ELT-43007 DIGITAL COMMUNICATIONS

Matlab Exercise #4

Channel Estimation and Basic Adaptive Filtering

In this exercise, you are **required** to create a runnable m-file during the exercise. Always use correct figure numbers, e.g., figure(3) for figure number 3, so the overall m-file will produces nice figures.

Be prepared to discuss with the teacher about the questions given in red text. Teacher might approach you and start discussion, so think about them and try them beforehand with your freshly generated Matlab m-file.

1 SYSTEM MODEL AND GENERATION OF TRANSMITTED SIGNAL AND MODELLING THE CHANNEL

1.1 SYSTEM MODEL

In this exercise, we create a baseband equivalent 16-QAM transmission system. We use oversampling factor of 1 for simplicity, so the model is working at symbol rate. We create symbols at the transmitter, model a channel, and perform linear equalization of the channel at the receiver. The system model is depicted in Figure 1. Notice that the setup is very similar to the setup of the previous exercise. Previously we designed equalizer, based on known channel. Now we use training data to estimate the channel impulse response, or the equalizer directly. Typically, we do not know the channel, so this is more practically oriented example.

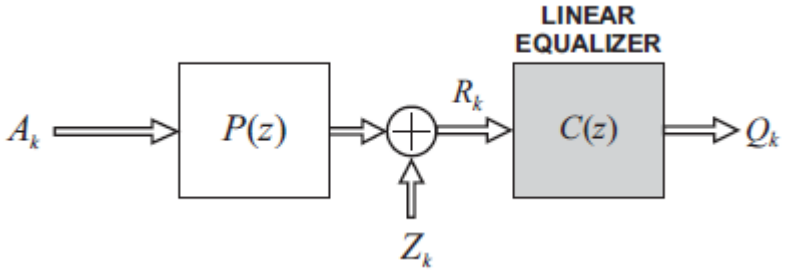

Figure 1: System model with

1.2 SYSTEM PARAMETERS AND TRANSMITTED SEQUENCE GENERATION

- Let's first set the simulation parameters:

```
SNR = 15; %Signal to noise ratio
Rs = 20e6; %Let's define symbol rate for the plotting purposes
```

- Then let's create the transmitted symbols/samples (note 1/Rs=T):

```
%% Creation of the data
%16-QAM alphabet:
```

```
a = repmat(-3:2:3,1,4)-[repmat(-3j,1,4) repmat(-1j,1,4) repmat(1j,1,4) repmat(3j,1,4)];
Ak = a(randi(16,1000,1)); % 16-QAM sequence of 1000 samples
```

- Let's also plot ideal constellation:

```
figure(1); plot(real(a),imag(a),'o'); title('16-QAM constellation'); grid;
```

You should now see a pure 16-QAM constellation. Try to understand how the random data is mapped to constellation points. The above way to create 16-QAM alphabets is short. Can you think of other ways to create the alphabet?

1.3 BASEBAND EQUIVALENT SYSTEM MODEL

- We start by defining the channel:

```
%% Channel creation and channel modelling
p = [0.19+.56j .45-1.28j -.14-.53j -.19+.23j .33+.51j]; % Example complex channel model
L1=1; % Channel maximum tap is the second one
```

This is just an example channel. It is a single realization of truncated version of actual channel used, e.g., in 3G LTE link evaluations.

- Now we plot the amplitude response of the channel. Later we will add more plots in this figure, so don't delete it:

```
%Plotting amplitude response of the channel:
figure(2); [H,f]=freqz(p,1,-Rs/2:Rs/200:Rs/2,Rs); plot(f/1e6,20*log10(abs(H)),'b');
xlabel('Frequency [MHz]');ylabel('Amplitude response [dB]'); legend('Channel');
```

1.4 APPLYING THE CHANNEL AND NOISE TO THE TRANSMITTED SEQUENCE

- Now we use filter function to apply the channel. We also get rid of the pre-cursor delay:

```
Rk = filter(p,1,Ak); % Received signal
Rk = Rk(L1+1:end); % Discard the delay, if channel has pre-cursor taps
```

- Then, we create and add the noise, and have the received signal

```
noise = (1/sqrt(2))*(randn(size(Rk)) + 1j*randn(size(Rk))); %Initial noise vector
P_s = var(Rk); % Signal power
P_n = var(noise); % Noise power
% Defining noise scaling factor based on the desired SNR:
noise_scaling_factor = sqrt(P_s/P_n./10.^(SNR./10));
Rk_noisy=Rk+noise*noise_scaling_factor; % Received signal
```

- Now, we plot the constellation of the signal after the channel and noise. ISI and noise are now present in the constellation

```
figure(1); plot(Rk,'*k'); legend('Received constellation')
```

2 CHANNEL ESTIMATION

Here we are estimating the channel by using block least squares method (see p.335-336). The method uses pilot symbols for the estimation task. We assume that we estimate finite number of impulse response taps with finite number of pilot symbols.

- We assume that we have *M* pilots symbols, and that we try to estimate *estimate_length* number of taps of the ISI channel with block-LS approach (p.335-336)

```
M = 30; %number of reference symbols used for channel estimation; this is pretty small amount
estimate_length = 7; %this defines how long is the channel estimate's impulse response
A_conv=convmtx(Ak(2:M+1).',estimate_length); % Convolution matrix
p_LS=((A_conv'*A_conv)\(A_conv'))*Rk_noisy(1:size(A_conv,1)).'; % LS solution
```

- We plot the estimated channel amplitude response to the same figure with the actual amplitude response of the channel

```
figure(2); hold on; [H,f]=freqz(p_LS,1,-Rs/2:Rs/200:Rs/2,Rs);
plot(f/1e6,20*log10(abs(H)),'r'); legend('Channel','LS Channel Estimate');
```

- We also plot the absolute values of the impulse responses of the channel and the LS channel estimate:

```
figure(3); hold on; stem(-L1:length(p)-L1-1,abs(p),'k');
stem(-L1:length(p_LS)-L1-1,abs(p_LS),'r');
legend('Channel','LS channel estimate');
title('Absolute values of the impulse responses')
```

Try out different number of estimated channel taps and with different number of pilot symbols. What can you see?

3 DESIGN OF CHANNEL EQUALIZER WITH MINIMUM MSE CRITERION AND WITH ADAPTIVE LMS ALGORITHM

Here we design directly the channel equalizers instead of estimating the channel. This way, we do not have need to construct the channel equalizer separately from the estimated channel impulse response. We first use minimum MSE criterion (see p.337-338). We use huge amount of training symbols, to attain near ideal equalizer. Then, we use least mean square (LMS) algorithm (see p.355-359) to design channel equalizer. This is very practical solution for channel equalizer design.

3.1 MMSE CHANNEL EQUALIZATION

- We now do MMSE channel equalization. We have equalizer with 31 taps designed, and we use all data as training symbol set. This gives us nearly optimal equalizer. This is our reference filter. With a relatively long equalizer, we deliberately allow for considerable amount of both pre-cursor and post-cursor taps (15+15 in the below example code)

```
%% MSE Equalizer (example with 10000 training symbols)
% Initialization
FII = zeros(31,31); % Autocorrelation Matrix initialization
alfa = zeros(31,1); % Cross-correlation vector initialization
% Estimating FII and alfa using sample estimates based on training symbols
% Notice that here we use all the generated data as training symbols
for i = 16:length(Rk_noisy)-15,
    rk = flipud(Rk_noisy(i-15:i+15).'); % Received signal vector
    FII = FII + rk*conj(rk).'; % Autocorrelation matrix
    alfa = alfa + Ak(i)*conj(rk);
end
FII = FII/(length(Rk_noisy)-30); % Final sample estimate of the autocorrelation matrix
alfa = alfa/(length(Rk_noisy)-30); % Final sample estimate of the cross-correlation vector
c_MSE = inv(conj(FII))*alfa; % Equalizer coefficients
figure(4); stem(abs(conv(p,c_MSE)));
```

```
title('Effective impulse reponse (abs) of the MSE equalized system')
figure(5); [H,f]=freqz(p,1,-Rs/2:Rs/200:Rs/2,Rs); plot(f/1e6,20*log10(abs(H)),'b');
xlabel('Frequency [MHz]');ylabel('Amplitude response [dB]');
figure(5); hold on; [H,f]=freqz(c_MSE,1,-Rs/2:Rs/200:Rs/2,Rs);
plot(f/1e6,20*log10(abs(H)),'k');
legend('Channel','MSE Equalizer');
```

You can try different equalizer lengths and different amounts of training symbols, and see how it affect the equalizer amplitude response. Notice that you need to change many parts of the code. Are you able to do it and get it to work?

3.2 DESIGN OF CHANNEL EQUALIZER USING LMS ALGORITHM

- Now we design channel equalizer by using least mean squares algorithm (see p.355-359). We start from equalizer consisting of zeros. Then we check what is the error and take a small step toward the negative gradient of the error surface (approximatively). We use step-size 0.0001 to start with. We use all the data as training symbols so we can study the convergence process. Once again, our equalizer size is 31 taps with 15+15 pre-cursor and post-cursor taps.

```
beta = 0.0001; % step-size of the algorithm
c_LMS = zeros(31,1); % equalizer coefficients, initializations
for i = 16:length(Rk_noisy)-15
    rk = flipud(Rk_noisy(i-15:i+15).'); % Received signal vector
    Ek(i) = Ak(i) - c_LMS.'*rk; % Error signal, we assume a known symbol sequence
    c_LMS = c_LMS + beta*Ek(i)*conj(rk); % LMS update !
end

figure(6); hold on; plot(abs(Ek)); title('Convergence behavior of the LMS-algorithm.');
ylabel('LMS error'); xlabel('Iteration index');
figure(7); hold on; stem(abs(conv(p,c_LMS)));
title('Effective impulse response (abs) of the equalized system ')
figure(5); hold on; [H,f]=freqz(c_LMS,1,-Rs/2:Rs/200:Rs/2,Rs);
plot(f/1e6,20*log10(abs(H)),'r');
[H,f]=freqz(conv(c_LMS,p),1,-Rs/2:Rs/200:Rs/2,Rs);
plot(f/1e6,20*log10(abs(H)),'g');
legend('Channel','MSE Equalizer','LMS Equalizer','Total Response (LMS)');

figure(1); hold on; plot(filter(c_LMS,1,Rk),'rx');
legend('Received constellation','LMS Equalized constellation')
```

Try different step sizes (e.g., 0.001). What happens to the LMS error? Why? Can you see any difference in the equalizer amplitude response? You can for example copy the above code to your m-file two times. Then you can plot the convergence curves to the same figures with different step sizes. Try this. Use different colors for the different step size.

Try also different signal to noise ratios. Is the selection of the step size depending anyhow on the SNR?

You can also try out different amounts of training symbols and different equalizer sizes. Notice that you need to change the code. Are you able to do it?