COMM.SYS.300 COMMUNICATION THEORY

Matlab Exercise #4

Carrier modulated digital transmission: Transmitter and receiver structures – QAM signals, up/downconversion, timing and phase synchronization, and symbol detection

1 SYSTEM MODEL AND GENERATION OF QAM SYMBOLS

1.1 SYSTEM MODEL

In this exercise, we create a carrier modulated QAM transmission system including a transmitter (TX), a receiver (RX), and a simple AWGN (Additive White Gaussian Noise) channel model (see Fig. 1). To add an additional practical aspect, we assume that the TX and RX oscillator clocks are not synchronized which causes a phase error in the detected symbols in the receiver. To compensate this phase error and to obtain a correct timing for symbol sampling in the receiver, we include a specific training symbols (also called as preamble, pilots, etc.) in the transmission.

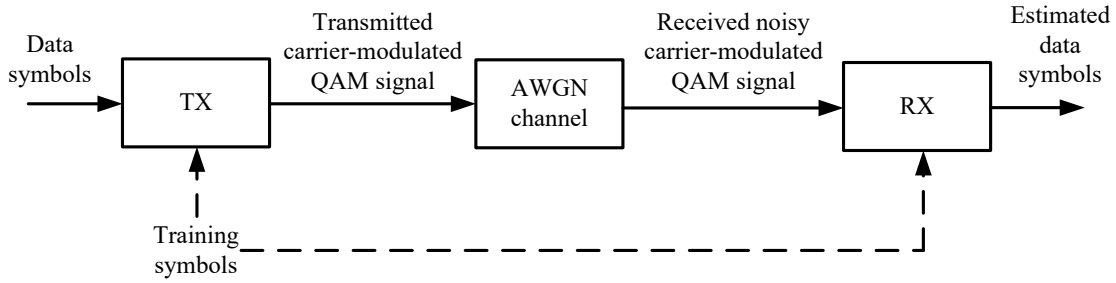

Figure 1: General system model

We consider the TX and RX structures to be similar to the ones shown in lecture notes (see Fig. 2 and Fig. 3), except that we skip the bit-to-symbol and symbol-to-bit transformation, and consider only transmission of symbols (if you like, it is trivial to include bits in the system later on). Here, both the TX and RX structure are based on complex calculations, but the same structures can also be implemented by using only real-valued signals (i.e., I/Q-modulation with separate I and Q branches, see the lecture notes for more details).

As transmit and receive filters we use the Root-Raised-Cosine (RRC) filters (see the lecture slides). Together (in TX and RX) these filters fulfil the Nyquist criterion (i.e. no Inter-Symbol-Interference (ISI). Notice that a single RRC does not do this unlike with the conventional raised-cosine filter.

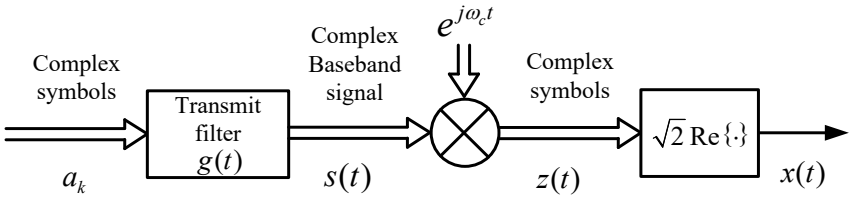

Figure 2: Transmitter (TX) structure.

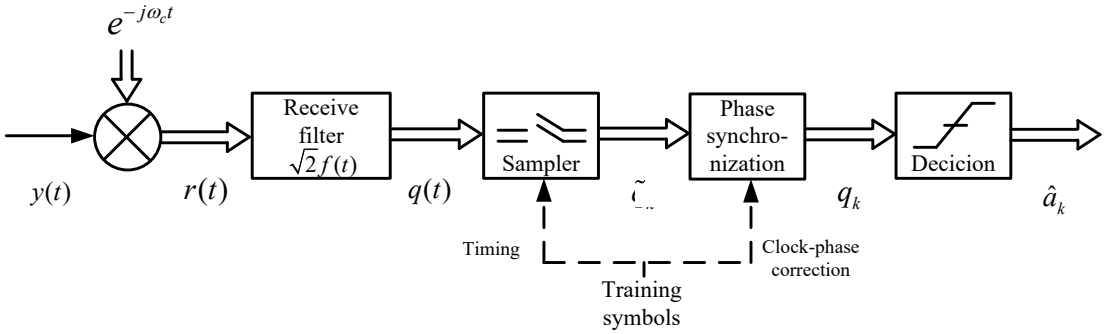

Figure 3: Receiver (RX) structure.

1.2 CONSIDERED SYSTEM PARAMETERS

- First let's define the general system parameters as follows:

```
alphabet_size = 16;
% Number of symbols in the QAM alphabet (e.g. 16 means 16-QAM). Valid alphabet
% sizes are 4, 16, 64, 256, 1024, ... (i.e. the possible values are given
% by the vector 2.^(2:2:N), for any N)

SNR = 10;                    % Signal-to-noise power ratio [dB]
T = 1/10e6;                  % Symbol time interval [s]
fc = 75e6;                   % Carrier frequency
r = 20;                      % Oversampling factor (r samples per pulse)
N_symbols_per_pulse = 30;    % Duration of TX/RX-filters in numbers of symbols
alfa = 0.25;                 % Roll-off factor (excess bandwidth)
```

- Based on above we can define sampling frequency and sampling time interval as

```
Fs = r/T;                % Sampling frequency
Ts = 1/Fs;               % Sampling time interval
```

Then we define the number of symbols to be transmitted. To acquire timing and phase synchronization in the receiver, we add training symbols (QAM-symbols as well) in the beginning of the transmitted symbol frame (see Fig. 4). Notice that training symbols are predefined in the system, and thus known by the receiver, and they do not carry any user information which reduces the spectral efficiency of the system.

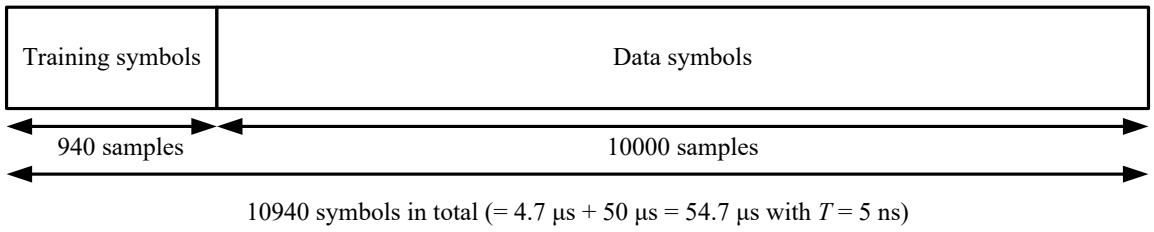

Figure 4: Symbol frame to be transmitted

- Based on Fig. 4, we define the number of training symbols and data symbols as

```
N_data_symbols = 10000;      % Number of data symbols
N_training_symbols = 940;    % Number of training symbols
```

1.3 GENERATION OF QAM SYMBOLS AND THE TRANSMITTED SYMBOL FRAME

- Generate the user data symbols
  - Create a QAM constellation (help bsxfun)
  - Scale the constellation so that the expected average power of transmitted symbols equals to one
  - Generate the specified number (N_data_symbols) of random data symbols
  - Plot the transmitted symbols in complex plane (a constellation)

```
% Here qam_axis presents the symbol values in real/imaginary axis. So,
%generally for different alphabet/constellation sizes ():
qam_axis = -sqrt(alphabet_size)+1:2:sqrt(alphabet_size)-1;
% For example, the above results in
% qam_axis = [-1 1];               % for QPSK
% qam_axis = [-3 -1 1 3];          % for 16-QAM
% qam_axis = [-7 -5 -3 -1 1 3 5 7]; % for 64-QAM

% generation of a complex constellation:
alphabet = bsxfun(@plus,qam_axis',1j*qam_axis); %help bsxfun
%%% equivalent to alphabet = repmat(qam_axis', 1, sqrt(alphabet_size)) +
repmat(1j*qam_axis, sqrt(alphabet_size), 1); %%%
alphabet = alphabet(:).'; % alphabet symbols as a row vector

% Scaling the constellation, so that the mean power of a transmitted symbol
% is one (e.g., with QPSK this is 1/sqrt(2), and for 16-QAM 1/sqrt(10))
alphabet_scaling_factor = 1/sqrt(mean(abs(alphabet).^2));
alphabet = alphabet*alphabet_scaling_factor;

% Random vector of symbol indices (i.e., numbers between 1...alphabet_size)
symbol_ind = randi(length(alphabet),1,N_data_symbols);

data_symbols = alphabet(symbol_ind); % Data symbols to be transmitted

figure
plot(data_symbols,'bo')
xlabel('Re')
xlabel('Im')
title('Transmitted data symbols')
```

- Generate training (QAM) symbols and create the transmitted symbol frame
  - Use the above-defined QAM constellation also for
  - Generate the specified number (N_training_symbols) of random training symbols

```
% Generation of training symbols (similar to data symbols):
training_symbols = alphabet(randi(length(alphabet),1,N_training_symbols));

% Concatenating the training and data symbols to get the overall
% transmitted symbols:
symbol_frame = [training_symbols data_symbols];
```

2 TRANSMITTER STRUCTURE

By following the TX structure in Fig. 2, we generate a bandpass (carrier modulated) QAM signal by based on the above-defined system parameters and the given symbol frame.

- Implement the transit filter: Root-Raised-Cosine (RRC) and plot the pulse shape

```
p = rcosdesign(alfa,N_symbols_per_pulse,r,'sqrt');
figure
plot(-N_symbols_per_pulse*r/2*Ts:Ts:N_symbols_per_pulse*r/2*Ts,p,'b')
hold on
plot(-N_symbols_per_pulse*r/2*Ts:T:N_symbols_per_pulse*r/2*Ts,p(1:r:end),'ro')
xlabel('time [s]')
xlabel('Amplitude')
title('Transmit/receive RRC filter (pulse shape)')
legend('Pulse shape','Ideal symbol-sampling locations')
```

- Filter the transmitted symbol frame
    - Remember to upsample the symbol sequence rate to match with sampling rate of the filter/pulse:

```
symbols_upsampled = zeros(size(1:r*length(symbol_frame)));    % Zero vector
initilized for Up-sampled symbol sequence
symbols_upsampled(1:r:r*length(symbol_frame)) = symbol_frame;
% I.e. now the up-sampled sequence looks like {a1 0 0... a2 0 0... a3 0 0...}
x_LP = filter(p,1,symbols_upsampled); % Transmitter filtering
x_LP = x_LP(1+(length(p)-1)/2:end); % Filter delay correction
%notice the here x_LP is the complex-valued lowpass equivalent signal of the
transmitted real-valued bandpass signal x_BP (generated in the next stage)
```

- Implement the upconversion (modulation / frequency translation to the carrier frequency f_c)
    - Define the time vector for the oscillator signal based on the reference clock time in the TX
    - Generate the complex-exponential carrier signal using the above-defined time vector
    - Multiply (=”mix”) the carrier signal with the low-pass equivalent QAM signal (x_LP)

```
% Time vector for the TX oscillator signal:
t_TX_oscillator = 0:Ts:Ts*(length(x_LP)-1);
% TX oscillator signal:
TX_oscillator_signal = exp(1j*2*pi*fc*t_TX_oscillator);
% Carrier modulation / upconversion (still complex valued):
x_BP_complex = x_LP.*TX_oscillator_signal;
```

- To finalize the TX process (lowpass-to-bandpass transformation) take the real part of the signal (and scale with sqrt(2))

```
% Taking the real value to finalize the lowpass-to-bandpass transformation:
x_BP = sqrt(2)*real(x_BP_complex);
```

- Plot the following figures
    - The complex-valued low-pass equivalent signal x_LP in time and frequency domain
    - The complex-valued bandpass signal x_BP_complex in frequency domain
    - The real-valued bandpass signal x_BP in time and frequency domain

```
figure % zoom manually to see the signal better
plot(t_TX_oscillator, abs(x_LP))
xlabel('Time [s]')
ylabel('Amplitude (of a complex signal)')
title('Lowpass signal in time domain')

figure % zoom manually to see the signal better
plot(t_TX_oscillator, x_BP) %notice no abs needed
xlabel('Time [s]')
```

```
ylabel('Amplitude')
title('Bandpass signal in time domain')

NFFT = 2^14; %FFT size
f = -Fs/2:1/(NFFT*Ts):Fs/2-1/(NFFT*Ts);%frequency vector

figure
plot(f/1e6, fftshift(abs(fft(x_LP,NFFT))))
xlabel('Frequency [MHz]')
ylabel('Amplitude ')
title('Amplitude spectrum of the lowpass signal')

figure
plot(f/1e6, fftshift(abs(fft(x_BP_complex,NFFT)))) %notice no abs needed
xlabel('Frequency [MHz]')
ylabel('Amplitude')
title('Amplitude spectrum of the bandpass signal')

figure
plot(f/1e6, fftshift(abs(fft(x_BP,NFFT)))) %notice no abs needed
xlabel('Frequency [MHz]')
ylabel('Amplitude')
title('Amplitude spectrum of the bandpass signal')
```

3 CHANNEL MODEL

Here we consider a simple AWGN channel model. Based on the earlier exercises, we create white random noise, scale it with the proper scaling factor to obtain the desired SNR, and then add it on top of the transmitted signal (x_BP).

- Generate the noise

```
n = randn(size(x_BP));       % White Gaussian random noise
P_x_BP = var(x_BP); % Signal power
P_n = var(n); % Noise power
% Defining noise scaling factor based on the desired SNR:
noise_scaling_factor = sqrt(P_x_BP/P_n/10^(SNR/10)*(r/(1+alfa)));
```

  - Why r/(1+alfa) part?

```
% Noisy signal
y_BP = x_BP + noise_scaling_factor*n;
```

  - Plot the amplitude spectrum of the noisy bandpass signal

```
% use the previous spectrum figures by just replacing the plotted variable
```

4 RECEIVER STRUCTURE

Typically, due to many unknown/uncertain parameters, most of the complexity in a communications system is found on the RX side. Now, by following the RX structure in Fig. 3, we estimate the transmitted symbols from the received noisy bandpass QAM signal.

4.1 DOWNCONVERSION AND SIGNAL FILTERING

- Implement the downconversion (i.e. demodulation back to the baseband) by using the same principle as in the upconversion in TX, but remember to use the RX reference clock time:

```
% Time vector for the RX oscillator signal:
t_RX_oscillator = 0:Ts:Ts*(length(y_BP)-1);
% RX oscillator signal (notice the minus-sign compared to TX oscillator!!!):
```

```
RX_oscillator_signal = exp(-1j*2*pi*fc*t_RX_oscillator);
% Carrier demodulation / downconversion (signal becomes complex again)
y_BP_downconverted = y_BP.*RX_oscillator_signal;
```

- Plot the amplitude spectrum of the downconverted signal

```
% use the previous spectrum figures by just replacing the plotted variable
```

- Filter the received signal with the receive filter (RRC similar to TX)

```
X_LP_received = sqrt(2)*filter(p,1,y_BP_downconverted); % Receiver filtering
X_LP_received = X_LP_received(1+(length(p)-1)/2:end); Filter delay correction
```

- Plot the amplitude spectrum of the downconverted and filtered signal

```
% use the previous spectrum figures by just replacing the plotted variable
```

4.2 SIGNAL SAMPLING

- Sample the and remove oversampling

```
% Sampling the received signal in order to get symbol samples
RX_symbol_frame = X_LP_received(1:r:end);
```

- Take user data symbols and training symbols in separate vectors:

```
RX_training_symbols = RX_symbol_frame(1:N_training_symbols);
RX_data_symbols = RX_symbol_frame(N_training_symbols+1:end);
```

- Plot the received symbol samples in complex plane (constellation)

```
figure
plot(RX_data_symbols,'bo')
hold on
plot(alphabet,'rs')
hold off
xlabel('Re')
xlabel('Im')
title('Received data symbols with clock offset (phase error)')
```

4.3 OBTAINING SYMBOL DECISIONS AND CALCULATING THE SYMBOL ERROR RATE (SER)

The final step is to make the symbol decisions. This is done based on the minimum distance principle, where the symbol estimate is defined as that symbol of the alphabet, which minimizes the distance to the symbol sample.

- Calculate the Euclidian distance between each symbol sample and each alphabet symbol:

```
alphabet_error_matrix = abs(bsxfun(@minus,alphabet.',RX_data_symbols));
% Now, rows represent the alphabet symbol indices and columns represent the
% received symbol indices (e.g. the Euclidian distance between the 5th
% received symbol and the 3rd symbol in the alphabet is given as
% "alphabet_error_matrix(3,5)"
```

- Find the indices of the alphabet symbols, which have the minimum distance to the symbol samples:

```
[~,estimated_symbol_ind] = min(alphabet_error_matrix);
```

In the end, we find out which symbols were received incorrectly and define the observed Symbol Error Rate (SER)

```
% Finding out which symbols were estimated incorrecly:
symbol_errors = ...
    estimated_symbol_ind ~= symbol_ind(1:length(estimated_symbol_ind));
```

```
% notice that due to filtering transitions, we lose some of the last
% symbols in the symbol frame. In practice we would simply continue taking
% a few samples after the frame to try to get all the symbols. However, filter
% transitions in the beginning and in end of the frame are always creating
% non-idealities to the transmission (the same is also happening in
% frequency domain: e.g. compare data in the middle and in the edge of
% the used band).


% Symbol Error Rate (SER) (0 means 0% of errors, 1 means 100% of errors)
SER = mean(symbol_errors)
```

What's the worst value for the SER? Why?

- Try the above code by altering the SNR and the QAM constellation size (QPSK, 64-QAM, etc.)

5 TIMING SYNCHRONIZATION AND PHASE CORRECTION

This part of the exercise demonstrates common errors of transmission systems: timing and phase errors; and shows how they can be corrected. Therefore, first we introduce the errors in the transmission system and then synchronize the symbols in Sect. 5.1 and compensate the phase errors in Sect. 5.2.

- We add an unknown propagation delay to the AWGN channel model. In contrast to, e.g., the RX filter delay which is known, the channel delay is random and must be estimated in the RX. This is usually referred as timing synchronization. In LTE normal operation mode, the cyclic prefix duration is 4.7 µs. Thus, delays less than this are assumed:

```
delay = randi(940) % Delay in samples
x_BP = [zeros(1,delay), x_BP]; & Add 'delay' zeros
```

  NOTE: This should be added to the channel part before the noise is added to X_BP (see Sect. 3).

- Lastly, to consider the fact that the oscillator clocks (in the frequency translation with $e^{j\omega_c t}$ in the TX in Fig. 2 and with $e^{-j\omega_c t}$ in the RX in Fig. 3) are not synchronized, we define separate reference clock times for the TX and RX. Let's define the TX and RX clocks having random offsets uniformly distributed between 0…1s.
  NOTE: Modify the corresponding system parameters (see Sect. 2 and Sect. 4.1).

```
% Time vector for the RX oscillator signal:
TX_clock_start_time = rand; % Clock start time in the TX oscillator
RX_clock_start_time = rand; % Clock start time in the RX oscillator
```

  Then, the oscillator time vectors are *redefined* to use these offsets:

```
t_TX_oscillator = TX_clock_start_time + (0:Ts:Ts*(length(x_LP)-1));

t_RX_oscillator = RX_clock_start_time + (0:Ts:Ts*(length(y_BP)-1));
```

5.1 TIMING SYNCHRONIZATION

We implement an estimator for the correct symbol timing based on cross-correlation between the known training symbols and the received QAM signal.

NOTE: This should be performed after the channel and RX filtering but before the signal sampling (see Sect. 4).

- To perform the correlation between the known training symbols and the received QAM signal, we must upsample the training symbols in order match the sampling rates (like what we did in TX before the TX-filtering):

```
training_signal = zeros(size(1:r*length(training_symbols)));    % Zero vector
initilized for Up-sampled symbol sequence
training_signal(1:r:r*length(training_symbols)) = training_symbols;
% I.e. now the up-sampled sequence looks like {a1 0 0 a2 0 0 a3 0 0 a4 0 ...}
```

- Calculate the cross correlation as function time-delay between the received signal and the upsampled training symbols (help xcorr)

```
% Calculate the cross-correlation
[corr_fun, delay] = xcorr(X_LP_received,training_signal); %cross-correlation
```

  - Plot the amplitude of cross-correlation function

```
figure
plot(delay,abs(corr_fun))
xlabel('Delay [samples]')
ylabel('Correlation')
title('Cross-correlation between transmitted and received training symbols')
```

- Find the correlation peak to obtain the timing delay (help max):

```
% Find the sample index with the maximum correlation value:
[~, max_ind] = max(abs(corr_fun));

%Estimated delay: The 1st signal sample should be taken at "timing_ind+1"
timing_ind = delay(max_ind);
```

  Now, the sampling can be performed as

```
RX_symbol_frame = X_LP_received(timing_ind+1:r:end);
```

  where after the training symbols and data symbols are separated, as above.

5.2 PHASE ERROR ESTIMATION AND PHASE COMPENSATION

Since we have a set of training symbols, whose original phase is known by the receiver, we can estimate the phase error caused by the offset between TX and RX clocks. Basically, this is done by comparing the phases between the known training symbols and received training symbols.

- Plot the received symbol samples in complex plane (constellation) before phase error compensation to observe the effects of the error.

```
% Just copy here the same symbol plotting commands as above.
```

- Without going to details here, we use the following expression to calculate the so called channel estimate (if you are interested, you can see the *Appendix* in the end of the exercise for more details):

```
% Here RX_training_symbols and training_symbols should be row-vectors
channel_estimate = ... %with 3 dots the expression continues on the next line
    RX_training_symbols*training_symbols'/norm(training_symbols)^2; %help norm
```

- Then, the phase error can be compensated by simply dividing the signal with the channel estimate as

```
RX_data_symbols = RX_data_symbols/channel_estimate;
```

- Again, plot the received symbol samples in complex plane (constellation) to observe the compensation result.

```
% Just copy here the same symbol plotting commands as above.
```

Actually, the above channel compensation procedure would also compensate amplitude errors (based on so called zero-forcing principle).

5.3 SYMBOL DECISIONS AND CALCULATING THE SYMBOL ERROR RATE (SER)

Implement the remaining RX processing as above.

- Compare the realization of random channel delay to the estimated delay. What can you say about the accuracy?
- How the phase error effects the received constellation? How well is this error compensated?

APPENDIX (EXTRA MATERIAL) – CHANNEL ESTIMATION PRINCIPLE

Based on the AWGN channel, the received training symbol samples $\mathbf{y}$ (Nx1 vector) can be modeled as

$$\mathbf{y} = h\mathbf{x} + \mathbf{n} \ ,$$

where h is the channel coefficient (a complex scalar causing amplitude/phase error), $\mathbf{x}$ (Nx1 vector) is the transmitted (known) training symbols, and $\mathbf{n}$ (Nx1 vector) is the noise. Now, we calculate a (zero-shift) cross-correlation between $\mathbf{y}$ and $\mathbf{x}$ as

$$\mathbf{x}^H\mathbf{y} = \mathbf{x}^H h\mathbf{x} + \mathbf{x}^H\mathbf{n} = h\mathbf{x}^H\mathbf{x} + \mathbf{x}^H\mathbf{n} \, ,$$

where ^{H} is the complex-conjugate transpose. By noticing that $\mathbf{x}^H\mathbf{x} = \|\mathbf{x}\|^2$ the channel estimate $\hat{h}$ is obtained as

$$\hat{h} = \frac{\mathbf{x}^H\mathbf{y}}{\|\mathbf{x}\|^2} = h + \frac{\mathbf{x}^H\mathbf{n}}{\|\mathbf{x}\|^2} \ ,$$

where the last term defines the channel estimation error. Notice that when the number of training symbols (i.e. vector length) increases, the channel estimation error decreases.