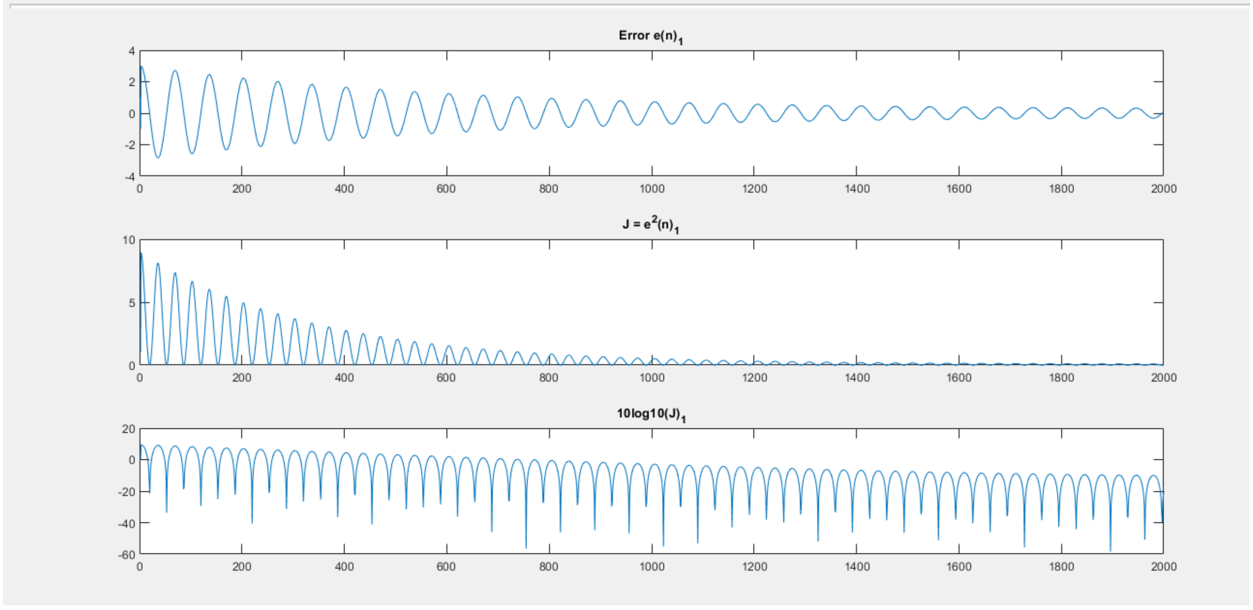
**F**



```
N=2000;
M = 3; % number of filter coefficients
x = cos(0.03*pi*(0:N-1)'); % input signal
w=[0,0,0,0];
s=[1 -2 4];
mu=0.001;%Decrease the value of µ.
a=1;
d=filter(s,a,x);
% Create LMS filter object
lms= dsp.LMSFilter('Length',M,'StepSize',mu);

% Filter the input signal
[y,err,wts] =lms(x,d);

N = length(x);
for n = 4:N
    x_in = cos(0.03*pi*n); % get the input vector
```



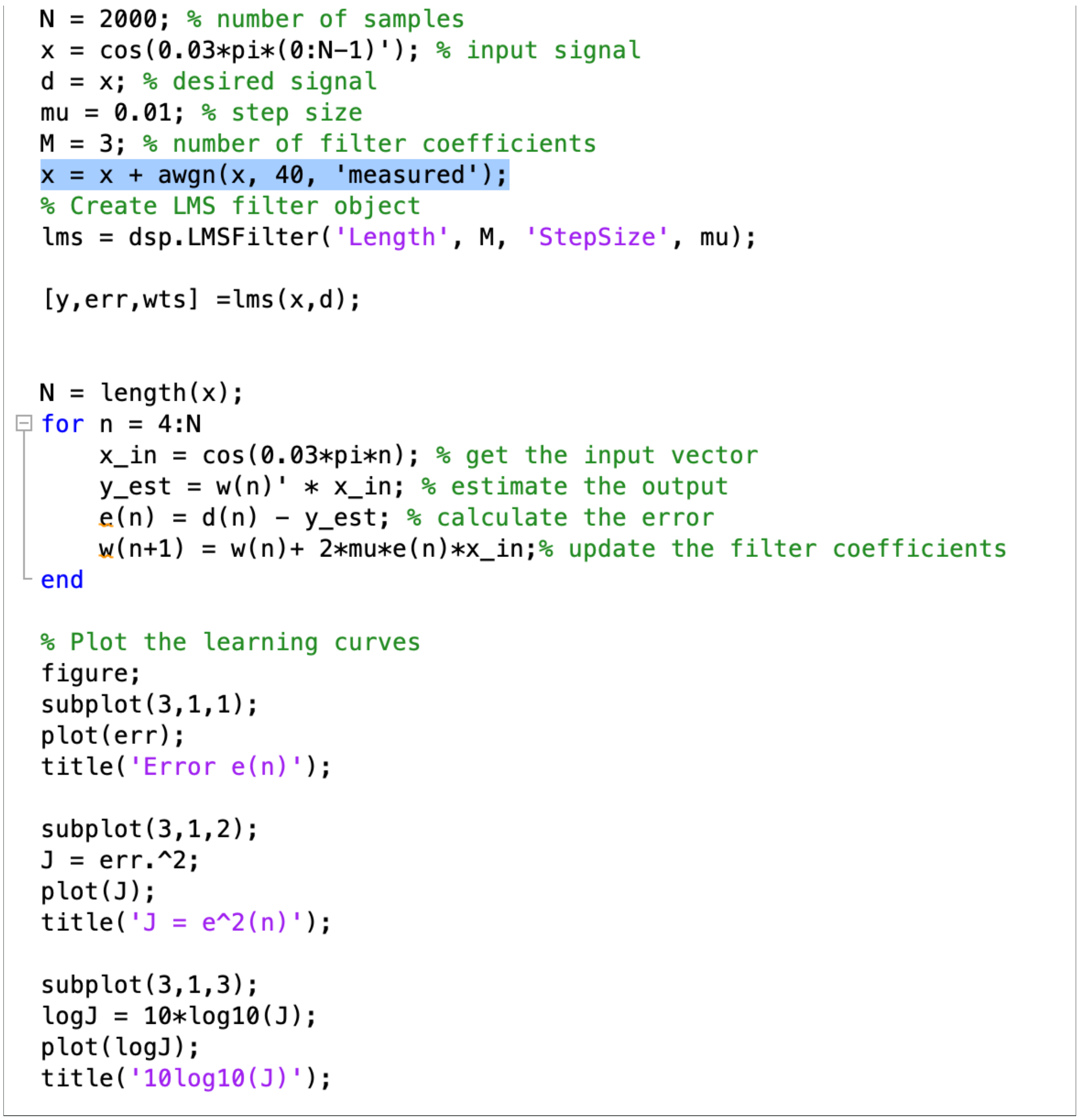
```matlab
        x_in = cos(0.03*pi*n); % get the input vector
        y_est = w(n)' * x_in; % estimate the output
        e(n) = d(n) - y_est; % calculate the error
        w(n+1) = w(n)+ 2*mu*e(n)*x_in;% update the filter coefficients
    end

    % Plot the learning curves
    figure;
    subplot(3,1,1);
    plot(err);
    title('Error e(n)_1');

    subplot(3,1,2);
    J = err.^2;
    plot(J);
    title('J = e^2(n)_1');

    subplot(3,1,3);
    logJ = 10*log10(J);
    plot(logJ);
    title('10log10(J)_1');
```



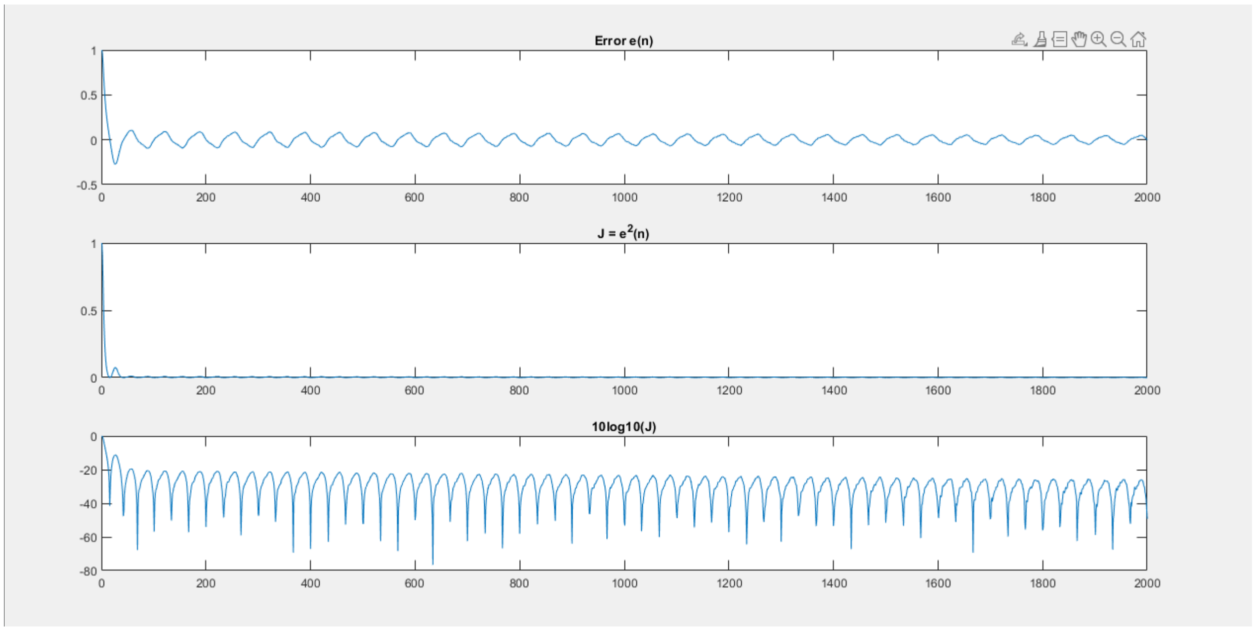
**Decreasing the value of μ will decrease the speed of the learning process. This means that the controller will respond more slowly to changes in the system, which can be beneficial for avoiding overshoot and oscillation. However, it can also result in slower convergence to the desired setpoint.**

**Steady-state error: Decreasing the value of μ will increase the steady-state error. This means that the final error between the actual output and the desired setpoint will be larger. However, it can also result in a more stable system, as the controller will be less sensitive to noise and disturbances in the system.**

**G_as_D**

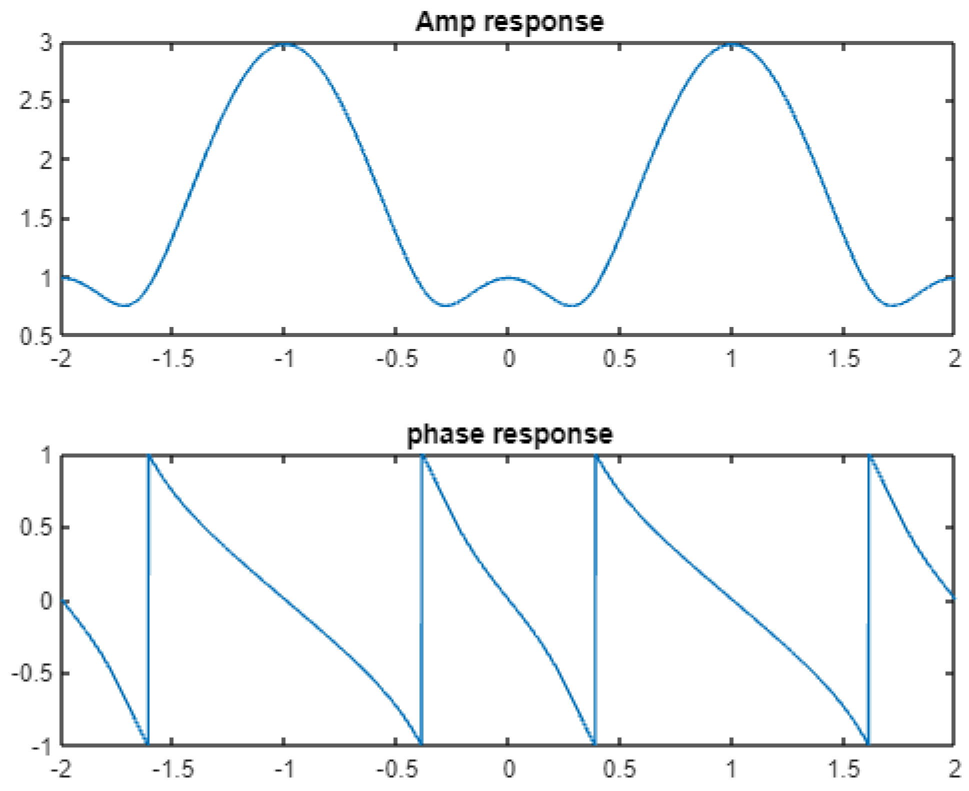


```
N = 2000; % number of samples
x = cos(0.03*pi*(0:N-1)'); % input signal
d = x; % desired signal
mu = 0.01; % step size
M = 3; % number of filter coefficients
x = x + awgn(x, 40, 'measured');
% Create LMS filter object
lms = dsp.LMSFilter('Length', M, 'StepSize', mu);

[y,err,wts] =lms(x,d);


N = length(x);
for n = 4:N
    x_in = cos(0.03*pi*n); % get the input vector
    y_est = w(n)' * x_in; % estimate the output
    e(n) = d(n) - y_est; % calculate the error
    w(n+1) = w(n)+ 2*mu*e(n)*x_in;% update the filter coefficients
end

% Plot the learning curves
figure;
subplot(3,1,1);
plot(err);
title('Error e(n)');

subplot(3,1,2);
J = err.^2;
plot(J);
title('J = e^2(n)');

subplot(3,1,3);
logJ = 10*log10(J);
plot(logJ);
title('10log10(J)');
```

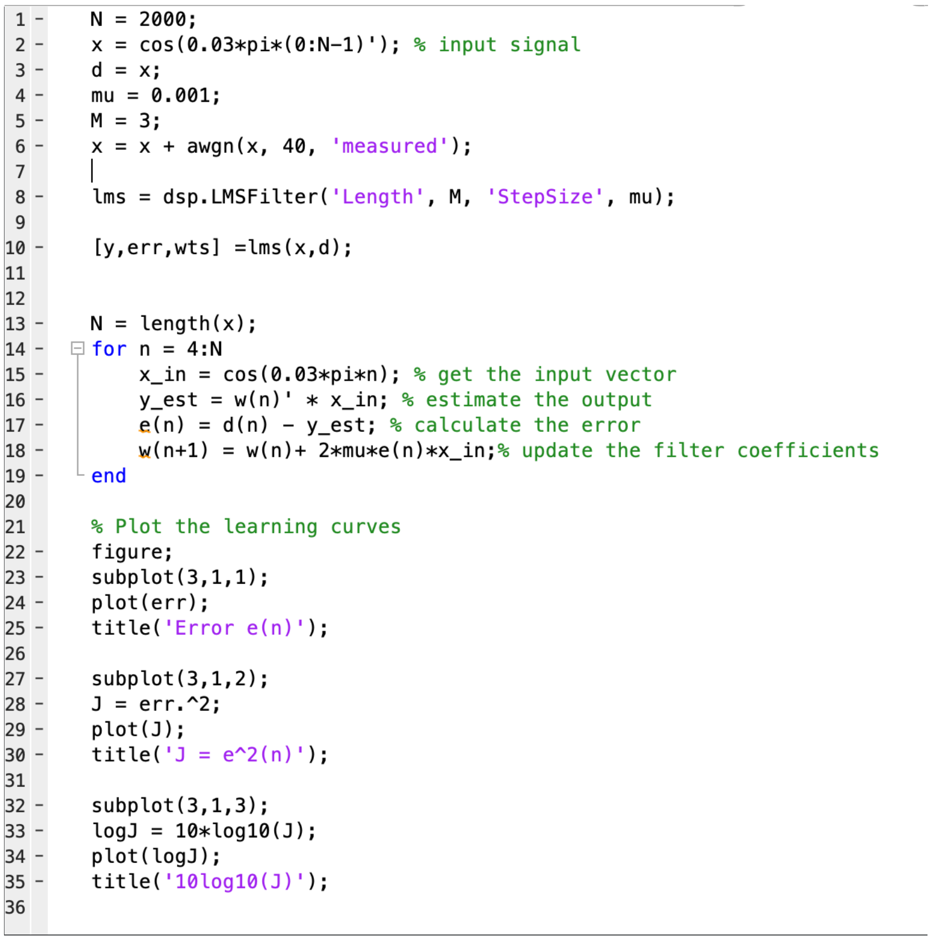


## G_as_E

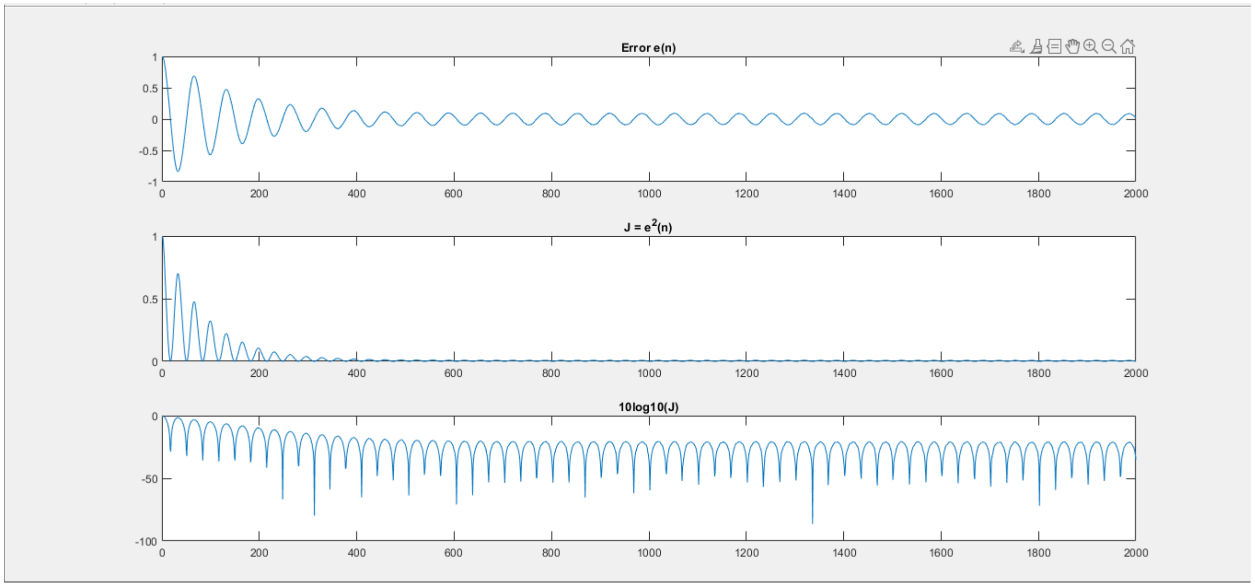
```
w=-2:0.001:2;
w=w*pi;
H1=0.5357-0.9933*exp(-j*w) +1.4421*exp(-j*2*w);
x=H1;
noisy_x=awgn(x,40);
figure
subplot(2,1,1);
plot(w/pi,abs(H1));
title("Amp response");
subplot(2,1,2);
plot(w/pi,angle(H1)/pi);
title("phase response");
```



## G_as_F

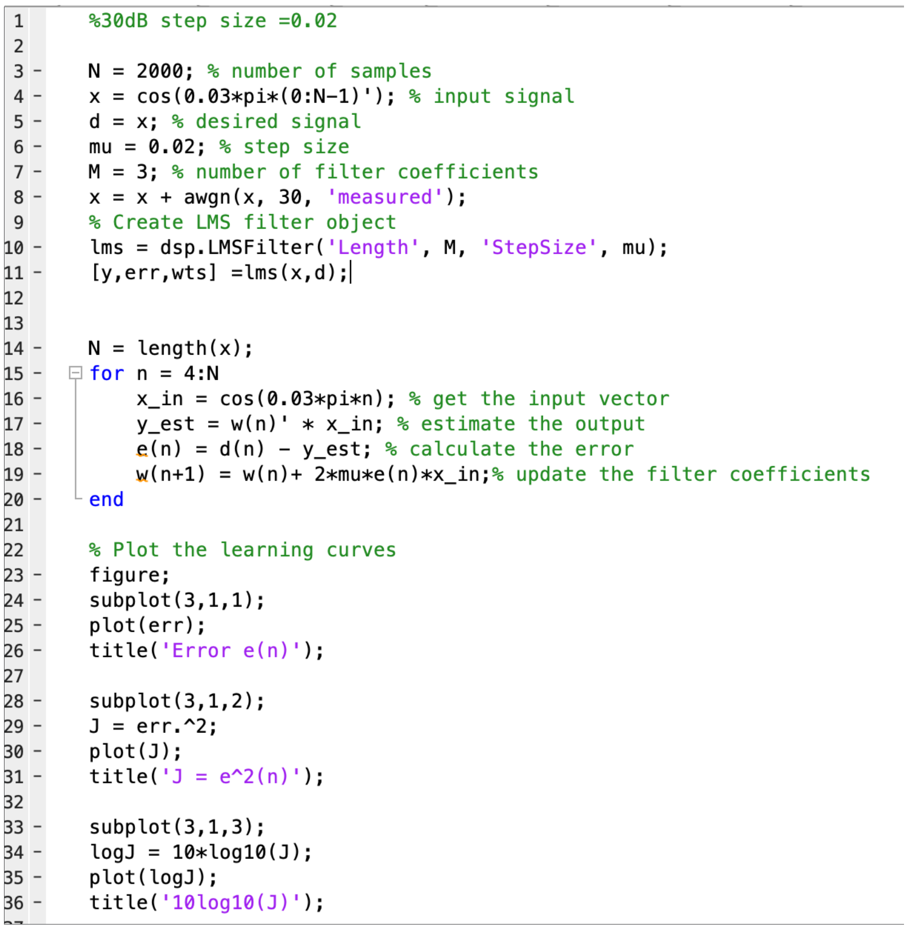


```
N = 2000;
x = cos(0.03*pi*(0:N-1)'); % input signal
d = x;
mu = 0.001;
M = 3;
x = x + awgn(x, 40, 'measured');

lms = dsp.LMSFilter('Length', M, 'StepSize', mu);

[y,err,wts] =lms(x,d);


N = length(x);
for n = 4:N
    x_in = cos(0.03*pi*n); % get the input vector
    y_est = w(n)' * x_in; % estimate the output
    e(n) = d(n) - y_est; % calculate the error
    w(n+1) = w(n)+ 2*mu*e(n)*x_in;% update the filter coefficients
end

% Plot the learning curves
figure;
subplot(3,1,1);
plot(err);
title('Error e(n)');

subplot(3,1,2);
J = err.^2;
plot(J);
title('J = e^2(n)');

subplot(3,1,3);
logJ = 10*log10(J);
plot(logJ);
title('10log10(J)');
```

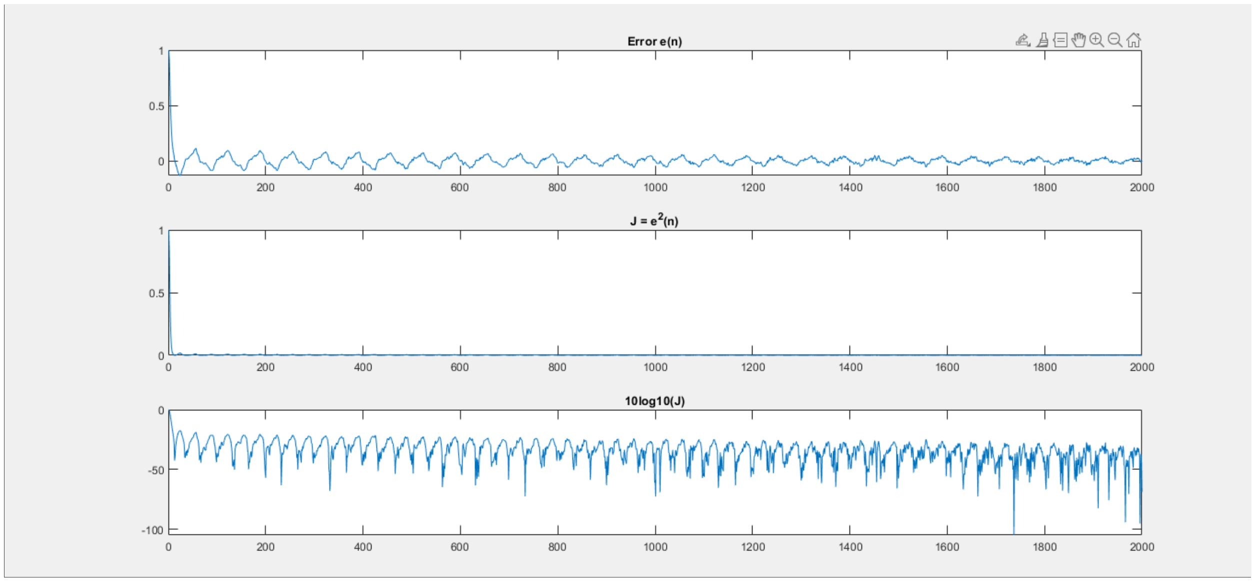


**After adding 40dB of zero mean white Gaussian noise to the signal x[n], the results of parts (D)-(F) change. This is because the added noise may cause the filter to perform less accurately**

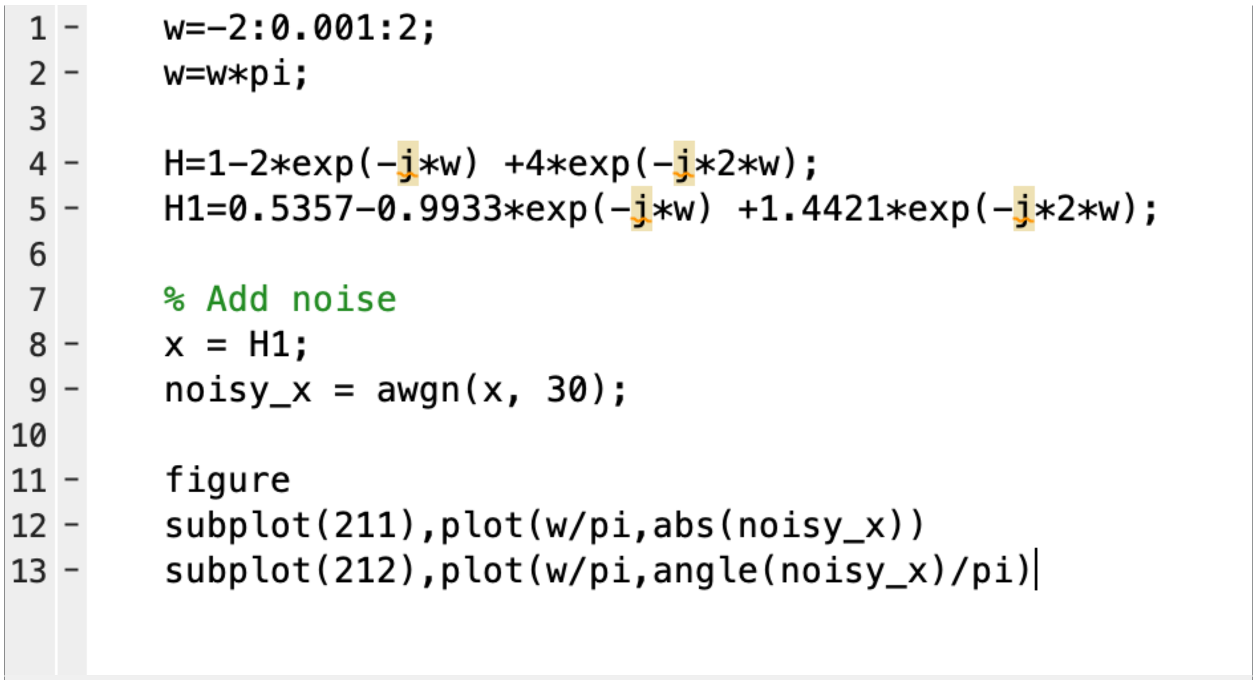
## H_as_D



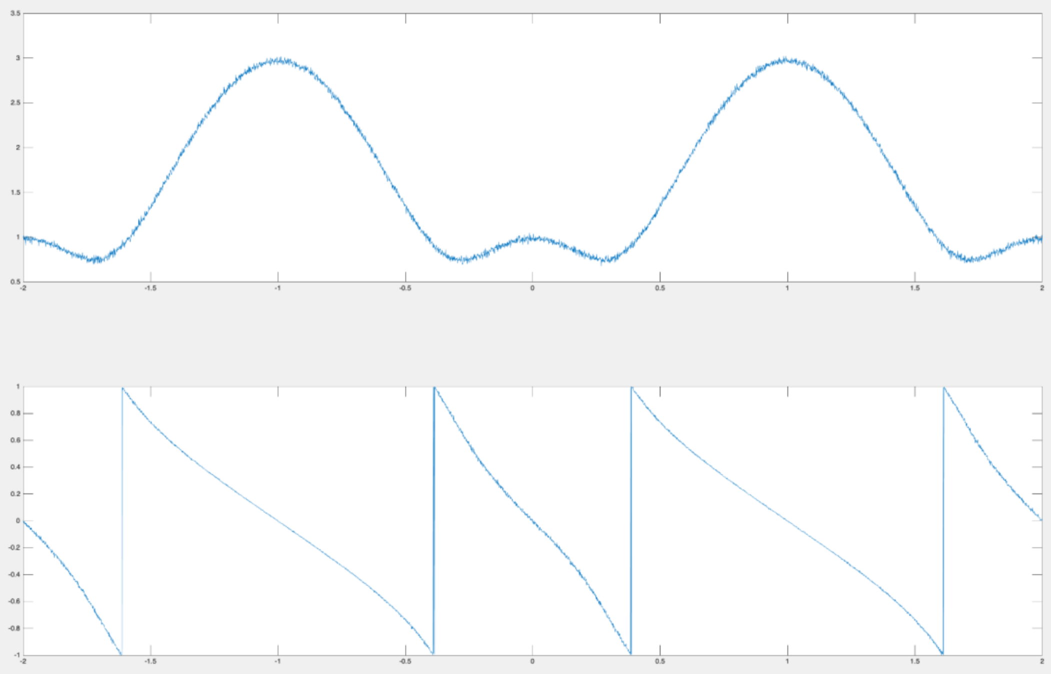
```matlab
%30dB step size =0.02

N = 2000; % number of samples
x = cos(0.03*pi*(0:N-1)'); % input signal
d = x; % desired signal
mu = 0.02; % step size
M = 3; % number of filter coefficients
x = x + awgn(x, 30, 'measured');
% Create LMS filter object
lms = dsp.LMSFilter('Length', M, 'StepSize', mu);
[y,err,wts] =lms(x,d);


N = length(x);
for n = 4:N
    x_in = cos(0.03*pi*n); % get the input vector
    y_est = w(n)' * x_in; % estimate the output
    e(n) = d(n) - y_est; % calculate the error
    w(n+1) = w(n)+ 2*mu*e(n)*x_in;% update the filter coefficients
end

% Plot the learning curves
figure;
subplot(3,1,1);
plot(err);
title('Error e(n)');

subplot(3,1,2);
J = err.^2;
plot(J);
title('J = e^2(n)');

subplot(3,1,3);
logJ = 10*log10(J);
plot(logJ);
title('10log10(J)');
```



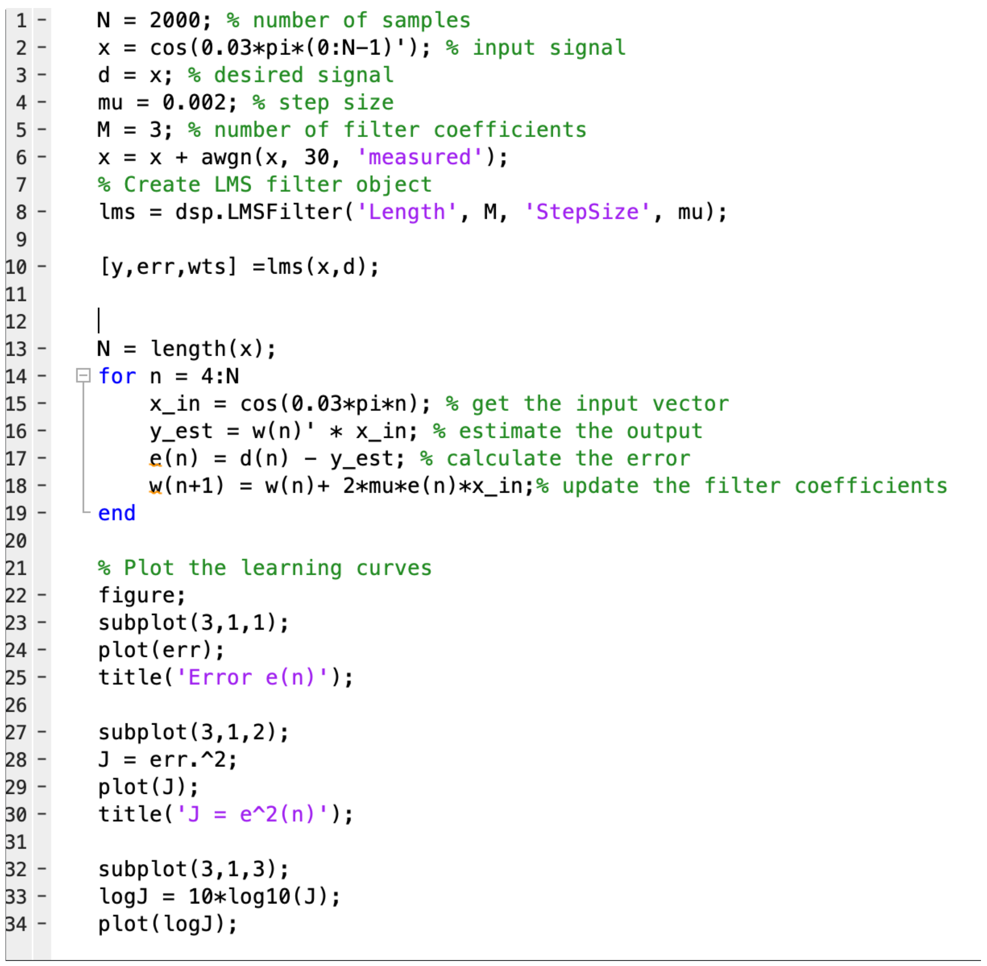
**H_as_E**



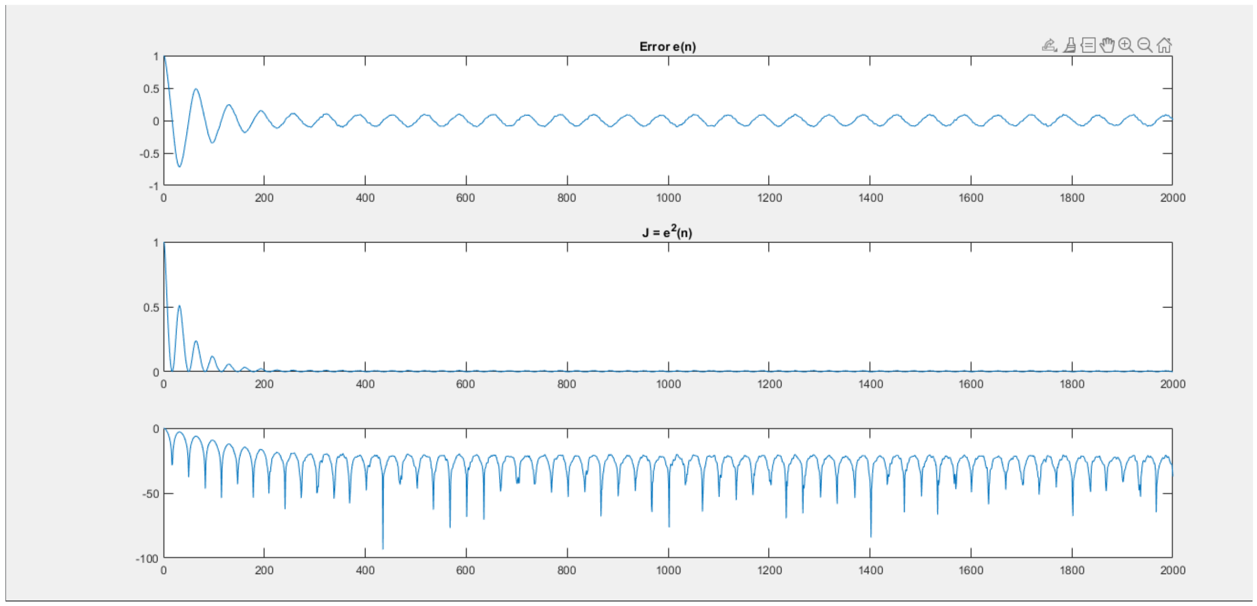
```
w=-2:0.001:2;
w=w*pi;

H=1-2*exp(-j*w) +4*exp(-j*2*w);
H1=0.5357-0.9933*exp(-j*w) +1.4421*exp(-j*2*w);

% Add noise
x = H1;
noisy_x = awgn(x, 30);

figure
subplot(211),plot(w/pi,abs(noisy_x))
subplot(212),plot(w/pi,angle(noisy_x)/pi)
```



## H_as_F

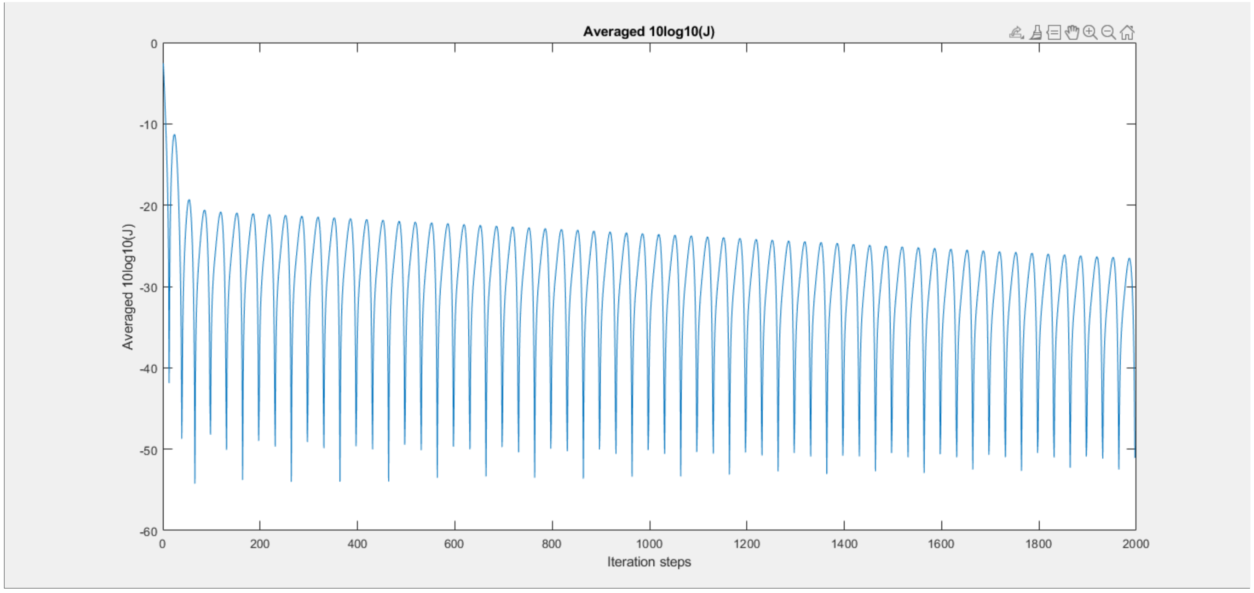


```matlab
N = 2000; % number of samples
x = cos(0.03*pi*(0:N-1)'); % input signal
d = x; % desired signal
mu = 0.002; % step size
M = 3; % number of filter coefficients
x = x + awgn(x, 30, 'measured');
% Create LMS filter object
lms = dsp.LMSFilter('Length', M, 'StepSize', mu);

[y,err,wts] =lms(x,d);


N = length(x);
for n = 4:N
    x_in = cos(0.03*pi*n); % get the input vector
    y_est = w(n)' * x_in; % estimate the output
    e(n) = d(n) - y_est; % calculate the error
    w(n+1) = w(n)+ 2*mu*e(n)*x_in;% update the filter coefficients
end

% Plot the learning curves
figure;
subplot(3,1,1);
plot(err);
title('Error e(n)');

subplot(3,1,2);
J = err.^2;
plot(J);
title('J = e^2(n)');

subplot(3,1,3);
logJ = 10*log10(J);
plot(logJ);
```



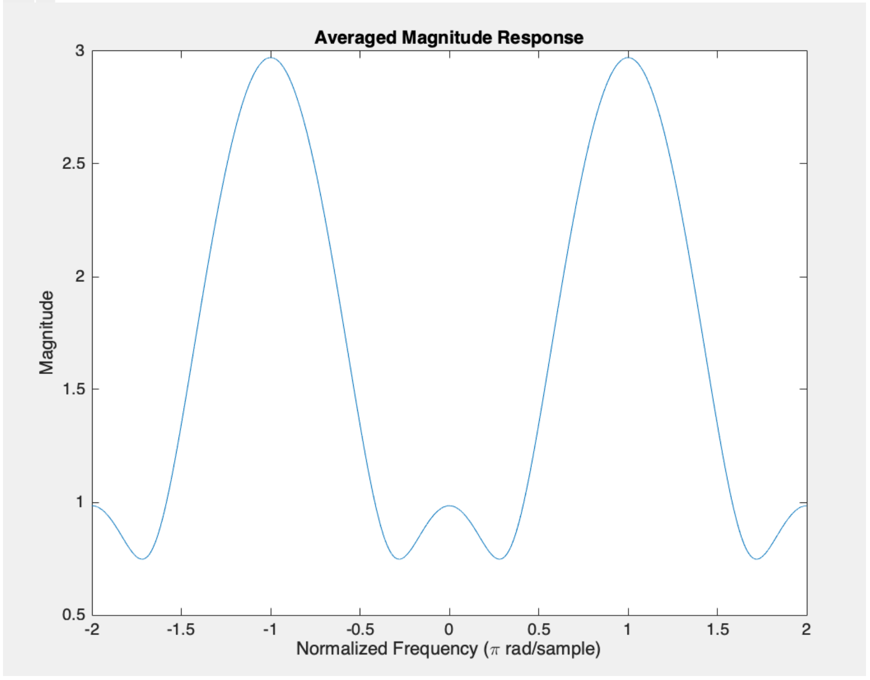
## I_as_D

```
num_trials = 1000;
J_avg = zeros(N-3,1);

for i = 1:num_trials
    x = cos(0.03*pi*(0:N-1)');
    d = x;
    mu = 0.01;
    M = 3;
    x = x + awgn(x, 40, 'measured');
    % Create LMS filter object
    lms = dsp.LMSFilter('Length', M, 'StepSize', mu);

    [y,err,wts] =lms(x,d);

    J = zeros(N-3,1); % initialize J with zeros before each trial
    for n = 4:N
        x_in = cos(0.03*pi*n); % get the input vector
        y_est = w(n)' * x_in;
        e(n) = d(n) - y_est;
        w(n+1) = w(n)+ 2*mu*e(n)*x_in;% update the filter coefficients
        J(n-3) = err(n)^2;
    end
    J_avg = J_avg + J;
end

J_avg = J_avg / num_trials;
logJ_avg = 10*log10(J_avg);

% Plot the averaged learning curves
figure;
plot(logJ_avg);
title('Averaged 10log10(J)');
xlabel('Iteration steps');
ylabel('Averaged 10log10(J)');
```

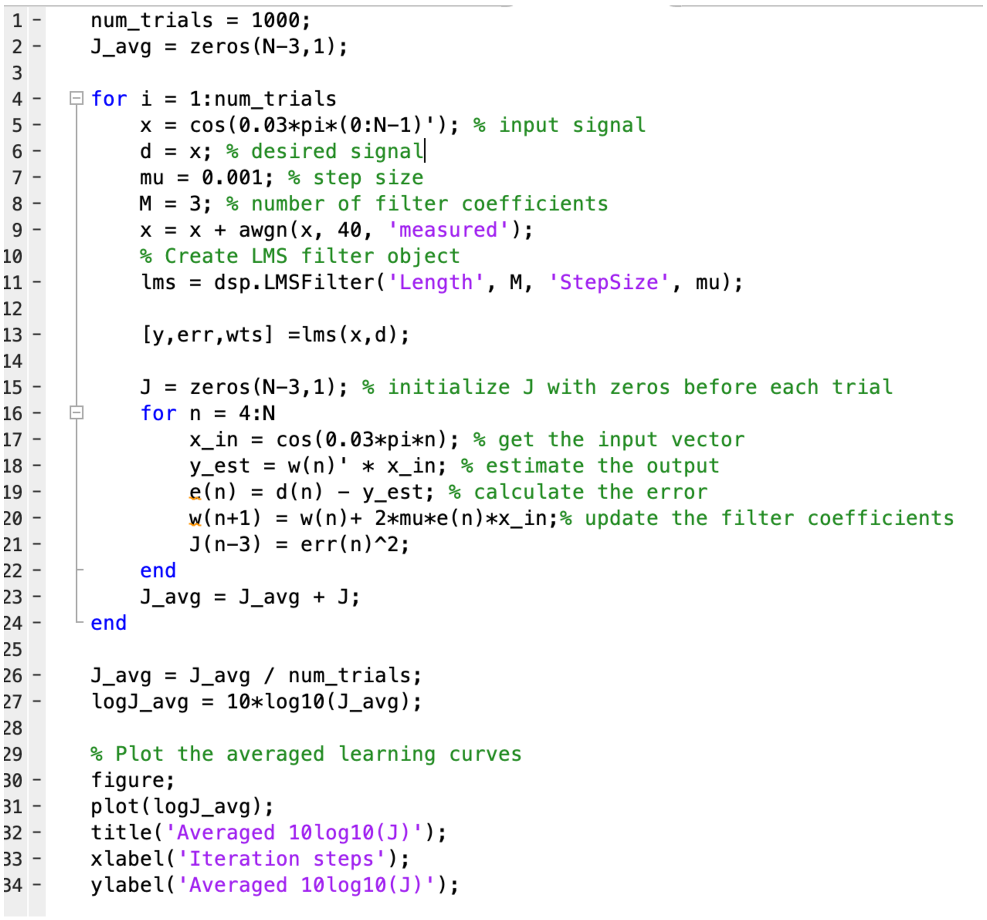


## I_as_E

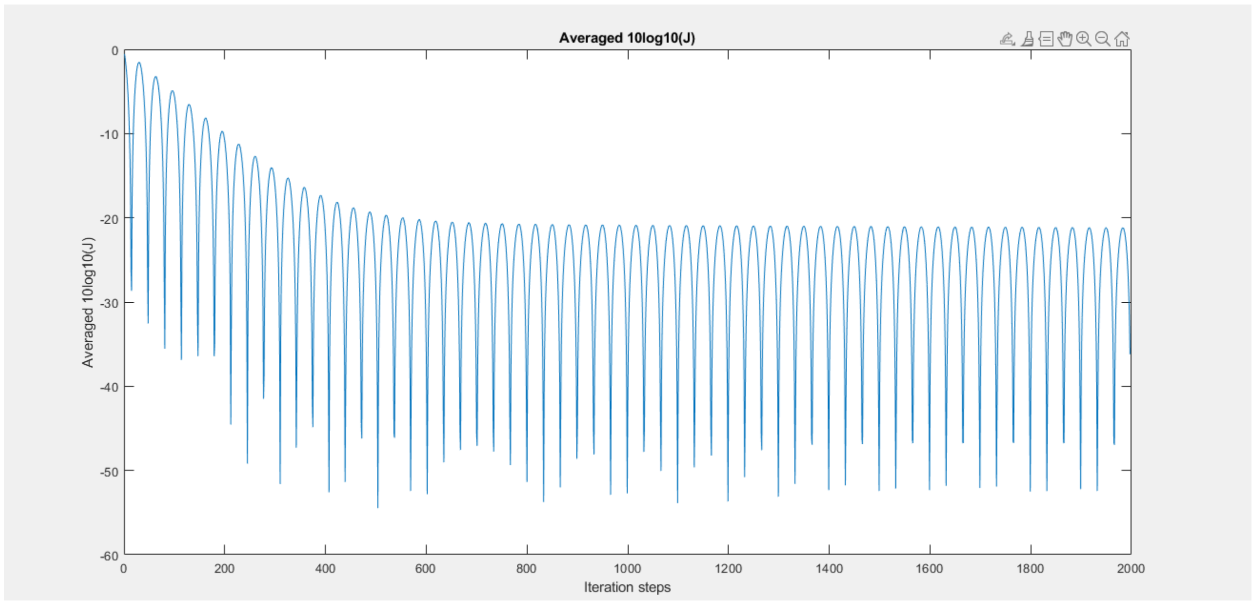
```
num_trials = 1000;
w = -2:0.001:2;
w = w * pi;
N = length(w);
H_avg = zeros(1, N);

for i = 1:num_trials
    H1=0.5357-0.9933*exp(-1i*w) +1.4421*exp(-1i*2*w);
    noisy_x = awgn(H1, 40);
    H_avg = H_avg + abs(noisy_x);
end

H_avg = H_avg / num_trials;

% Plot the averaged magnitude response
figure;
plot(w/pi, H_avg);
title('Averaged Magnitude Response');
xlabel('Normalized Frequency (\pi rad/sample)');
ylabel('Magnitude');
```



## I_as_F

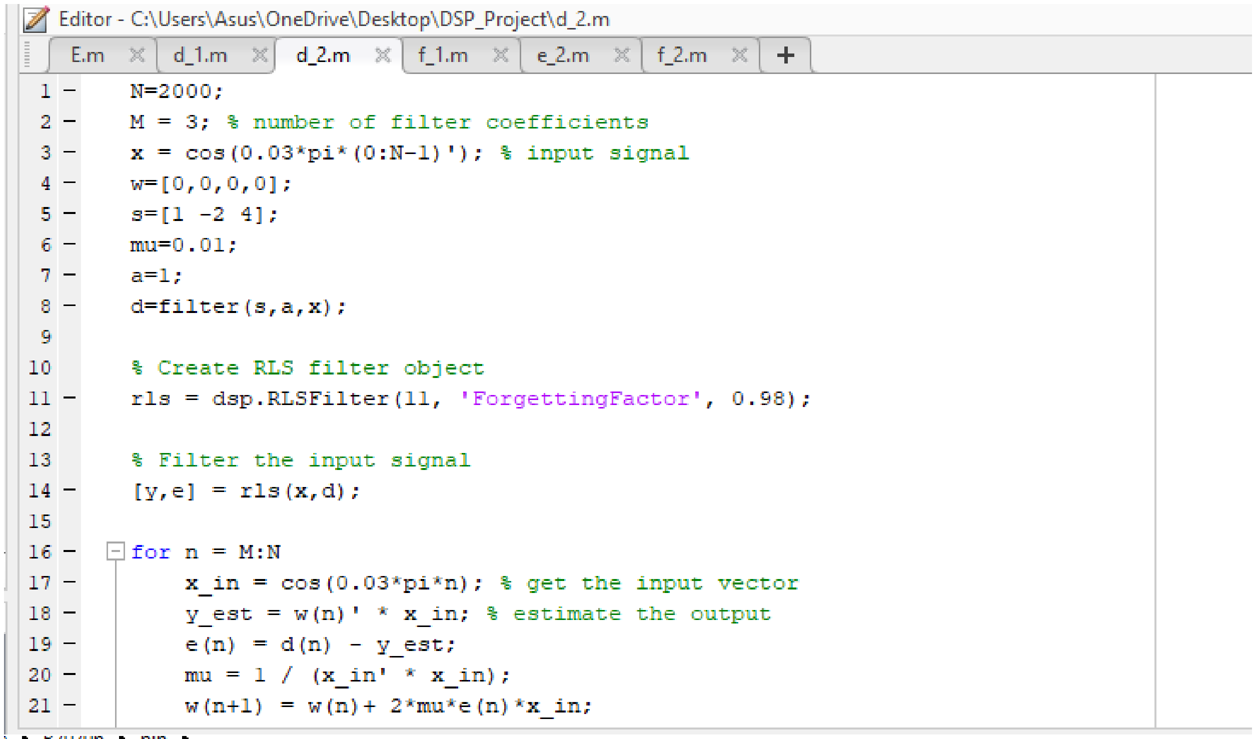


```
num_trials = 1000;
J_avg = zeros(N-3,1);

for i = 1:num_trials
    x = cos(0.03*pi*(0:N-1)'); % input signal
    d = x; % desired signal
    mu = 0.001; % step size
    M = 3; % number of filter coefficients
    x = x + awgn(x, 40, 'measured');
    % Create LMS filter object
    lms = dsp.LMSFilter('Length', M, 'StepSize', mu);

    [y,err,wts] =lms(x,d);

    J = zeros(N-3,1); % initialize J with zeros before each trial
    for n = 4:N
        x_in = cos(0.03*pi*n); % get the input vector
        y_est = w(n)' * x_in; % estimate the output
        e(n) = d(n) - y_est; % calculate the error
        w(n+1) = w(n)+ 2*mu*e(n)*x_in;% update the filter coefficients
        J(n-3) = err(n)^2;
    end
    J_avg = J_avg + J;
end

J_avg = J_avg / num_trials;
logJ_avg = 10*log10(J_avg);

% Plot the averaged learning curves
figure;
plot(logJ_avg);
title('Averaged 10log10(J)');
xlabel('Iteration steps');
ylabel('Averaged 10log10(J)');
```



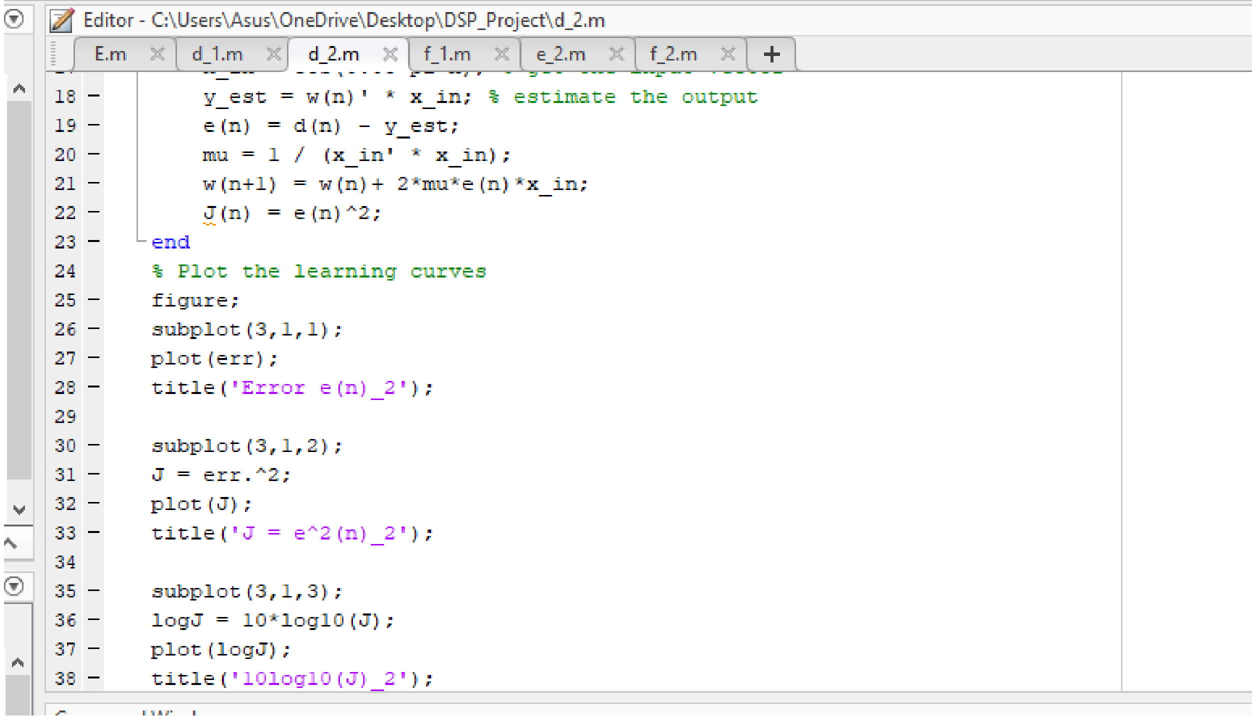
## Part 2

**D**



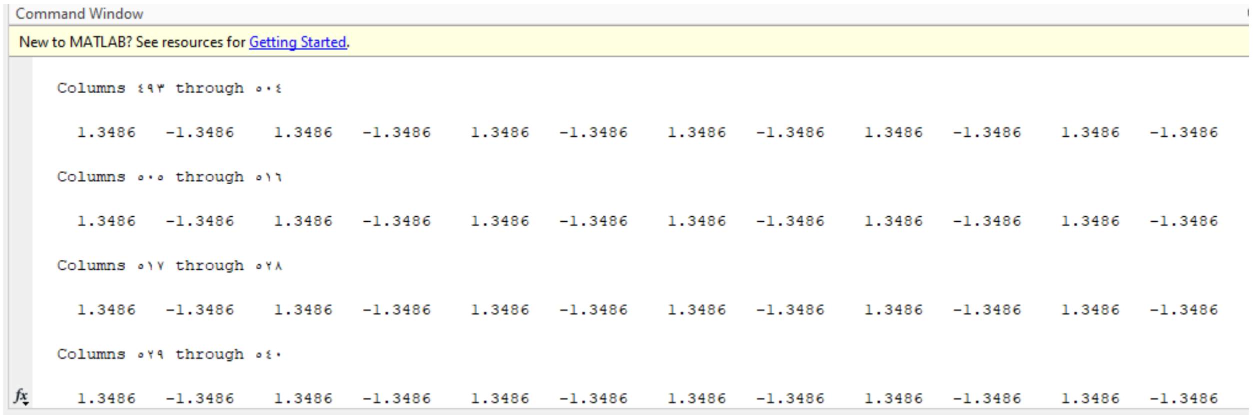
Editor - C:\Users\Asus\OneDrive\Desktop\DSP_Project\d_2.m

```
N=2000;
M = 3; % number of filter coefficients
x = cos(0.03*pi*(0:N-1)'); % input signal
w=[0,0,0,0];
s=[1 -2 4];
mu=0.01;
a=1;
d=filter(s,a,x);

% Create RLS filter object
rls = dsp.RLSFilter(11, 'ForgettingFactor', 0.98);

% Filter the input signal
[y,e] = rls(x,d);

for n = M:N
    x_in = cos(0.03*pi*n); % get the input vector
    y_est = w(n)' * x_in; % estimate the output
    e(n) = d(n) - y_est;
    mu = 1 / (x_in' * x_in);
    w(n+1) = w(n)+ 2*mu*e(n)*x_in;
```



Editor - C:\Users\Asus\OneDrive\Desktop\DSP_Project\d_2.m

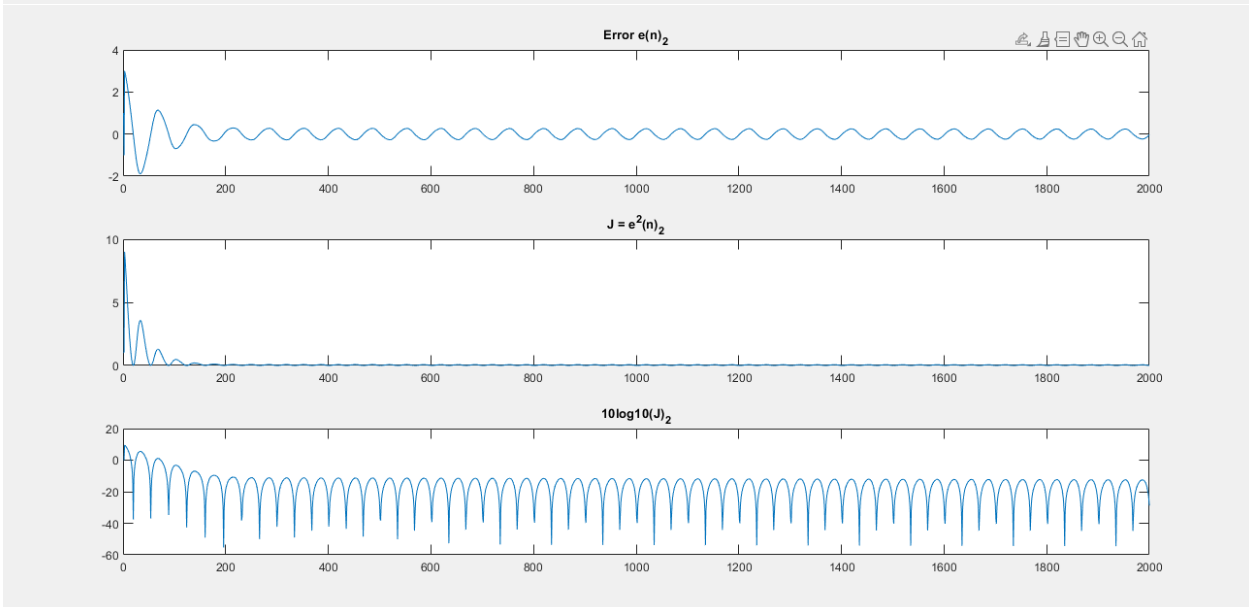
```
    y_est = w(n)' * x_in; % estimate the output
    e(n) = d(n) - y_est;
    mu = 1 / (x_in' * x_in);
    w(n+1) = w(n)+ 2*mu*e(n)*x_in;
    J(n) = e(n)^2;
end
% Plot the learning curves
figure;
subplot(3,1,1);
plot(err);
title('Error e(n)_2');

subplot(3,1,2);
J = err.^2;
plot(J);
title('J = e^2(n)_2');

subplot(3,1,3);
logJ = 10*log10(J);
plot(logJ);
title('10log10(J)_2');
```



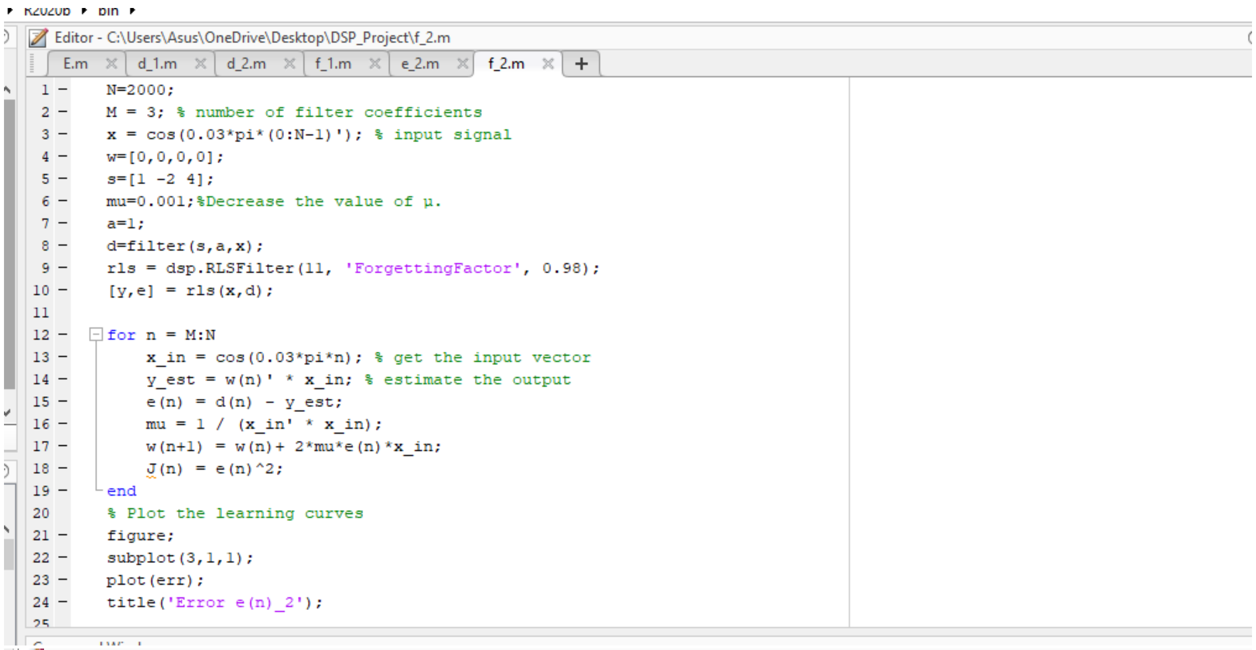
Command Window

New to MATLAB? See resources for Getting Started.

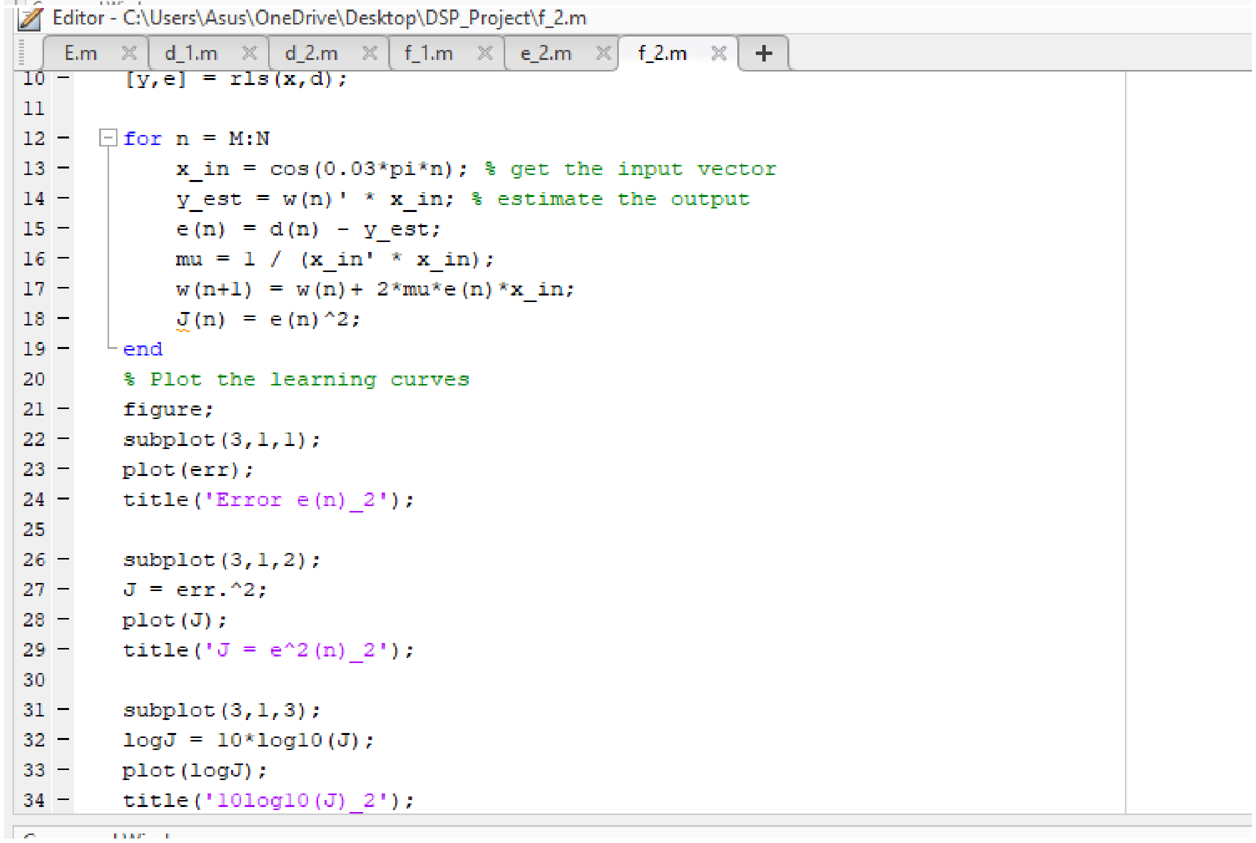
```
  Columns ٤٩٣ through ٥٠٤

    1.3486   -1.3486    1.3486   -1.3486    1.3486   -1.3486    1.3486   -1.3486    1.3486   -1.3486    1.3486   -1.3486

  Columns ٥٠٥ through ٥١٦

    1.3486   -1.3486    1.3486   -1.3486    1.3486   -1.3486    1.3486   -1.3486    1.3486   -1.3486    1.3486   -1.3486

  Columns ٥١٧ through ٥٢٨

    1.3486   -1.3486    1.3486   -1.3486    1.3486   -1.3486    1.3486   -1.3486    1.3486   -1.3486    1.3486   -1.3486

  Columns ٥٢٩ through ٥٤٠

    1.3486   -1.3486    1.3486   -1.3486    1.3486   -1.3486    1.3486   -1.3486    1.3486   -1.3486    1.3486   -1.3486
```



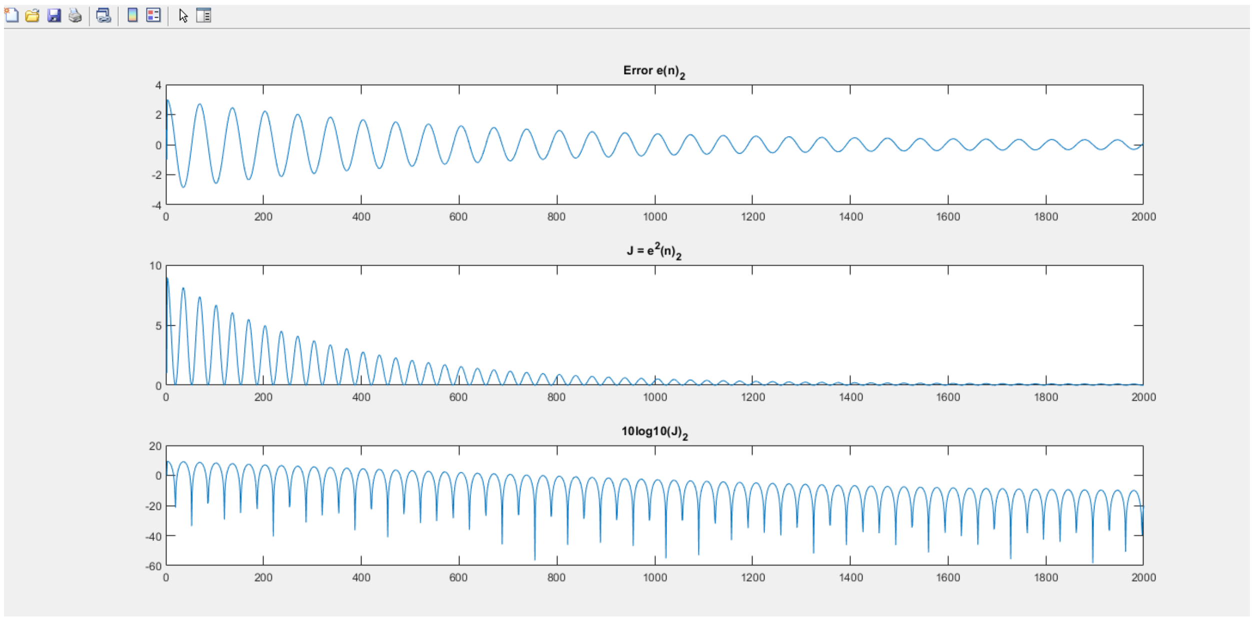
# F2



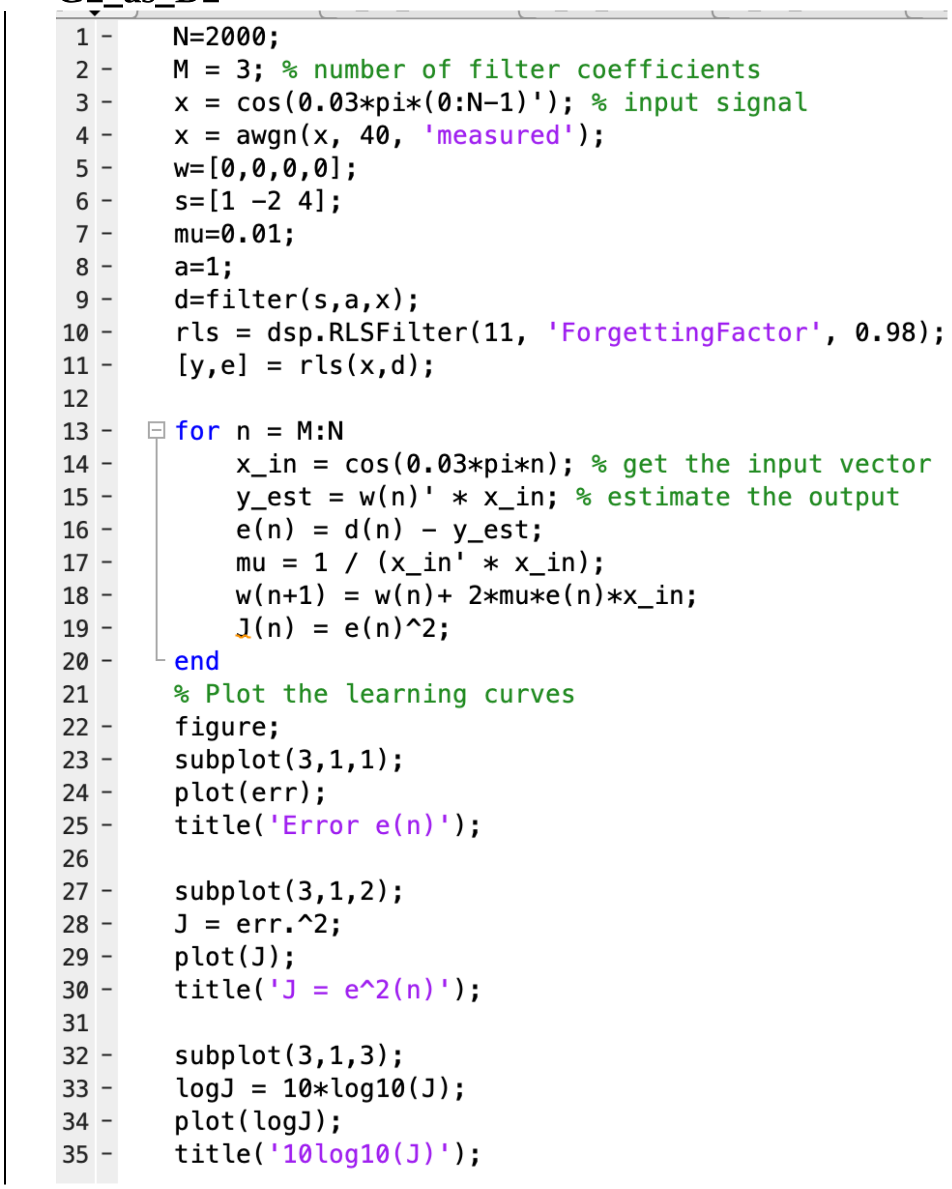
```
N=2000;
M = 3; % number of filter coefficients
x = cos(0.03*pi*(0:N-1)'); % input signal
w=[0,0,0,0];
s=[1 -2 4];
mu=0.001;%Decrease the value of μ.
a=1;
d=filter(s,a,x);
rls = dsp.RLSFilter(11, 'ForgettingFactor', 0.98);
[y,e] = rls(x,d);

for n = M:N
    x_in = cos(0.03*pi*n); % get the input vector
    y_est = w(n)' * x_in; % estimate the output
    e(n) = d(n) - y_est;
    mu = 1 / (x_in' * x_in);
    w(n+1) = w(n)+ 2*mu*e(n)*x_in;
    J(n) = e(n)^2;
end
% Plot the learning curves
figure;
subplot(3,1,1);
plot(err);
title('Error e(n)_2');
```



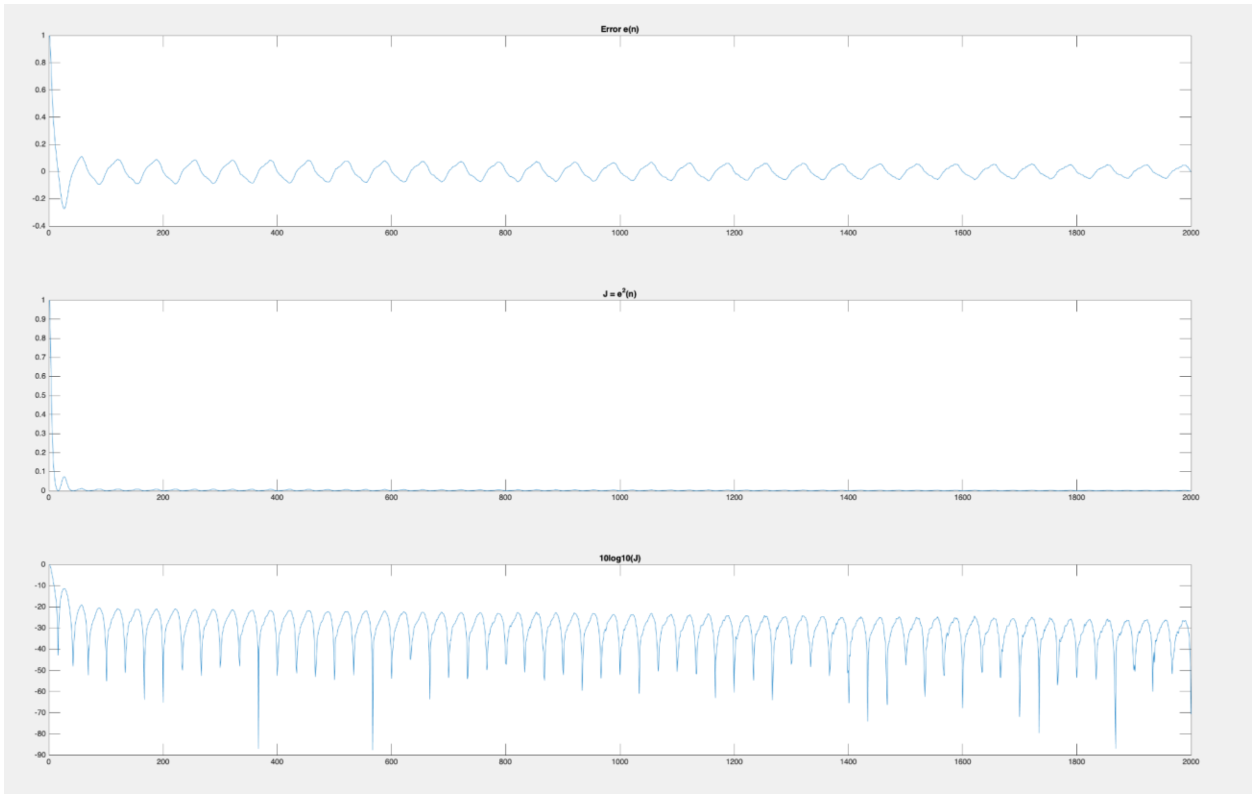
```
[y,e] = rls(x,d);

for n = M:N
    x_in = cos(0.03*pi*n); % get the input vector
    y_est = w(n)' * x_in; % estimate the output
    e(n) = d(n) - y_est;
    mu = 1 / (x_in' * x_in);
    w(n+1) = w(n)+ 2*mu*e(n)*x_in;
    J(n) = e(n)^2;
end
% Plot the learning curves
figure;
subplot(3,1,1);
plot(err);
title('Error e(n)_2');

subplot(3,1,2);
J = err.^2;
plot(J);
title('J = e^2(n)_2');

subplot(3,1,3);
logJ = 10*log10(J);
plot(logJ);
title('10log10(J)_2');
```



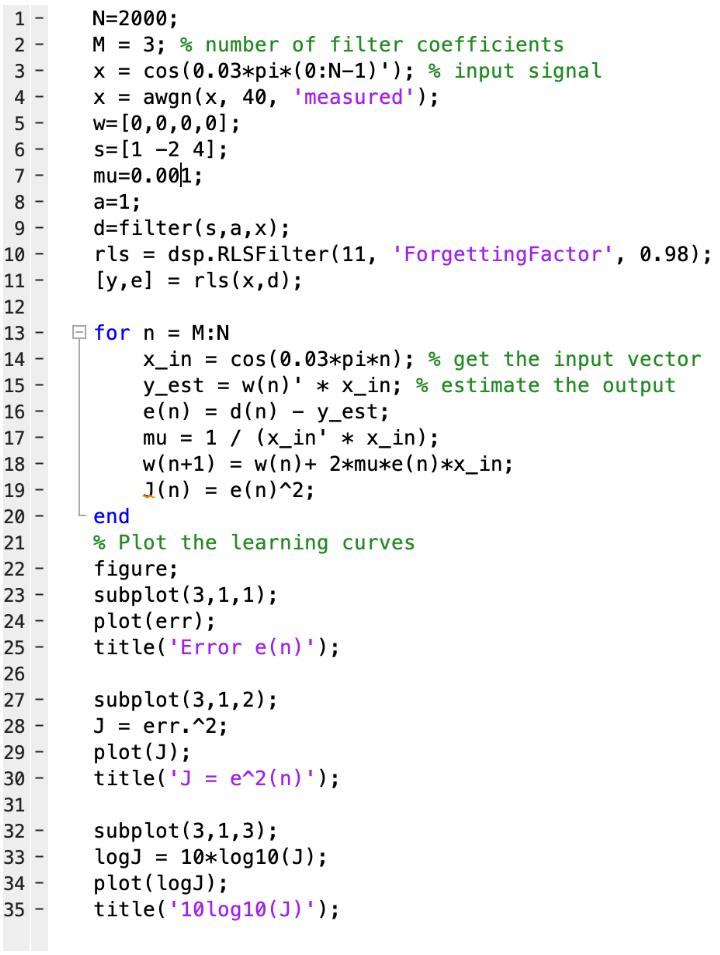
## G2_as_D2



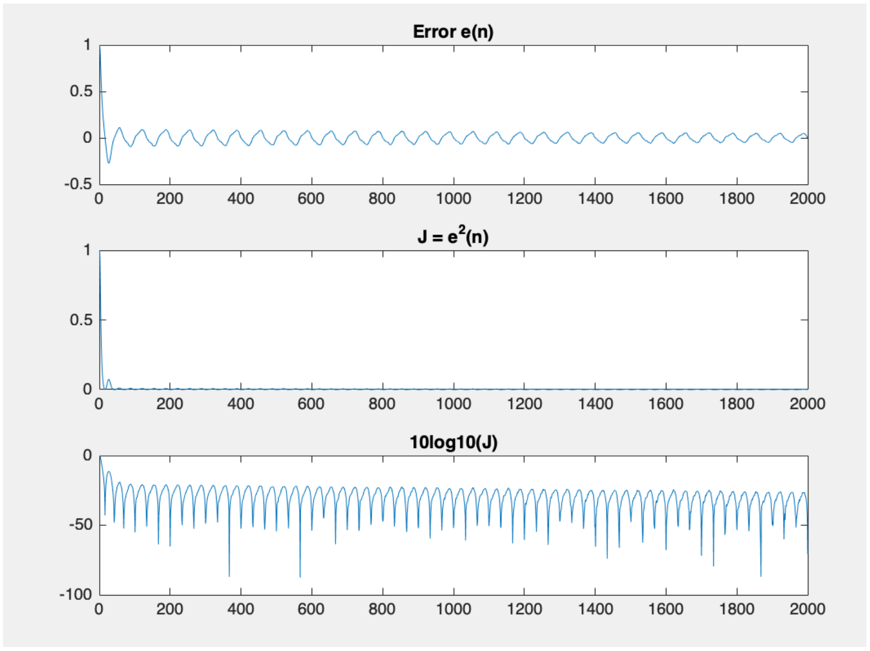
```
N=2000;
M = 3; % number of filter coefficients
x = cos(0.03*pi*(0:N-1)'); % input signal
x = awgn(x, 40, 'measured');
w=[0,0,0,0];
s=[1 -2 4];
mu=0.01;
a=1;
d=filter(s,a,x);
rls = dsp.RLSFilter(11, 'ForgettingFactor', 0.98);
[y,e] = rls(x,d);

for n = M:N
    x_in = cos(0.03*pi*n); % get the input vector
    y_est = w(n)' * x_in; % estimate the output
    e(n) = d(n) - y_est;
    mu = 1 / (x_in' * x_in);
    w(n+1) = w(n)+ 2*mu*e(n)*x_in;
    J(n) = e(n)^2;
end
% Plot the learning curves
figure;
subplot(3,1,1);
plot(err);
title('Error e(n)');

subplot(3,1,2);
J = err.^2;
plot(J);
title('J = e^2(n)');

subplot(3,1,3);
logJ = 10*log10(J);
plot(logJ);
title('10log10(J)');
```


Error e(n)
$J = e^2(n)$
10log10(J)

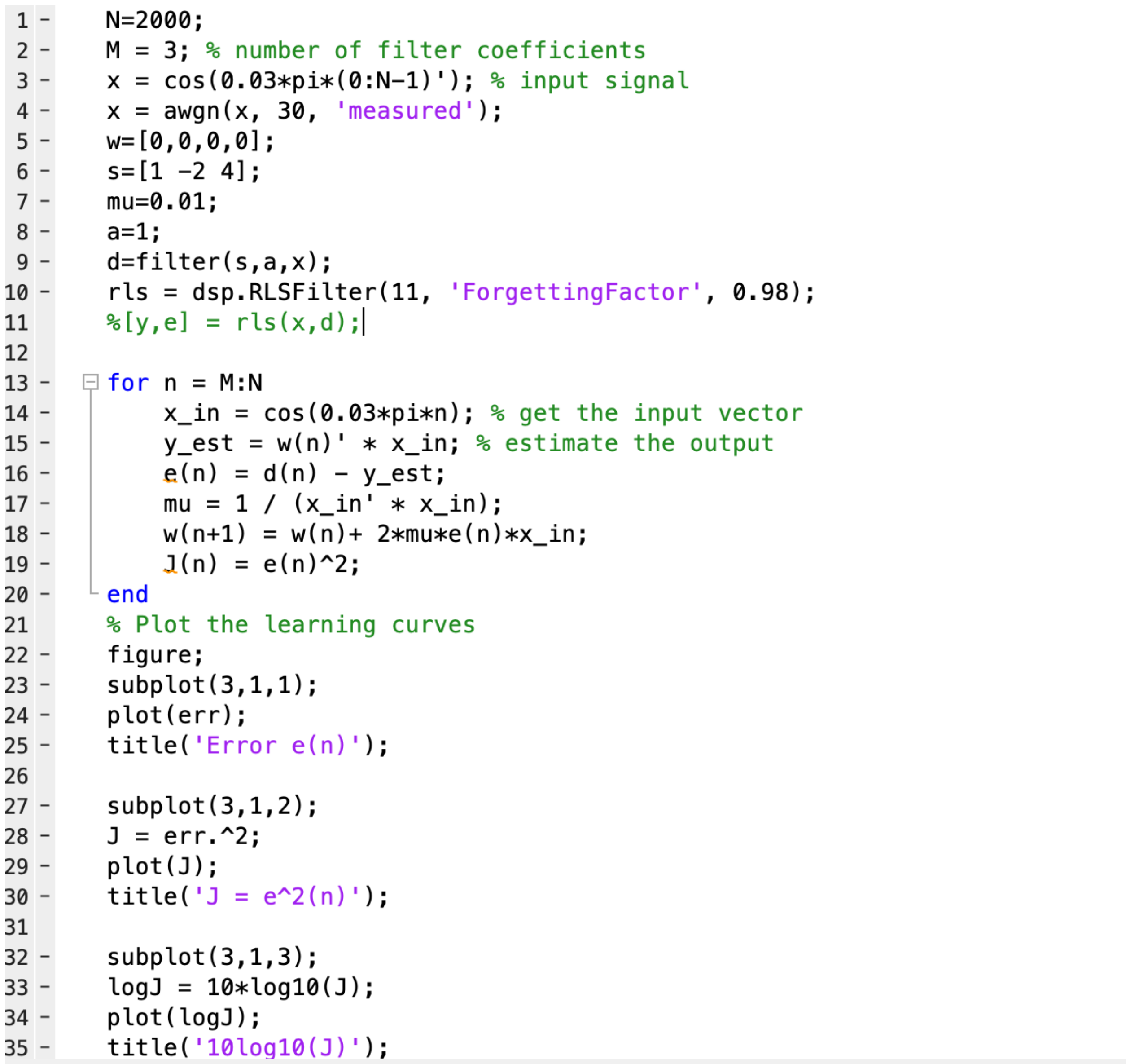
## G2_as_F2



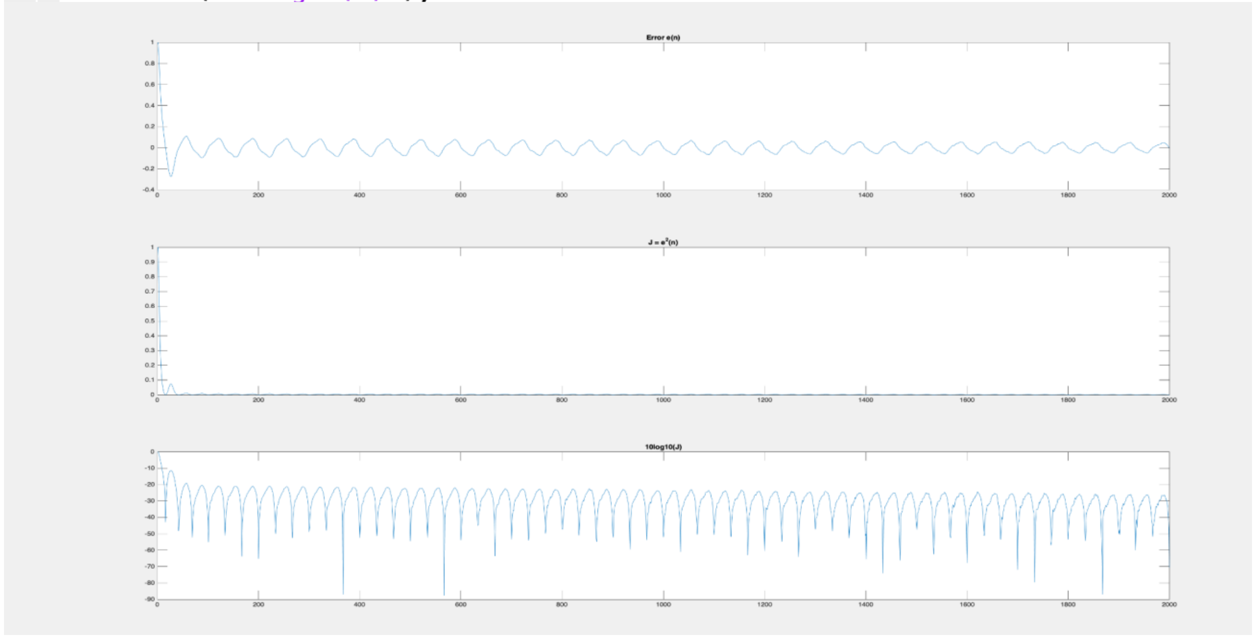
```
N=2000;
M = 3; % number of filter coefficients
x = cos(0.03*pi*(0:N-1)'); % input signal
x = awgn(x, 40, 'measured');
w=[0,0,0,0];
s=[1 -2 4];
mu=0.001;
a=1;
d=filter(s,a,x);
rls = dsp.RLSFilter(11, 'ForgettingFactor', 0.98);
[y,e] = rls(x,d);

for n = M:N
    x_in = cos(0.03*pi*n); % get the input vector
    y_est = w(n)' * x_in; % estimate the output
    e(n) = d(n) - y_est;
    mu = 1 / (x_in' * x_in);
    w(n+1) = w(n)+ 2*mu*e(n)*x_in;
    J(n) = e(n)^2;
end
% Plot the learning curves
figure;
subplot(3,1,1);
plot(err);
title('Error e(n)');

subplot(3,1,2);
J = err.^2;
plot(J);
title('J = e^2(n)');

subplot(3,1,3);
logJ = 10*log10(J);
plot(logJ);
title('10log10(J)');
```



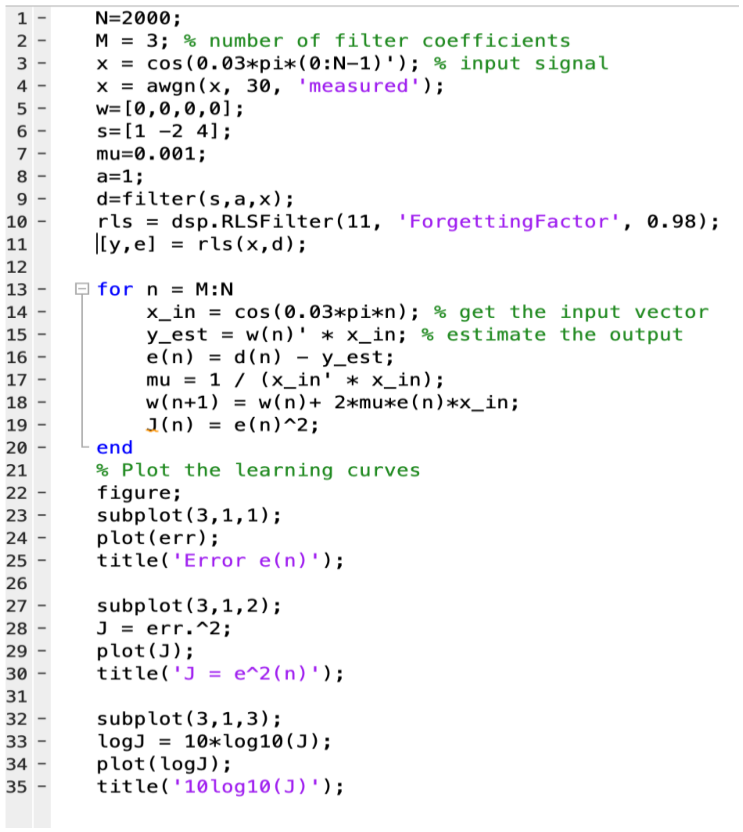
## H2_as_D2



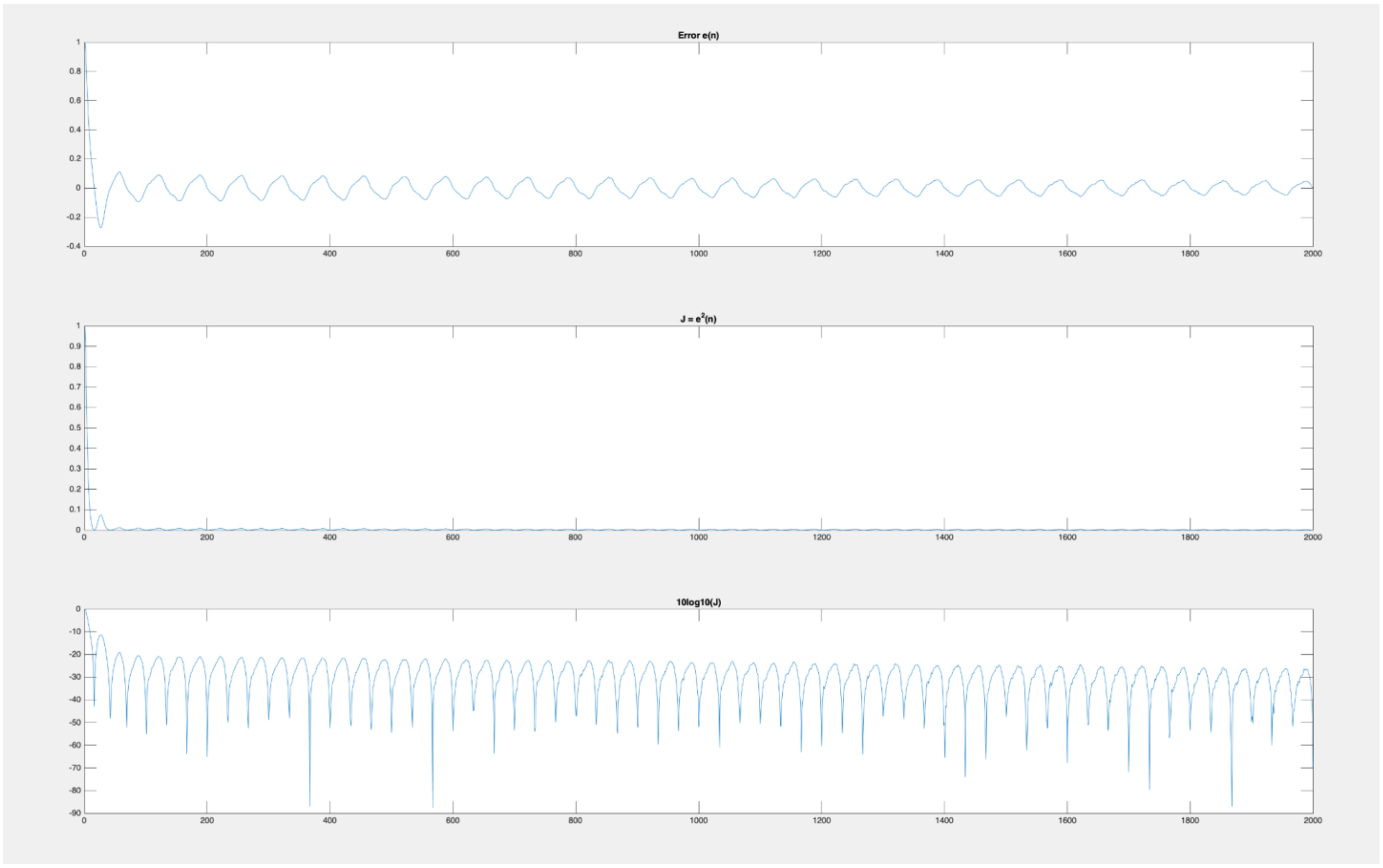
```
N=2000;
M = 3; % number of filter coefficients
x = cos(0.03*pi*(0:N-1)'); % input signal
x = awgn(x, 30, 'measured');
w=[0,0,0,0];
s=[1 -2 4];
mu=0.01;
a=1;
d=filter(s,a,x);
rls = dsp.RLSFilter(11, 'ForgettingFactor', 0.98);
%[y,e] = rls(x,d);

for n = M:N
    x_in = cos(0.03*pi*n); % get the input vector
    y_est = w(n)' * x_in; % estimate the output
    e(n) = d(n) - y_est;
    mu = 1 / (x_in' * x_in);
    w(n+1) = w(n)+ 2*mu*e(n)*x_in;
    J(n) = e(n)^2;
end
% Plot the learning curves
figure;
subplot(3,1,1);
plot(err);
title('Error e(n)');

subplot(3,1,2);
J = err.^2;
plot(J);
title('J = e^2(n)');

subplot(3,1,3);
logJ = 10*log10(J);
plot(logJ);
title('10log10(J)');
```



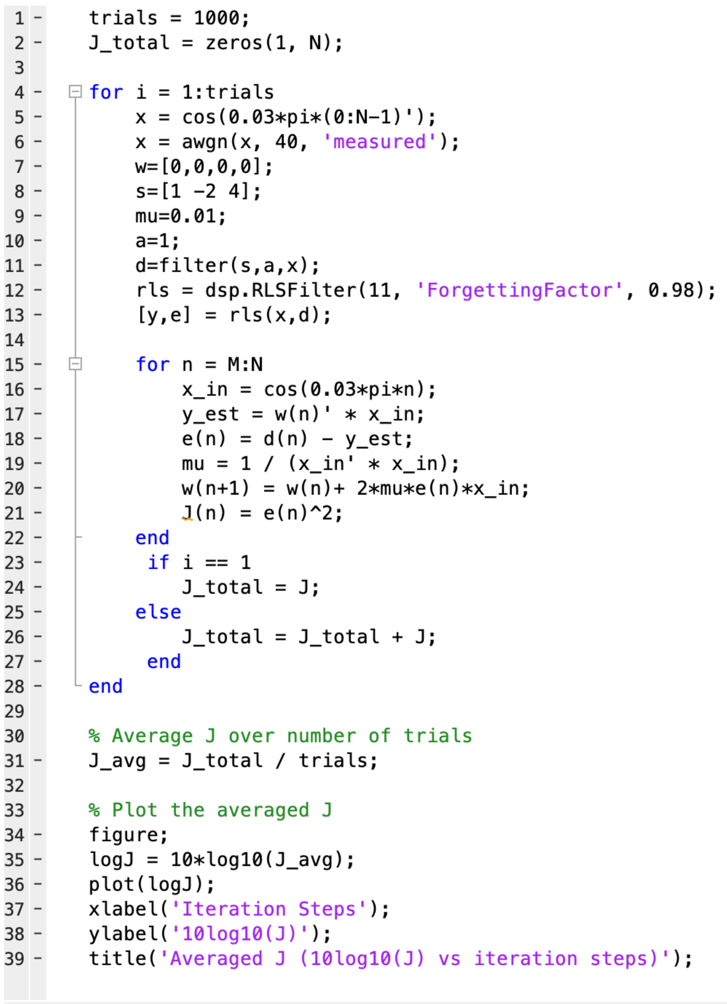
## H2_as_F2



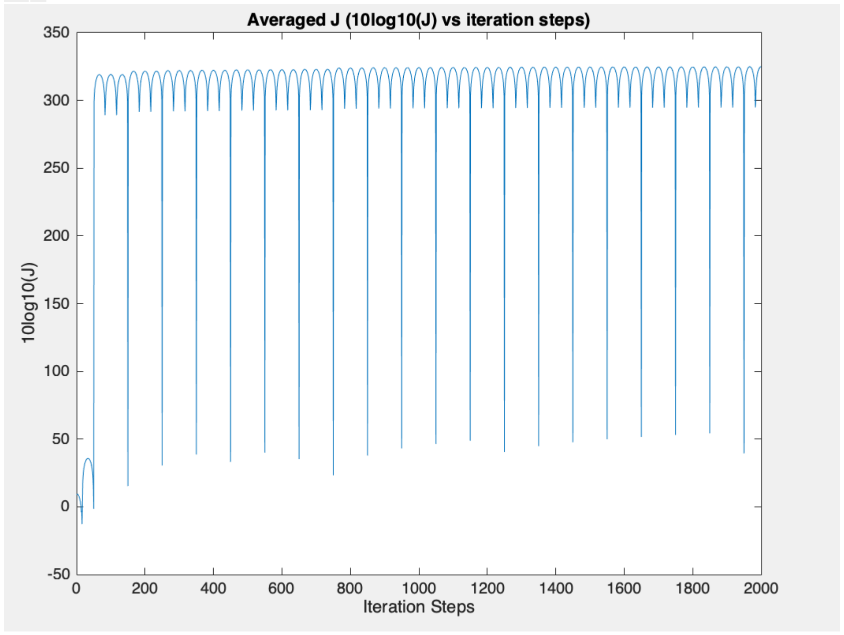
```matlab
N=2000;
M = 3; % number of filter coefficients
x = cos(0.03*pi*(0:N-1)'); % input signal
x = awgn(x, 30, 'measured');
w=[0,0,0,0];
s=[1 -2 4];
mu=0.001;
a=1;
d=filter(s,a,x);
rls = dsp.RLSFilter(11, 'ForgettingFactor', 0.98);
[y,e] = rls(x,d);

for n = M:N
    x_in = cos(0.03*pi*n); % get the input vector
    y_est = w(n)' * x_in; % estimate the output
    e(n) = d(n) - y_est;
    mu = 1 / (x_in' * x_in);
    w(n+1) = w(n)+ 2*mu*e(n)*x_in;
    J(n) = e(n)^2;
end
% Plot the learning curves
figure;
subplot(3,1,1);
plot(err);
title('Error e(n)');

subplot(3,1,2);
J = err.^2;
plot(J);
title('J = e^2(n)');

subplot(3,1,3);
logJ = 10*log10(J);
plot(logJ);
title('10log10(J)');
```



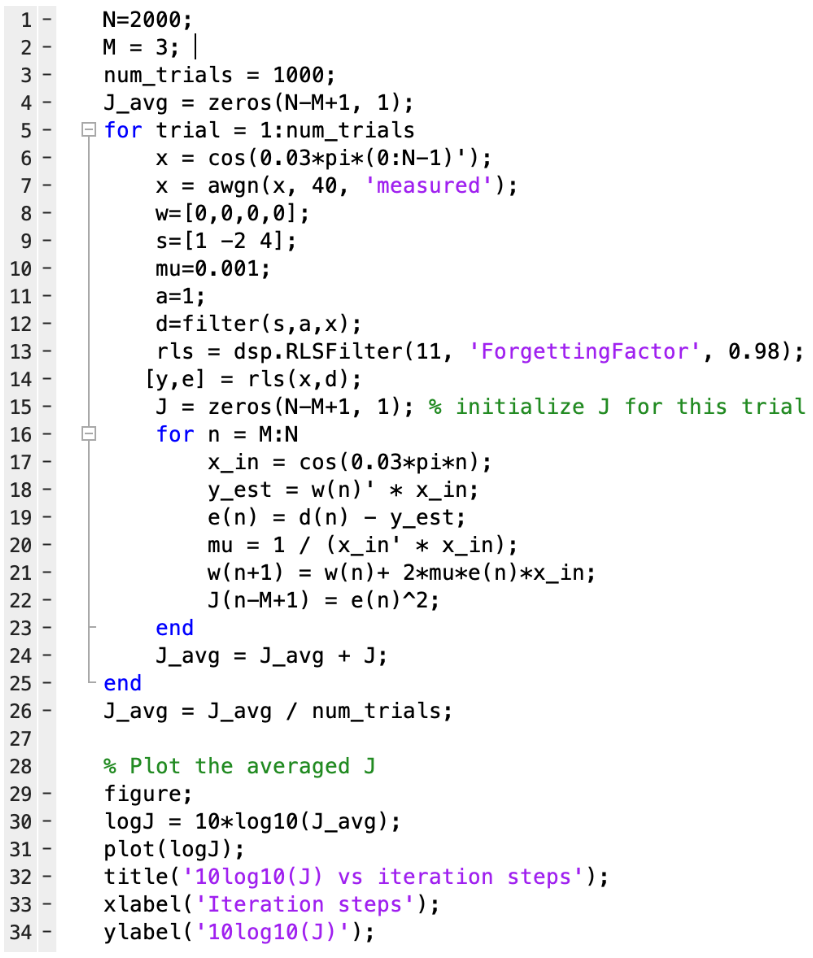
**I2_as_D2**



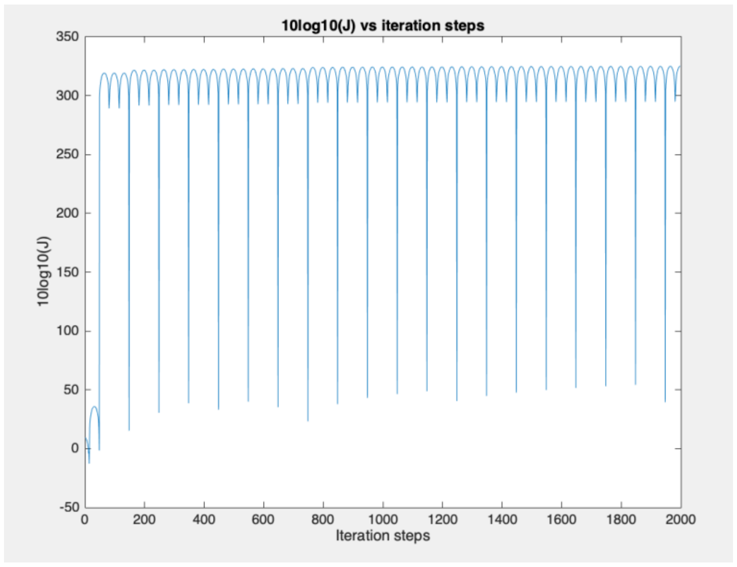
```
trials = 1000;
J_total = zeros(1, N);

for i = 1:trials
    x = cos(0.03*pi*(0:N-1)');
    x = awgn(x, 40, 'measured');
    w=[0,0,0,0];
    s=[1 -2 4];
    mu=0.01;
    a=1;
    d=filter(s,a,x);
    rls = dsp.RLSFilter(11, 'ForgettingFactor', 0.98);
    [y,e] = rls(x,d);

    for n = M:N
        x_in = cos(0.03*pi*n);
        y_est = w(n)' * x_in;
        e(n) = d(n) - y_est;
        mu = 1 / (x_in' * x_in);
        w(n+1) = w(n)+ 2*mu*e(n)*x_in;
        J(n) = e(n)^2;
    end
     if i == 1
        J_total = J;
    else
        J_total = J_total + J;
     end
end

% Average J over number of trials
J_avg = J_total / trials;

% Plot the averaged J
figure;
logJ = 10*log10(J_avg);
plot(logJ);
xlabel('Iteration Steps');
ylabel('10log10(J)');
title('Averaged J (10log10(J) vs iteration steps)');
```



**I2_as_F2**



```matlab
N=2000;
M = 3;
num_trials = 1000;
J_avg = zeros(N-M+1, 1);
for trial = 1:num_trials
  x = cos(0.03*pi*(0:N-1)');
  x = awgn(x, 40, 'measured');
  w=[0,0,0,0];
  s=[1 -2 4];
  mu=0.001;
  a=1;
  d=filter(s,a,x);
  rls = dsp.RLSFilter(11, 'ForgettingFactor', 0.98);
  [y,e] = rls(x,d);
  J = zeros(N-M+1, 1); % initialize J for this trial
  for n = M:N
    x_in = cos(0.03*pi*n);
    y_est = w(n)' * x_in;
    e(n) = d(n) - y_est;
    mu = 1 / (x_in' * x_in);
    w(n+1) = w(n)+ 2*mu*e(n)*x_in;
    J(n-M+1) = e(n)^2;
  end
  J_avg = J_avg + J;
end
J_avg = J_avg / num_trials;

% Plot the averaged J
figure;
logJ = 10*log10(J_avg);
plot(logJ);
title('10log10(J) vs iteration steps');
xlabel('Iteration steps');
ylabel('10log10(J)');
```


10log10(J) vs iteration steps
10log10(J)
Iteration steps