Lab SP2-41: Online parameter estimation

Jack Mackintosh - 539625
Jack Hurley - 539133

May 2017

Section 1

Preamble

The goal of this section is to cancel the echoes in an audio signal. Various cases will be assessed including with noise and time varying echo amplitudes. For this echo cancellation we will implement Least Mean Squares (LMS) and Recursive Least Squares (RLS) algorithms, which follow the form of:

$$\hat{\theta}(t) = \hat{\theta}(t-1) + G(t)(y(t) - \phi^T(t)\hat{\theta}(t-1)) \tag{1}$$

Where:
$y(t)$ is the output signal
$\phi(t)$ is the input signal vector
$\hat{\theta}(t)$ is the estimated parameter vector
$G(t)$ is the gain parameter which for:

RLS:

$$G(t) = P(t)\phi(t) \tag{2}$$

Noting also that P(t) is defined as the Riccati equation which in the vector case is with out a forgetting facor is:

$$P(t) = P(t-1) - \frac{P(t-1)\phi(t)^T\phi(t)P(t-1)}{1 + \phi(t)^T P(t-1)\phi(t)} \tag{3}$$

And with a forgetting factor equal to λ is:

$$P(t) = \frac{1}{\lambda}[P(t-1) - \frac{P(t-1)\phi(t)^T\phi(t)P(t-1)}{\lambda + \phi(t)^T P(t-1)\phi(t)}] \tag{4}$$

LMS:

$$G(t) = \mu\phi(t) \tag{5}$$

In the most simple case we know our signal is the composite of the original signal with two delayed versions of the original signal with constant amplitudes and delays of 1 and 2.5 seconds respectively. Hence we can view our signal of interest as:

$$y(t) = b_1 u(t) + b_2 u(t-1) + b_3 u(t-2.5) \tag{6}$$

Where:
$y(t)$ is the output signal
$u(t)$ is the input signal
b_1, b_2 and b_3 are the amplitudes of the components of the signal and are all constants.

From this conclusion we can define:

$$\phi(t) = \begin{bmatrix} u(t) \\ u(t-1) \\ u(t-2.5) \end{bmatrix} \tag{7}$$

and

$$\theta(t) = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \tag{8}$$

Due to the nature of an echo b_1 value will be taken as the maximum amplitude and be given the value of 1. All echos must be less in amplitude than the original signal, hence

$$0 \leq b_2, b_3 \leq 1$$

One other assumption we make throughout this report is that the loudspeaker begins at the same time as the microphone. This leads to an assumption that there will be no echoes present until after one second, and furher echoes after 2.5 seconds. In other words we will be taking the initial condition to be zero before either echoes have been applied. Different models can be applied to handle this situation in other cases but given our input signal did not have echoes before the 1 second mark we deemed this to be an adequate assumption.

MATLAB Interpretation

To implement RLS and LMS algorithims in MATLAB we iterate through the audio signals and generate estimates for the parameters in $\hat{\theta}(t)$. We do this whilst saving an output vector to be played back and analysed. To generate the matrix ϕ in MATLAB (ϕ defined at 7) we construct this vector from `loudspeaker` at position `[i]`, `[i-Fs*DELAY1]` and `[i-FS*DELAY2]`. If `[i-Fs*DELAYx]` < 0 we

set the value to zero since no echo has been registered at this point.

At each time step, the equation 1 is iterated with the appropriate gain term (G(t)) depending on the algorithm implemented. These will be discussed in their respective sections.

To implement the cancellation of the echoes from the original signal the following function can be applied to the `loudspeaker` vector (u(t) in this case) to produce a non-echoed output signal. This function is shown below:

$$output(t) = y(t) - b_2 u(t-1) - b_3 u(t-2.5)$$

which can be expanded to:

$$output(t) = u(t) + b_2 u(t-1) - b_2 u(t-1) + b_3 u(t-2.5) - b_3 u(t-2.5)$$

Leaving:

$$output(t) = u(t)$$

Which is our desired non-echoed output signal to be played back and sound perfect.

(a) Constant Echo Amplitudes

The goal of this section is to cancel the echoes in our signal when the echos have **constant but unknown amplitude and constant known delays**.

Recursive Least Squares

The RLS algorithm follows the MATLAB setup above, the gain term G(t) for this follows equations 2 and 3. At each time step the Riccati equation can be constructed as defined above which give weight to the error signal as seen in equation 1.

RLS: Initial Conditions

The initial conditions for the algorithms need to be considered before running the process. Since the echoes are of unknown amplitude the choices for b_1, b_2 and b_3 can be left to the choice of the designer. The effect that this decision has on the result will be explored below. The initial value for b_1 can be assumed to be 1 since we expect the microphone to pick up the loudspeaker well in this case. This may not be valid for different a different system but this is an assumption we made. As stated above, the values of these parameters are on the range of $[0:1]$.

Figure 1 indicates that in our system the initial values for the parameters are not overly important. The algorithm appears to converge to the desired values

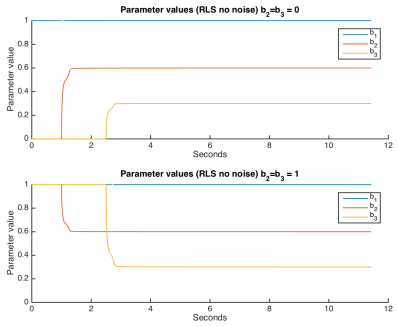

Figure 1: Comparing two different sets of initial parameter values

in a similar period of time regardless of the initial value. In a system with a slower response this initial guess may be more important.

The Riccati equation also needs to be given an initial value. This is typically some multiple of the identity matrix (a 3x3 in this case). Figure 2 shows the effect that different multiples of the identity matrix give to the performance of the estimation.

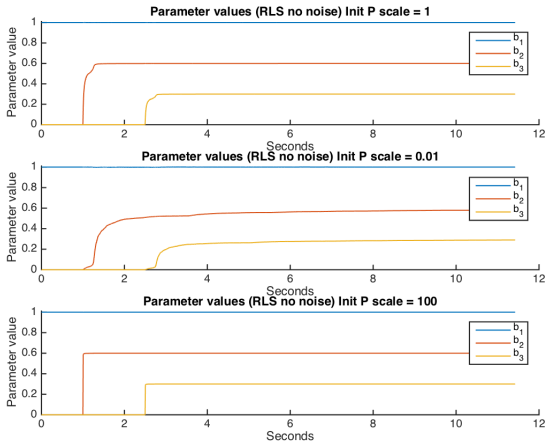

Figure 2: Varying Scalings of Initial P value and its effect on Parameter Values

As can be seen above in Figure 2, the initial scaling of P leads to different performance of the algorithm. The first graph shows the performance of our initial value Pinit = 1. The following two graphs show the performance of initial P values two orders of magnitude below and above the original value.

For $Pinit = 0.01I$ in Figure:2b. the response is considerably slower when compared with Figure:2a). Conversely $Pinit = 100I$ shown in Figure:2c) dis-

plays a rapid approach of the estimation to the system parameters.

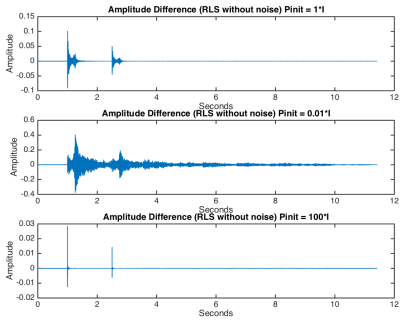

Figure 3: Varying initial P values and its effect on Echo Reduction

Figure 3 shows the difference between the input signal from the `loudspeaker` and the output of the algorithm over time. As can be seen a smaller initial value leads to slower parameter estimation and thus the echoes are not reduced as quickly.

RLS: Noise

We now present the algorithms performance when noise is applied to the system. The characteristics of the noise are unknown yet the parameters remain the same so we can apply the same process as above.

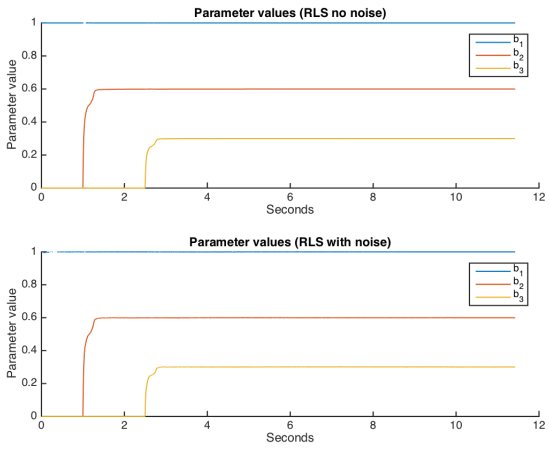

Figure 4: Parameter Estimation under the effect of noise

Figure 4ab) show the change in the estimation of the system parameters

over time under the effect of noise. Figure 4a) depicts the system with no noise applied. Comparing the estimation with that of Figure 4b) shows minimal difference. The algorithm is able to approach the parameter values at a similar rate in this scenario. Figure 5 shows the difference between the loudspeaker output and the microphone inputs both with and without noise. It can be seen from the bottom graph that the system responds slightly slower when noise is present but is still able to remove the echo. When the noisy output is listened to it is clear that the echo has been removed, whilst the noise remains. See the appendix below for the code to implement this.

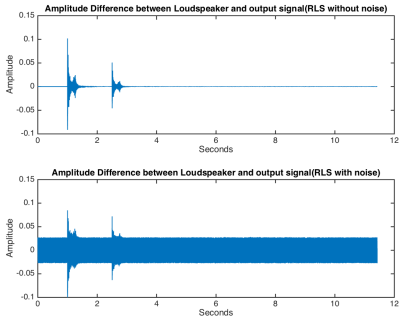

Figure 5: RLS: Echo Reduction under the effect of noise

Least Mean Squares

To implement LMS in MATLAB we again followed the MATLAB interpretation described above, the gain term in the LMS case dictated by μ and can be seen in equation 5.

LMS: Initial Conditions

Similarly to the RLS case we can apply different initial guesses to the parameters to see how the algorithm responds. In Figure 6 we see that there is little change to the response time when varying the initial guess from 0 to 1.

The choice of μ is important in the performance of the LMS algorithm. As stated in 5 the μ scales the error signal and acts as the step size of the algorithm. If the step size is small the algorithm takes a longer time to approach the ideal values, whilst if the step size is too large the algorithm may not converge close enough to the system parameters and can be seen to 'jump' across the value.

The value for μ should be chosen to satisfy the following equation:

$$0 < \mu \ll \frac{1}{\mathbb{E}[||\phi(k)||^2]}$$

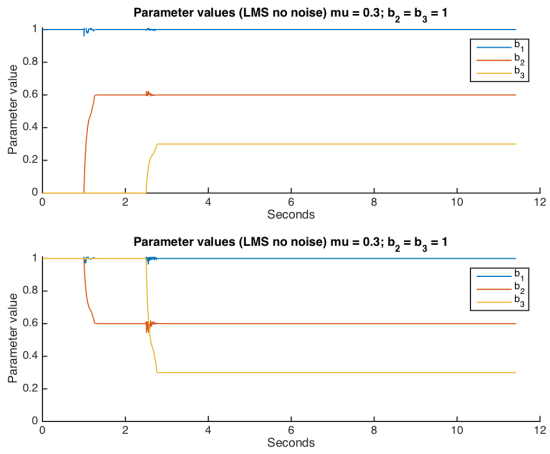

Figure 6: LMS: Response due to differing initial parameters

Using this we found that μ should be much less than ≈ 22. Using this we have generated three plots for μ values of 0.3, 0.03 and 3.

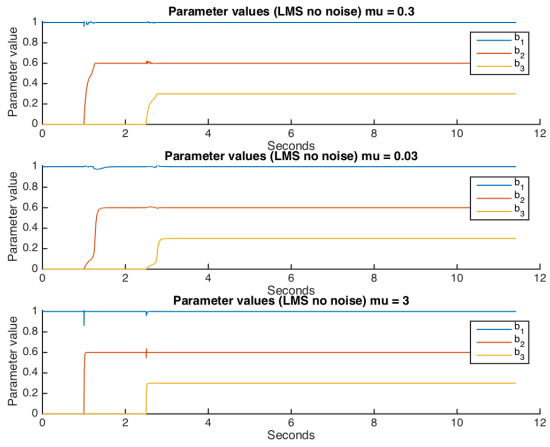

Figure 7: LMS: Response due to changing mu values

As expected the graphs with the lower μ values converge to the expected system parameter values at a slower rate since the step size is smaller. When we drove μ much higher than in Figure 7c) ($\mu = 3$) the estimations became unusable and jumped around a lot. We found 3 to be close to the upper bound of what we deemed acceptable.

LMS: Noise

Similarly to the RLS case, the presence of noise has an effect on the algorithm's performance but the parameters can still be estimated. Comparing Figure 8 with Figure 4 it can be seen that LMS appears to fluctuate more under the influence of noise than in the RLS case.

Figure 9 shows that the echo dies away at a similar rate in both the noise and noise-free systems, and the audio output agrees with this.

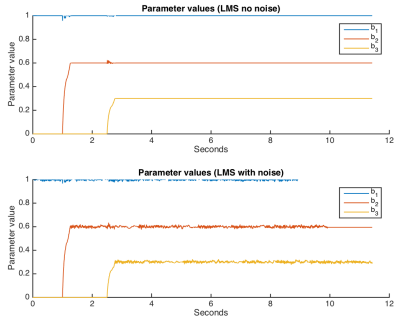

Figure 8: LMS: Parameter estimation under the effect of noise

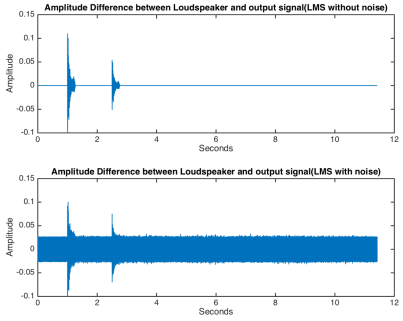

Figure 9: LMS: Response due to changing mu values with noise

Comparing RLS and LMS

Comparing the two implemented algorithms we firstly notice how much less computationally complex LMS is. This is due to its only reliance being on the last time step value, the current input value and step size (μ). This does however lead it to being far less robust against noise than RLS which in this case handles noise extremely well due to its reliance on past values.

(b) Time Varying Echo Amplitudes

The goal of this section is to cancel the echoes in our signal when the echos have **time varying unknown amplitude and constant known delays**. For this echo cancellation we will again implement LMS and RLS algorithms. The set up will be the same as that described in the previous section except we now define:

$$\theta(t) = \begin{bmatrix} b_1(t) \\ b_2(t) \\ b_3(t) \end{bmatrix}$$

In terms of our MATLAB setup the only major differences we need to implement are we now have a different input signal and in the RLS case we now implement a Riccati equation with a forgetting factor, as seen in equation 4.

Recursive Least Squares with Varying echo amplitudes

As stated above we now implement a forgetting factor in our Riccati equation, unless stated otherwise we used a $\lambda = 0.995$ for our simulations as it seemed to consistently show us reasonable performance.

RLS: Varying λ

In this section we examine the effect that difference in λ makes. We can examine figure 10, we clearly see our estimation of the b parameters no longer settles to constants like in the previous section but now changes in discrete steps. These amplitude changes appear to start post 6 seconds (excluding amplitude change from zero when the echos start).

λ is extremely sensitive in this setup and we can see dramatic changes in the signal output depending on what value we choose. As lambda dictates our forgetting factor it imposes the speed at which we close in on a new parameter estimation, again looking at figure 10 we see a $\lambda = 0.9999$ parameter estimate moves much more slowly than the $\lambda = 0.990$ or $\lambda = 0.995$ cases.

The forgetting factor is required for this time varying case as without it estimation values from far in the past are given weight. The result of a low forgetting factor can be seen in the $\lambda = 0.9999$ case of figure 11. Clearly the changes in the parameter values post 6 seconds cause unwanted signal to be present. This occurs because the algorithm is no longer correctly estimating $\hat{\theta}$. It has attached a strong weighting to many samples with old parameter values and as such causes an undesirable output. We see these same amplitude changes quickly cut out in the $\lambda = 0.990$ and $\lambda = 0.995$ cases.

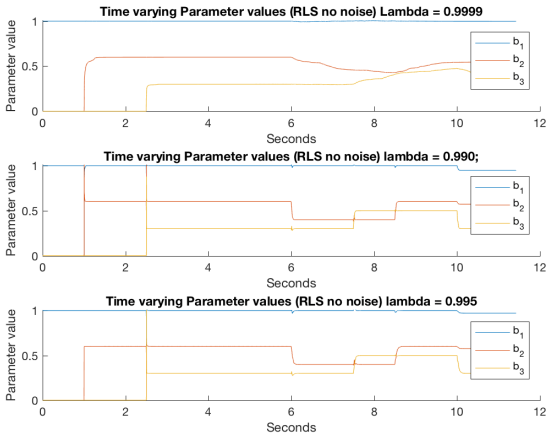

Figure 10: The estimation of b values using RLS and varying Lambda

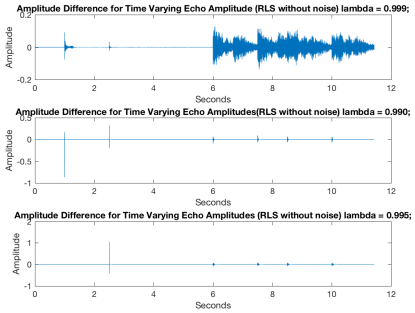

Figure 11: RLS output difference for varying λ values

RLS: Varying echo amplitudes with noise

Adding noise to our setup makes a small difference to our parameter estimation with a $\lambda = 0.995$ as seen in figure 12. Adding a forgetting factor to the RLS algorithm does make it less robust against noise. In the case without a forgetting factor if enough samples are taken even a relatively large deviation due to noise for the input is countered by a long history of unbiased noise is expected to have zero mean. However with a forgetting factor we are dropping past information and hence have less history to counter weight a noisy input value. However with a reasonable λ selected we will still have enough weight on previous samples to ensure noise doesn't distort our estimates.

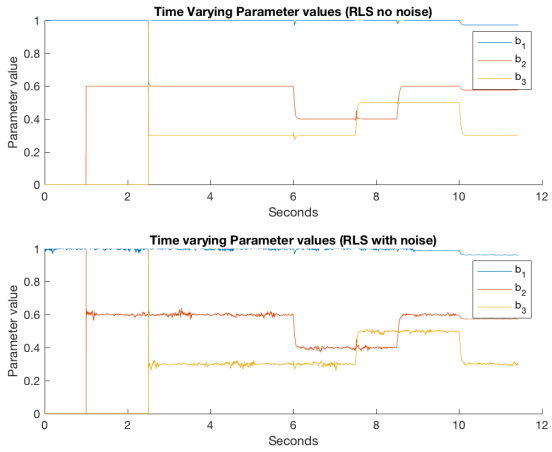

Figure 12: The estimation of b values using RLS and varying Lambda with noise present

LMS: Time Varying Echo Amplitudes

We now inspect how LMS responds to time varying parameter values. We begin by inspecting the effect that varying μ values have on the performance of the algorithm.

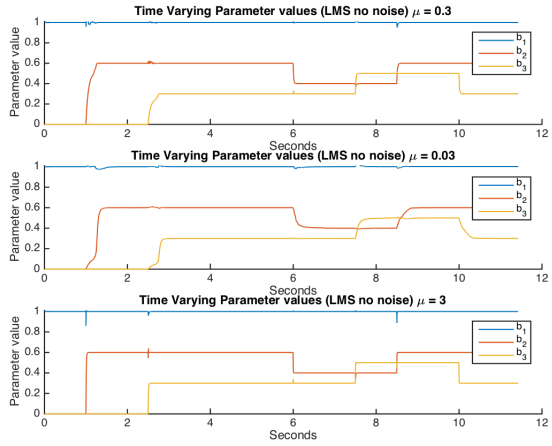

Figure 13: LMS: Effect of varying μ on a time varying parameter system

Assessing Figure 13 we see that the system is able to respond effectively to changes to the system parameters. The rise times of changes appear similar to

those found in the previous section in figure 7. Since LMS is inherently based on the previous sample there is no need for a forgetting factor since it cannot go to sleep. Once again we found that μ values that were too high caused unwanted behaviour such as severe jumping in the estimations.

LMS: Time Varying System With Noise

We now assess the performance of LMS on a noisy system with time varying parameters.

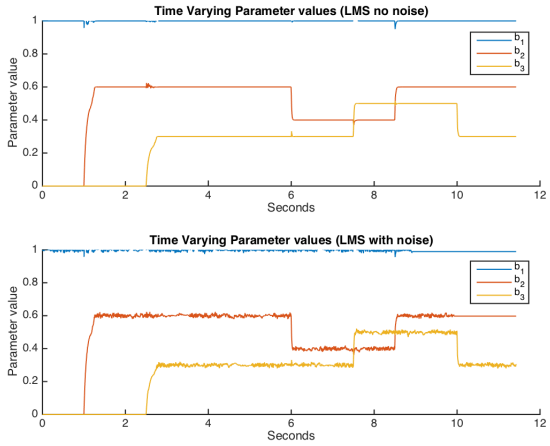

Figure 14: LMS: Effect of varying μ on a time varying parameter system

nce again the performance is very similar to the noisy system with non-time varying parameters. The estimation is able to reach a reasonably consistent value for the parameters in a similar time to the non-noisy case, however it is susceptible to small deviations caused by the noise. This final point can be seen in the fluctuations in the settled signal.

Comparing RLS and LMS in Time Varying Systems

Since LMS depends only upon the previous timestep a time varying system does not effect its performance and it will continue to take the step of greatest descent. RLS must be implemented with a forgetting factor for time varying systems and thus adds another design parameter. Once again RLS is more robust to noise, however the choice of λ will define this robustness.

(c) Time Varying Echo Amplitudes with Delay Uncertainty

The goal of this section is to cancel the echoes in our signal when **the echos time delays are only known within $\pm$ 1 millisecond**.

Since the delays are only known within 1 millisecond of the previously mentioned values we have another uncertainty in our model. However we can model this as a probabilistic parameter. We model the delay time as a uniform distribution over +- 0.001s.

Since a uniform distribution takes its average value at the center of the distribution we determine the maximum likelihood of the delay to be right on the 1s and 2.5s mark and thus the algorithm design is the same as in previous sections however the performance will be decreased. Since the input signal we were given does not have these uncertainties in the delay the results and graphs are also the same.

Section 2: DSP Implementation

For this part a similar problem of canceling an unwanted part of a signal is proposed, in this case they are two unrelated music signals instead of echoed versions of the same signal. This time the processing will be done on the DSP board in real time, hence we will be required to implement our algorithms in C code. Again we will implement RLS and LMS algorithms described by in equations 1, 2, 4 and 5.

All other parameters are the same as before.

Prelab

1

LMS psuedo code:

```
mu = some_value_to_be_calculated;
error = output - (input * theta_hat);
theta_hat = theta_hat + (mu * input) * error;
gain = theta_hat;
```

RLS psuedo code:

```
lambda = some_value_to_be_calculated;
error = output - (input * theta_hat);
calculations_for_each_riccati_matrix_element;
theta_hat = theta_hat + (riccati_matrix_elements * input) * error;
gain = theta_hat;
```

2

To reduce the multiplications and divisions per iteration in the RLS case two major steps will be taken. Firstly any factors multiplied more than once will be

taken out of the calculations and saved as variables beforehand, reducing total calculations. Secondly $\frac{1}{\lambda}$ will be calculated once and saved as a variable.
For example:

inv_lambda = 1.0 / lambda;
p11_in1 = p11 * in1;
p11_new = inv_lambda * (p11 - (p11_in1 ...other_multiplications

3

To make the signal $u_2(t)$ disappear after 5 seconds instead of two we would need to adjust μ and λ to make our parameter predictor value approach the real parameter values more slowly.

In the case of LMS this is relatively simple, μ dictates the step size we take towards finding the minimum value in our cost function, hence a smaller μ results in smaller steps and a slower convergence to our cost functions minimum value (which when reached makes the unwanted audio disappear). The handout indicates that μ should be selected as such:

$$0 < \mu \ll \frac{1}{\mathbb{E}[||\phi(k)||^2]}$$

Hence to find the square expectation of μ we looked at the magnitude of the audio signals coming in (u_1 and u_2) found they had an approximate upper bound of 10^7 this squared gave us $\mu \approx \frac{1}{10^{14}}$ which gave us an upper bound for our decision, however we were likely going to choose one a few orders of magnitude smaller.

In the RLS case λ dictates our functions forgetting factor. Therefore we wish our parameter prediction to forget more slowly (making the music change in 5 seconds not 2). To achieve this we would increase our forgetting factor (i.e increase λ to place more weight onto previous values. This would move us to the new music more slowly.

Lab tasks

Least Mean Squares

The parameter estimation algorithm was implemented on the DSP board (see appendix for code details). The theta history (parameter estimate) was plotted and is shown in figure 15. In this plot a $\mu = 10^{-20}$ was used and it can be seen that over the 10 second sample our predicted θ values approach 0.9 and 0.1. By ear this μ value got the audio to fade out at approximately 5 seconds, which can be seen in figure 15, where the theta values are within 10% of their expected final values at the 5 second mark.

A larger μ value would make our theta values approach their final values more quickly. However if too large a μ is selected there is a chance of overshooting the minimum value of the cost function and oscillating about the minimum point (visualised as overstepping the bottom of a bowl and up the other side).

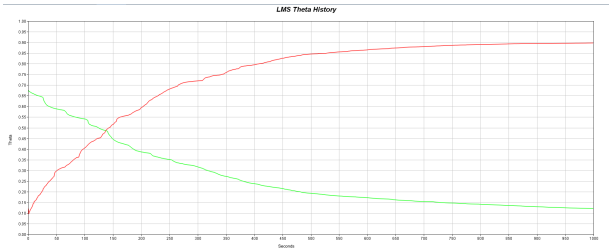

Figure 15: DSP Board LMS Theta History for $\mu = 10^{-20}$

Recursive Least Squares

The parameter estimation algorithm was implemented on the DSP board (see appendix for code details). The theta history (parameter estimate) was plotted and is shown in figure 16. In this plot $\lambda = 0.9999$ which gives a low forgetting factor (A high amount of value is placed on previous values).

This forgetting factor combined with a high sampling frequency allows the music to fade at a noticeable rate. In our experiments a $\lambda < 0.9999$ had our theta values move to their final values of 0.9 and 0.1 almost immediately, as very little value was placed on previous theta values. The practical output of this was that the music sounded like it was just switching from the PC input to the phone without any fade, whereas with $\lambda = 0.9999$ there was an distinguishable fade.

When a $\lambda = 1$ we place an equal amount of value on our error function as we do our new input value. The result of this was in a practical sense that our output became fixed on one of the audio signals and cut out the other, the result of this was the music from only one of the players being able to be heard.

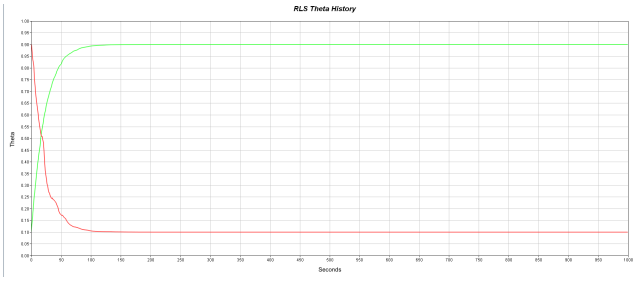

Figure 16: DSP Board RLS Theta History for $\lambda = 0.9999$

A-Appendix

A-Workshop1A

```
%% A.1
% RLS without noise

clear all; close all; clc
load('data1.mat');

% Constants
Fs = 8192; %sampling freq
D1 = 1 * Fs; % delay 1
D2 = 2.5 * Fs; % delay 2
timespan = [0: 1/Fs: length(loudspeaker)/Fs - 1/Fs]; % Timespan for time-axis

% RLS with inital params = 0

init_params = [1; 0; 0];
P_init_scale = 1;

% ---------------------------------- %
%     RLS: Different initial conditions.
% ---------------------------------- %

% RLS with initial params = 0

[output, theta_hat] = RLS_function(loudspeaker, mike1, init_params, P_init_scale);

figure(1)
subplot(2,1,1)
hold on
plot(timespan,theta_hat(1,:))
plot(timespan,theta_hat(2,:))
plot(timespan, theta_hat(3,:))
hold off

title('Parameter values (RLS no noise) b_2=b_3 = 0')
xlabel('Seconds')
ylabel('Parameter value')
legend('b_1', 'b_2', 'b_3')
ylim([0 1])


% RLS with inital params = 1

init_params = [1; 1; 1];
P_init_scale = 1;

[output, theta_hat] = RLS_function(loudspeaker, mike1, init_params, P_init_scale);


subplot(2,1,2)
hold on
plot(timespan,theta_hat(1,:))
plot(timespan,theta_hat(2,:))
plot(timespan, theta_hat(3,:))
hold off

title('Parameter values (RLS no noise) b_2=b_3 = 1')
xlabel('Seconds')
ylabel('Parameter value')
legend('b_1', 'b_2', 'b_3')
ylim([0 1])

saveas(gcf, 'figures/q1a_params.png')
```

```
63
64  %%
65  % ---------------------------------- %
66  %     RLS: P scaling
67  % ---------------------------------- %
68  clear all
69  close all
70  clc
71  load('data1.mat');
72
73  Fs = 8192; %sampling freq
74  D1 = 1 * Fs; % delay 1
75  D2 = 2.5 * Fs; % delay 2
76  timespan = [0: 1/Fs: length(loudspeaker)/Fs - 1/Fs]; % Timespan for time-axis
77
78
79  % RLS with P_init_scale = 1
80  init_params = [1; 0; 0];
81  P_init_scale = 1;
82
83  [output1, theta_hat] = RLS_function(loudspeaker, mike1, init_params, P_init_scale);
84
85  figure(2)
86  subplot(3,1,1)
87  hold on
88  plot(timespan,theta_hat(1,:))
89  plot(timespan,theta_hat(2,:))
90  plot(timespan, theta_hat(3,:))
91  hold off
92
93  title('Parameter values (RLS no noise) Init P scale = 1')
94  xlabel('Seconds')
95  ylabel('Parameter value')
96  legend('b_1', 'b_2', 'b_3')
97  ylim([0 1.01])
98
99
100 % RLS with P_init_scale = 0.01
101 init_params = [1; 0; 0];
102 P_init_scale = 0.01;
103
104 [output001, theta_hat] = RLS_function(loudspeaker, mike1, init_params, P_init_scale);
105
106 figure(2)
107 subplot(3,1,2)
108 hold on
109 plot(timespan,theta_hat(1,:))
110 plot(timespan,theta_hat(2,:))
111 plot(timespan, theta_hat(3,:))
112 hold off
113
114 title('Parameter values (RLS no noise) Init P scale = 0.01')
115 xlabel('Seconds')
116 ylabel('Parameter value')
117 legend('b_1', 'b_2', 'b_3')
118 ylim([0 1.01])
119
120 % RLS with P_init_scale = 100
121 init_params = [1; 0; 0];
122 P_init_scale = 100;
123
124 [output100, theta_hat] = RLS_function(loudspeaker, mike1, init_params, P_init_scale);
125
126 figure(2)
127 subplot(3,1,3)
128 hold on
129 plot(timespan,theta_hat(1,:))
130 plot(timespan,theta_hat(2,:))
```

```
plot(timespan, theta_hat(3,:))
hold off

title('Parameter values (RLS no noise) Init P scale = 100')
xlabel('Seconds')
ylabel('Parameter value')
legend('b_1', 'b_2', 'b_3')
ylim([0 1.01])

saveas(gcf, 'figures/q1a_Pinit.png')

figure(6)
subplot(3,1,1)
plot(timespan, transpose(loudspeaker) - output1)
title('Amplitude Difference (RLS without noise) Pinit = 1*I')
xlabel('Seconds')
ylabel('Amplitude')

subplot(3,1,2)
plot(timespan, transpose(loudspeaker) - output001)
title('Amplitude Difference (RLS without noise) Pinit = 0.01*I')
xlabel('Seconds')
ylabel('Amplitude')

subplot(3,1,3)
plot(timespan, transpose(loudspeaker) - output100)
title('Amplitude Difference (RLS without noise) Pinit = 100*I')
xlabel('Seconds')
ylabel('Amplitude')

saveas(gcf, 'figures/q1a_Pinit_output_diff.png')

%%
% ---------------------------------- %
%     RLS: With Noise
% ---------------------------------- %

clear all
close all
clc
load('data1.mat');

Fs = 8192; %sampling freq
D1 = 1 * Fs; % delay 1
D2 = 2.5 * Fs; % delay 2
timespan = [0: 1/Fs: length(loudspeaker)/Fs - 1/Fs]; % Timespan for time-axis

init_params = [1; 0; 0];
P_init_scale = 1;

[output, theta_hat] = RLS_function(loudspeaker, mike1, init_params, P_init_scale);
[noisy_output, noisy_theta_hat] = RLS_function(loudspeaker, noisymike1, init_params, ...
    P_init_scale);

figure(4)
subplot(2,1,1)
hold on
plot(timespan,theta_hat(1,:))
plot(timespan,theta_hat(2,:))
plot(timespan, theta_hat(3,:))
hold off

title('Parameter values (RLS no noise)')
xlabel('Seconds')
ylabel('Parameter value')
legend('b_1', 'b_2', 'b_3')
ylim([0 1])

```

```
subplot(2,1,2)
hold on
plot(timespan,noisy_theta_hat(1,:))
plot(timespan,noisy_theta_hat(2,:))
plot(timespan, noisy_theta_hat(3,:))
hold off

title('Parameter values (RLS with noise)')
xlabel('Seconds')
ylabel('Parameter value')
legend('b_1', 'b_2', 'b_3')
ylim([0 1])

saveas(gcf, 'figures/q1a_noise_params.png')

% ---------------------------------- %
%     RLS: With noise, comparing output
% ---------------------------------- %

figure(5)
subplot(2,1,1)
plot(timespan, transpose(loudspeaker) - output)
title('Amplitude Difference between Loudspeaker and output signal(RLS without noise)')
xlabel('Seconds')
ylabel('Amplitude')
ylim([-0.1 0.15])
subplot(2,1,2)
plot(timespan, transpose(loudspeaker) - noisy_output)
title('Amplitude Difference between Loudspeaker and output signal(RLS with noise)')
xlabel('Seconds')
ylabel('Amplitude')
ylim([-0.1 0.15])

saveas(gcf, 'figures/q1a_noise_diff.png')



%%
% ---------------------------------- %
%     LMS: No Noise
% ---------------------------------- %

clear all
clc
close all
load('data1.mat');

Fs = 8192; %sampling freq
D1 = 1 * Fs; % delay 1
D2 = 2.5 * Fs; % delay 2
timespan = [0: 1/Fs: length(loudspeaker)/Fs - 1/Fs]; % Timespan for time-axis

% LMS no noise with mu = 3 b2=b3 = 0
init_params = [1; 0; 0];
mu = 0.3;
[output, theta_hat] = LMS_function(loudspeaker, mike1, init_params, mu);

% LMS no noise with mu = 3 b2=b3 = 1
init_params = [1; 1; 1];
[output2, theta_hat2] = LMS_function(loudspeaker, mike1, init_params, mu);


figure(2)
subplot(2,1,1)
hold on
plot(timespan,theta_hat(1,:))
plot(timespan,theta_hat(2,:))
plot(timespan, theta_hat(3,:))
```

```
hold off

title('Parameter values (LMS no noise) mu = 0.3; b_2 = b_3 = 1')
xlabel('Seconds')
ylabel('Parameter value')
legend('b_1', 'b_2', 'b_3')
ylim([0 1.01])

subplot(2,1,2)
hold on
plot(timespan,theta_hat2(1,:))
plot(timespan,theta_hat2(2,:))
plot(timespan, theta_hat2(3,:))
hold off

title('Parameter values (LMS no noise) mu = 0.3; b_2 = b_3 = 1')
xlabel('Seconds')
ylabel('Parameter value')
legend('b_1', 'b_2', 'b_3')
ylim([0 1.01])

saveas(gcf,'figures/q1a_lms_params.png')


%%
% ---------------------------------- %
%     LMS:Changing Mu
% ---------------------------------- %

clear all
clc
close all
load('data1.mat');

Fs = 8192; %sampling freq
D1 = 1 * Fs; % delay 1
D2 = 2.5 * Fs; % delay 2
timespan = [0: 1/Fs: length(loudspeaker)/Fs - 1/Fs]; % Timespan for time-axis

% LMS no noise with mu = 3 b2=b3 = 0
init_params = [1; 0; 0];
mu = 0.3;
[output, theta_hat] = LMS_function(loudspeaker, mike1, init_params, mu);

% LMS no noise with mu = 0.03 b2=b3 = 0
mu = 0.03;
[output2, theta_hat2] = LMS_function(loudspeaker, mike1, init_params, mu);

% LMS no noise with mu = 3 b2=b3 = 0
mu = 3;
[output3, theta_hat3] = LMS_function(loudspeaker, mike1, init_params, mu);

figure(2)
subplot(3,1,1)
hold on
plot(timespan,theta_hat(1,:))
plot(timespan,theta_hat(2,:))
plot(timespan, theta_hat(3,:))
hold off

title('Parameter values (LMS no noise) mu = 0.3')
xlabel('Seconds')
ylabel('Parameter value')
legend('b_1', 'b_2', 'b_3')
ylim([0 1.01])

subplot(3,1,2)
hold on
```

```
plot(timespan,theta_hat2(1,:))
plot(timespan,theta_hat2(2,:))
plot(timespan, theta_hat2(3,:))
hold off

title('Parameter values (LMS no noise) mu = 0.03')
xlabel('Seconds')
ylabel('Parameter value')
legend('b_1', 'b_2', 'b_3')
ylim([0 1.01])

subplot(3,1,3)
hold on
plot(timespan,theta_hat3(1,:))
plot(timespan,theta_hat3(2,:))
plot(timespan, theta_hat3(3,:))
hold off

title('Parameter values (LMS no noise) mu = 3')
xlabel('Seconds')
ylabel('Parameter value')
legend('b_1', 'b_2', 'b_3')
ylim([0 1.01])

saveas(gcf,'figures/q1a_lms_mu.png')


%% A.2
% ---------------------------------- %
%     LMS with noise
% ---------------------------------- %

clear all
close all
clc
load('data1.mat');

Fs = 8192; %sampling freq
D1 = 1 * Fs; % delay 1
D2 = 2.5 * Fs; % delay 2
timespan = [0: 1/Fs: length(loudspeaker)/Fs - 1/Fs]; % Timespan for time-axis

init_params = [1; 0; 0];
mu = 0.3;

[output, theta_hat] = LMS_function(loudspeaker, mike1, init_params, mu);
[noisy_output, noisy_theta_hat] = LMS_function(loudspeaker, noisymike1, init_params, mu);

figure(4)
subplot(2,1,1)
hold on
plot(timespan,theta_hat(1,:))
plot(timespan,theta_hat(2,:))
plot(timespan, theta_hat(3,:))
hold off

title('Parameter values (LMS no noise)')
xlabel('Seconds')
ylabel('Parameter value')
legend('b_1', 'b_2', 'b_3')
ylim([0 1])

subplot(2,1,2)
hold on
plot(timespan,noisy_theta_hat(1,:))
plot(timespan,noisy_theta_hat(2,:))
plot(timespan, noisy_theta_hat(3,:))
hold off
```

```
402
403 title('Parameter values (LMS with noise)')
404 xlabel('Seconds')
405 ylabel('Parameter value')
406 legend('b_1', 'b_2', 'b_3')
407 ylim([0 1])
408
409 saveas(gcf, 'figures/q1a_lms_noise_params.png')
410
411 % ---------------------------------- %
412 %     LMS: Noise, output graphs
413 % ---------------------------------- %
414
415 figure(5)
416 subplot(2,1,1)
417 plot(timespan, transpose(loudspeaker) - output)
418 title('Amplitude Difference between Loudspeaker and output signal(LMS without noise)')
419 xlabel('Seconds')
420 ylabel('Amplitude')
421 ylim([-0.1 0.15])
422 subplot(2,1,2)
423 plot(timespan, transpose(loudspeaker) - noisy_output)
424 title('Amplitude Difference between Loudspeaker and output signal(LMS with noise)')
425 xlabel('Seconds')
426 ylabel('Amplitude')
427 ylim([-0.1 0.15])
428
429 saveas(gcf, 'figures/q1a_lms_noise_diff.png')
```

A-Workshop1B

```
1  %% B.1
2  % ---------------------------------- %
3  %     RLS: Time Varying. No noise
4  % ---------------------------------- %
5
6  clear all
7  close all
8  clc
9  load('data1.mat');
10 load('data2.mat');
11
12 Fs = 8192; %sampling freq
13 D1 = 1 * Fs; % delay 1
14 D2 = 2.5 * Fs; % delay 2
15 timespan = [0: 1/Fs: length(loudspeaker)/Fs - 1/Fs]; % Timespan for time-axis
16
17
18 % RLS with inital params = 0
19
20 init_params = [1; 0; 0];
21 P_init_scale = 1;
22 lambda = 0.995;
23
24
25 [output, theta_hat] = RLS_function2(loudspeaker, mike2, init_params, P_init_scale, lambda);
26
27 figure(1)
28 hold on
29 plot(timespan,theta_hat(1,:))
30 plot(timespan,theta_hat(2,:))
31 plot(timespan, theta_hat(3,:))
32 hold off
33
34 title('Time varying Parameter values (RLS no noise)')
```

```
xlabel('Seconds')
ylabel('Parameter value')
legend('b_1', 'b_2', 'b_3')
ylim([0 1])

saveas(gcf, 'figures/q1b_params.png')

%%
% ---------------------------------- %
%     RLS: Time Varying.  Lambda changing
% ---------------------------------- %
clear all
close all
clc
load('data1.mat');
load('data2.mat')

Fs = 8192; %sampling freq
D1 = 1 * Fs; % delay 1
D2 = 2.5 * Fs; % delay 2
timespan = [0: 1/Fs: length(loudspeaker)/Fs - 1/Fs]; % Timespan for time-axis


% RLS with P_init_scale = 1
init_params = [1; 0; 0];
P_init_scale = 1;
lambda = 0.999;

[output1, theta_hat] = RLS_function2(loudspeaker, mike2, init_params, P_init_scale, lambda);

figure(2)
subplot(3,1,1)
hold on
plot(timespan,theta_hat(1,:))
plot(timespan,theta_hat(2,:))
plot(timespan, theta_hat(3,:))
hold off

title('Time varying Parameter values (RLS no noise) Lambda = 0.999')
xlabel('Seconds')
ylabel('Parameter value')
legend('b_1', 'b_2', 'b_3')
ylim([0 1.01])


% RLS with lambda = 0.990;
lambda = 0.990;

[output001, theta_hat] = RLS_function2(loudspeaker, mike2, init_params, P_init_scale, lambda);

figure(2)
subplot(3,1,2)
hold on
plot(timespan,theta_hat(1,:))
plot(timespan,theta_hat(2,:))
plot(timespan, theta_hat(3,:))
hold off

title('Time varying Parameter values (RLS no noise) lambda = 0.990;')
xlabel('Seconds')
ylabel('Parameter value')
legend('b_1', 'b_2', 'b_3')
ylim([0 1.01])

% RLS with lambda = 0.999;
init_params = [1; 0; 0];
P_init_scale = 100;
lambda = 0.995;
```

```
103
104 [output100, theta_hat] = RLS_function2(loudspeaker, mike2, init_params, P_init_scale, lambda);
105
106 figure(2)
107 subplot(3,1,3)
108 hold on
109 plot(timespan,theta_hat(1,:))
110 plot(timespan,theta_hat(2,:))
111 plot(timespan, theta_hat(3,:))
112 hold off
113
114 title('Time varying Parameter values (RLS no noise) lambda = 0.995')
115 xlabel('Seconds')
116 ylabel('Parameter value')
117 legend('b_1', 'b_2', 'b_3')
118 ylim([0 1.01])
119
120 saveas(gcf, 'figures/q1b_Pinit.png')
121
122 figure(6)
123 subplot(3,1,1)
124 plot(timespan, transpose(loudspeaker) - output1)
125 title('Amplitude Difference for Time Varying Echo Amplitude (RLS without noise) lambda = ...
        0.999;')
126 xlabel('Seconds')
127 ylabel('Amplitude')
128
129 subplot(3,1,2)
130 plot(timespan, transpose(loudspeaker) - output001)
131 title('Amplitude Difference for Time Varying Echo Amplitudes(RLS without noise) lambda = ...
        0.990;')
132 xlabel('Seconds')
133 ylabel('Amplitude')
134
135 subplot(3,1,3)
136 plot(timespan, transpose(loudspeaker) - output100)
137 title('Amplitude Difference for Time Varying Echo Amplitudes (RLS without noise) lambda = ...
        0.995;')
138 xlabel('Seconds')
139 ylabel('Amplitude')
140
141 saveas(gcf, 'figures/q1b_Pinit_output_diff.png')
142
143 %% B.1
144 % ---------------------------------- %
145 %     RLS: Time Varying with noise
146 % ---------------------------------- %
147
148 clear all
149 close all
150 clc
151 load('data1.mat');
152 load('data2.mat')
153
154 Fs = 8192; %sampling freq
155 D1 = 1 * Fs; % delay 1
156 D2 = 2.5 * Fs; % delay 2
157 timespan = [0: 1/Fs: length(loudspeaker)/Fs - 1/Fs]; % Timespan for time-axis
158
159
160 init_params = [1; 0; 0];
161 P_init_scale = 1;
162 lambda = 0.995;
163
164 [output, theta_hat] = RLS_function2(loudspeaker, mike2, init_params, P_init_scale, lambda);
165 [noisy_output, noisy_theta_hat] = RLS_function2(loudspeaker, noisymike2, init_params, ...
        P_init_scale, lambda);
166
```

```
% sound(noisy_output)


figure(4)
subplot(2,1,1)
hold on
plot(timespan,theta_hat(1,:))
plot(timespan,theta_hat(2,:))
plot(timespan, theta_hat(3,:))
hold off

title('Time Varying Parameter values (RLS no noise)')
xlabel('Seconds')
ylabel('Parameter value')
legend('b_1', 'b_2', 'b_3')
ylim([0 1])

subplot(2,1,2)
hold on
plot(timespan,noisy_theta_hat(1,:))
plot(timespan,noisy_theta_hat(2,:))
plot(timespan, noisy_theta_hat(3,:))
hold off

title('Time varying Parameter values (RLS with noise)')
xlabel('Seconds')
ylabel('Parameter value')
legend('b_1', 'b_2', 'b_3')
ylim([0 1])

figure(5)
subplot(2,1,1)
plot(timespan, transpose(loudspeaker) - output)
title('Amplitude Difference between Loudspeaker and output signal for Time Varying Echo ...
    Amplitudes(RLS without noise)')
xlabel('Seconds')
ylabel('Amplitude')
ylim([-0.1 0.15])
subplot(2,1,2)
plot(timespan, transpose(loudspeaker) - noisy_output)
title('Amplitude Difference between Loudspeaker and output signalfor Time Varying Echo ...
    Amplitudes(RLS with noise)')
xlabel('Seconds')
ylabel('Amplitude')
ylim([-0.1 0.15])


%%
% ---------------------------------- %
%     LMS: Time Varying
% ---------------------------------- %

% Changing mu

clear all
close all
clc
load('data1.mat');
load('data2.mat');

Fs = 8192; %sampling freq
D1 = 1 * Fs; % delay 1
D2 = 2.5 * Fs; % delay 2
timespan = [0: 1/Fs: length(loudspeaker)/Fs - 1/Fs]; % Timespan for time-axis

init_params = [1; 0; 0];

% LMS no noise with mu = 3 b2=b3 = 0
```

```
mu = 0.3;
[output, theta_hat] = LMS_function(loudspeaker, mike2, init_params, mu);

% LMS no noise with mu = 0.03 b2=b3 = 0
mu = 0.03;
[output2, theta_hat2] = LMS_function(loudspeaker, mike2, init_params, mu);

% LMS no noise with mu = 3 b2=b3 = 0
mu = 3;
[output3, theta_hat3] = LMS_function(loudspeaker, mike2, init_params, mu);

figure(2)
subplot(3,1,1)
hold on
plot(timespan,theta_hat(1,:))
plot(timespan,theta_hat(2,:))
plot(timespan, theta_hat(3,:))
hold off

title('Time Varying Parameter values (LMS no noise) \mu = 0.3')
xlabel('Seconds')
ylabel('Parameter value')
legend('b_1', 'b_2', 'b_3')
ylim([0 1.01])

subplot(3,1,2)
hold on
plot(timespan,theta_hat2(1,:))
plot(timespan,theta_hat2(2,:))
plot(timespan, theta_hat2(3,:))
hold off

title('Time Varying Parameter values (LMS no noise) \mu = 0.03')
xlabel('Seconds')
ylabel('Parameter value')
legend('b_1', 'b_2', 'b_3')
ylim([0 1.01])

subplot(3,1,3)
hold on
plot(timespan,theta_hat3(1,:))
plot(timespan,theta_hat3(2,:))
plot(timespan, theta_hat3(3,:))
hold off

title('Time Varying Parameter values (LMS no noise) \mu = 3')
xlabel('Seconds')
ylabel('Parameter value')
legend('b_1', 'b_2', 'b_3')
ylim([0 1.01])

saveas(gcf,'figures/q1b_lms_mu.png')

%%
% ---------------------------------- %
%     LMS: Time Varying and Noise
% ---------------------------------- %

clear all
close all
clc
load('data1.mat');
load('data2.mat');

Fs = 8192; %sampling freq
D1 = 1 * Fs; % delay 1
D2 = 2.5 * Fs; % delay 2
timespan = [0: 1/Fs: length(loudspeaker)/Fs - 1/Fs]; % Timespan for time-axis
```

```
301
302 init_params = [1; 0; 0];
303 mu = 0.3;
304
305 [output, theta_hat] = LMS_function(loudspeaker, mike2, init_params, mu);
306 [noisy_output, noisy_theta_hat] = LMS_function(loudspeaker, noisymike2, init_params, mu);
307
308 figure(4)
309 subplot(2,1,1)
310 hold on
311 plot(timespan,theta_hat(1,:))
312 plot(timespan,theta_hat(2,:))
313 plot(timespan, theta_hat(3,:))
314 hold off
315
316 title('Time Varying Parameter values (LMS no noise)')
317 xlabel('Seconds')
318 ylabel('Parameter value')
319 legend('b_1', 'b_2', 'b_3')
320 ylim([0 1])
321
322 subplot(2,1,2)
323 hold on
324 plot(timespan,noisy_theta_hat(1,:))
325 plot(timespan,noisy_theta_hat(2,:))
326 plot(timespan, noisy_theta_hat(3,:))
327 hold off
328
329 title('Time Varying Parameter values (LMS with noise)')
330 xlabel('Seconds')
331 ylabel('Parameter value')
332 legend('b_1', 'b_2', 'b_3')
333 ylim([0 1])
334
335 saveas(gcf, 'figures/q1b_lms_noise_params.png')
336
337 figure(5)
338 subplot(2,1,1)
339 plot(timespan, transpose(loudspeaker) - output)
340 title('Time Varying Params: Amplitude Difference between Loudspeaker and output signal(LMS ...
        without noise)')
341 xlabel('Seconds')
342 ylabel('Amplitude')
343 ylim([-0.1 0.15])
344 subplot(2,1,2)
345 plot(timespan, transpose(loudspeaker) - noisy_output)
346 title('Time Varying Params: Amplitude Difference between Loudspeaker and output signal(LMS ...
        with noise)')
347 xlabel('Seconds')
348 ylabel('Amplitude')
349 ylim([-0.1 0.15])
350
351 saveas(gcf, 'figures/q1b_lms_noise_diff.png')
```

A-RLS Function 1

```
1 function [ output, theta_hat ] = RLS_function(loudspeaker, mike1, init_params, P_init_scale)
2
3 Fs = 8192; %sampling freq
4 D1 = 1 * Fs; % delay 1
5 D2 = 2.5 * Fs; % delay 2
6
7 P = P_init_scale * eye(3);
8
9 theta_hat = zeros(3,length(loudspeaker));
```

```
loudspeakerDelay1 = 0;
loudspeakerDelay2 = 0;

output  = zeros(1,length(loudspeaker));


for i=1:length(loudspeaker)

    if (i > D1)
        loudspeakerDelay1 = loudspeaker(i - D1);
    end

    if (i > D2)
        loudspeakerDelay2 = loudspeaker(i - D2);
    end

    phi = [loudspeaker(i); loudspeakerDelay1;  loudspeakerDelay2];

    P = P - (P*phi*transpose(phi)*P)/(1 + transpose(phi)*P*phi);

    G = P*phi;

    if ( i > 1)
        theta_hat(:,i) = theta_hat(:,i-1) + G*(mike1(i) - transpose(phi)*theta_hat(:,i-1));
    else
        theta_hat(:,i) = init_params + G*(mike1(i) - transpose(phi)*init_params);
    end

    output(i) = mike1(i) - theta_hat(2,i)*loudspeakerDelay1 - ...
        theta_hat(3,i)*loudspeakerDelay2;
end


end
```

A-RLS Function with Lambda

```
function [ output, theta_hat ] = RLS_function2(loudspeaker, mike1, init_params, ...
    P_init_scale, lambda)

% ---------------------------------- %
%     RLS function with lambda value
% ---------------------------------- %

Fs = 8192; %sampling freq
D1 = 1 * Fs; % delay 1
D2 = 2.5 * Fs; % delay 2

P = P_init_scale * eye(3);

theta_hat = zeros(3,length(loudspeaker));
loudspeakerDelay1 = 0;
loudspeakerDelay2 = 0;

output  = zeros(1,length(loudspeaker));


for i=1:length(loudspeaker)

    if (i > D1)
        loudspeakerDelay1 = loudspeaker(i - D1);
    end

    if (i > D2)
        loudspeakerDelay2 = loudspeaker(i - D2);
```

```
    end

    phi = [loudspeaker(i); loudspeakerDelay1;  loudspeakerDelay2];

    P = (1/lambda)*(P - (P*phi*transpose(phi)*P)/(lambda + transpose(phi)*P*phi));

    G = P*phi;

    if ( i > 1)
        theta_hat(:,i) = theta_hat(:,i-1) + G*(mike1(i) - transpose(phi)*theta_hat(:,i-1));
    else
        theta_hat(:,i) = init_params + G*(mike1(i) - transpose(phi)*init_params);
    end

    output(i) = mike1(i) - theta_hat(2,i)*loudspeakerDelay1 - ...
        theta_hat(3,i)*loudspeakerDelay2;
end


end
```

A-LMS Function

```
function [ output, theta_hat ] = LMS_function(loudspeaker, mike1, init_params, mu)

% ---------------------------------- %
%     LMS: function
% ---------------------------------- %

Fs = 8192; %sampling freq
D1 = 1 * Fs; % delay 1
D2 = 2.5 * Fs; % delay 2

theta_hat = zeros(3,length(loudspeaker));
loudspeakerDelay1 = 0;
loudspeakerDelay2 = 0;

output  = zeros(1,length(loudspeaker));


for i=1:length(loudspeaker)

    if (i > D1)
        loudspeakerDelay1 = loudspeaker(i - D1);
    end

    if (i > D2)
        loudspeakerDelay2 = loudspeaker(i - D2);
    end

    phi = [loudspeaker(i); loudspeakerDelay1;  loudspeakerDelay2];

    G = mu*phi;

    if ( i > 1)
        theta_hat(:,i) = theta_hat(:,i-1) + G*(mike1(i) - transpose(phi)*theta_hat(:,i-1));
    else
        theta_hat(:,i) = init_params + G*(mike1(i) - transpose(phi)*init_params);
    end

    output(i) = mike1(i) - theta_hat(2,i)*loudspeakerDelay1 - ...
        theta_hat(3,i)*loudspeakerDelay2;
end


```

```
end
```

B- Gain Estimate

```
#include "SP2WS1.h"


// input signal history
float insignal1[100], insignal2[100];



// function prototypes
void gainestimateLMS(float,float,float,float[2]);
void gainestimateRLS(float,float,float,float[2]);


void gainestimate(float in1, float in2, float out, float gain[2])
{
    // record input signal history for checking signal magnitude using plot facility
    for (int i = 99; i > 0; i--)
    {
        insignal1[i] = insignal1[i-1];
        insignal2[i] = insignal2[i-1];
    }
    insignal1[0] = in1;
    insignal2[0] = in2;


    // estimate gain
    gain[0] = 0;
    gain[1] = 1;
//    gainestimateLMS(in1, in2, out, gain);
     gainestimateRLS(in1, in2, out, gain);
}

void gainestimateLMS(float in1, float in2, float out, float gain[2])
{
    // TODO: Implement gain estimation using LMS algorithm
    static float theta_hat1 = 0, theta_hat2 = 1.0;

    float mu = 1E-20;

    float err = out - (in1 * theta_hat1 + in2 * theta_hat2);

    theta_hat1 = theta_hat1 + (mu * in1) * err;
    theta_hat2 = theta_hat2 + (mu * in2) * err;

    gain[0] = theta_hat1;
    gain[1] = theta_hat2;

}

void gainestimateRLS(float in1, float in2, float out, float gain[2])
{
    // TODO: Implement gain estimation using RLS algorithm
    static float theta_hat1 = 0, theta_hat2 = 1.0, p11 = 1E-25, p12 = 0, p21 = 0, p22 = ...
        1E-25;  // p11 = p22 = 1 for identity

    float lambda = 0.9999;

    float err = out - (in1 * theta_hat1 + in2 * theta_hat2);


    float p11_in1 = p11 * in1;
```

```
61      float p11_in2 = p11 * in2;
62      float p12_in1 = p12 * in1;
63      float p12_in2 = p12 * in2;
64      float p21_in1 = p21 * in1;
65      float p21_in2 = p21 * in2;
66      float p22_in1 = p22 * in1;
67      float p22_in2 = p22 * in2;
68
69      float inv_lambda = 1.0 / lambda;
70
71      float denom = (lambda + in1 * (p11_in1 + p21_in2) + in2 * (p12_in1 + p22_in2));
72
73      float p11_new = inv_lambda * (p11 - (p11_in1 * (p11_in1 + p12_in2) + p21_in2 * ...
            (p11_in1 + p12_in2)) / denom);
74      float p12_new = inv_lambda * (p12 - (p12_in1 * (p11_in1 + p12_in2) + p22_in2 * ...
            (p11_in1 + p12_in2)) / denom);
75      float p21_new = inv_lambda * (p21 - (p11_in1 * (p21_in1 + p22_in2) + p21_in2 * ...
            (p21_in1 + p22_in2)) / denom);
76      float p22_new = inv_lambda * (p22 - (p12_in1 * (p21_in1 + p22_in2) + p22_in2 * ...
            (p21_in1 + p22_in2)) / denom);
77
78
79      p11 = p11_new;
80      p12 = p12_new;
81      p21 = p21_new;
82      p22 = p22_new;
83
84      theta_hat1 = theta_hat1 + ((p11 * in1) + (p12 * in2)) * err;
85      theta_hat2 = theta_hat2 + ((p21 * in1) + (p22 * in2)) * err;
86
87      gain[0] = theta_hat1;
88      gain[1] = theta_hat2;
89      // then save p values for next round
90
91  }
```