ECOLE POLYTECHNIQUE FEDERALE DE LAUSANNE
School of Computer and Communication Sciences

Software-Defined Radio: A Hands-On Course
Assignment date: Sep 28, 2016
Due date: Oct 5, 2016

Your Own Communication Toolbox

Exercise 1. In this exercise you will implement a set of functions that make up an elementary communication system. These functions are a subset of the functionality that the MATLAB *Communication Toolbox* provides. Unless you are very fast, implementing every function in this assignment may be too much. For this reason, a few functions are labeled as optional. In general, any function that you implement in this exercise should rely only on *built-in* functions (as opposed to *toolbox* functions). To test if a function is in a toolbox you may type "which" followed by the function name. MATLAB will reply with the function location and will explicitly say if a function is built-in. (Built-in functions are in "$MATLAB_HOME/toolbox/matlab/...").

General guidelines: Implement your functions so that they behave exactly as specified and as shown in the examples. Your functions must check the supplied arguments and display an error if the value of a supplied argument does not make sense. For example, if `my_qammap` is called with an alphabet size `M` that does not correspond to a square QAM constellation, you might write

```
error('M must be in the form M = 2^(2K), where K is a positive integer.');
```

The general idea is to make your functions foolproof and user friendly.

1. Implement the function `my_qammap` defined as follows:

   ```
   % MY_QAMMAP Creates constellation for square QAM modulations
   % C = MY_QAMMAP(M) outputs a 1×M vector with the
   % constellation for the quadrature amplitude modulation of
   % alphabet size M, where M is the square of an integer power
   % of 2 (e.g. 4, 16, 64, ...).
   % The signal constellation is a square constellation.
   ```

   Your function must use the modulation schemes exactly as shown on Figure 1.
   Hint: you may want to check the help for function `meshgrid`.

   Example:

   ```
   >> my_qammap(4)
   ans =

     -1.0000 + 1.0000i  -1.0000 - 1.0000i   1.0000 + 1.0000i   1.0000 - 1.0000i
   ```

   If the function is called with invalid arguments, for example if `M` is not the square of an integer power of 2, an error must be displayed.

2. Implement the function `my_pskmap` defined as follows:

   ```
   % MY_PSKMAP Creates constellation for Phase Shift Keying modulation
   % C = MY_PSKMAP(M) outputs a 1×M vector with the complex symbols
   % of the PSK constellation of alphabet size M, where M is an integer power of 2.
   ```

   Your function must use the modulation scheme exactly as shown on Figure 2.

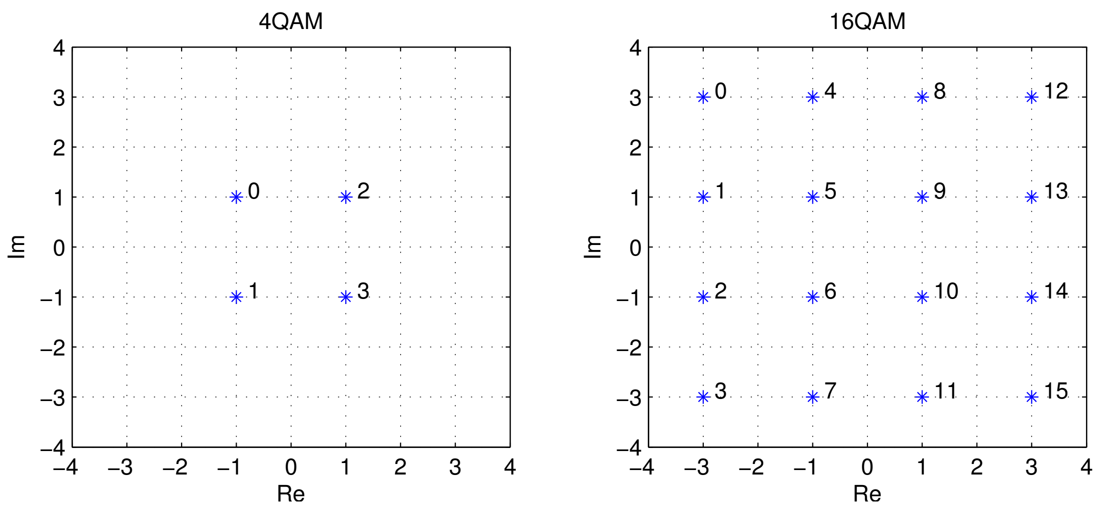

Figure 1: 4-QAM and 16-QAM constellations. The numbers show the correspondence between constellation points and message symbols.

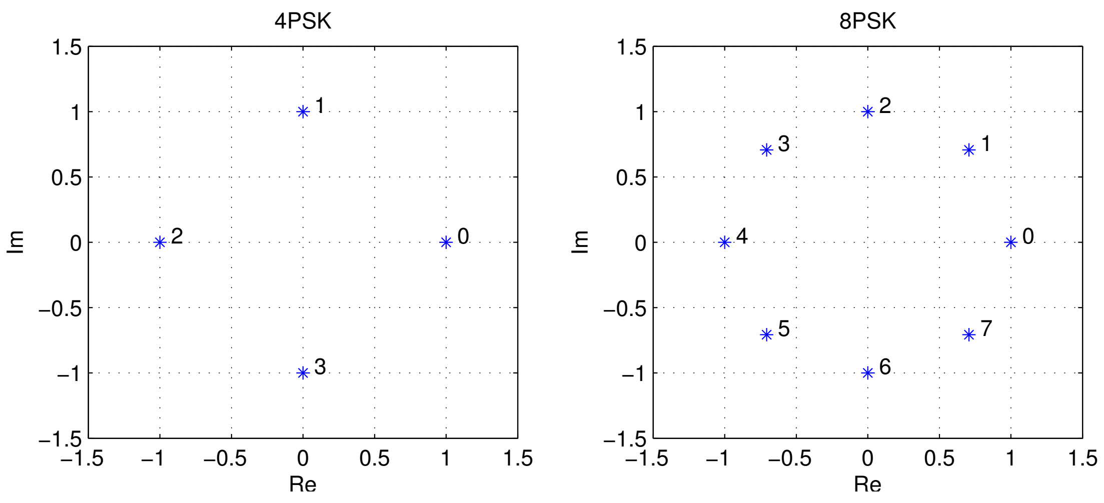

Figure 2: 4-PSK and 8-PSK constellations. The numbers show the correspondence between constellation points and message symbols.

Example:

```
>> my_pskmap(4)
ans =

  1.0000             0.0000 + 1.0000i  -1.0000 + 0.0000i  -0.0000 - 1.0000i
```

If the function is called with invalid arguments, for example if `M` is not an integer power of 2, an error must be displayed.

3. Implement the function `my_modulator` defined as follows:

```
% MY_MODULATOR Maps a vector of M-ary integers to constellation points
% Y = MY_MODULATOR(X, MAP) outputs a vector of (possibly complex)
% symbols from the constellation specified as second parameter,
% corresponding to the integer valued symbols of X.
% Input X can be a row or column vector, and output Y has the same
% dimensions as X.
% If the length of MAP is M, then the message symbols of X
% must be integers between 0 and M-1.
```

If the function is called with invalid arguments, for example if any element of `X` is not between 0 and `length(MAP)-1`, an error must be displayed.

Example: 4-QAM modulation can be achieved combining the use of the `MY_QAMMAP` and `MY_MODULATOR` functions:

```
>> my_modulator([0 2 1 3], my_qammap(4))
ans =

  -1.0000 + 1.0000i   1.0000 + 1.0000i  -1.0000 - 1.0000i   1.0000 - 1.0000i
```

4. Implement the function `my_demodulator` defined as follows:

```
% MY_DEMODULATOR Minimum distance slicer
% Z = MY_DEMODULATOR(Y, MAP) demodulates vector Y
% by finding the element of the specified constellation that is
% closest to each element of input Y. Y contains the outputs of
% the matched filter of the receiver, and it can be a row or
% column vector. MAP specifies the constellation used, it can
% also be a row or column vector.
% Output Z has the same dimensions as input Y.
% The elements of Z are M-ary symbols, i.e., integers between
% 0 and M=length(MAP)-1.
```

Example:

```
>> z = my_demodulator([-3+3i, -3+0.7i, 1.1-0.9i, 2.8 + 1.7i], my_qammap(16))
z =

     0     1    10    13
```

Hint: notice than the function `min` can be called with two output arguments.
Hint: there are multiple ways to implement this function, but you may want to check the help for command `repmat` or `meshgrid`.

5. Read the documentation for the Matlab function `firrcos` (alternatively, you can use the functions `rcosine` or `rcosdesign`), which will allow you to design a root raised cosine filter. We will use it as shaping filter (basic pulse) in the transmitter, and as matched filter in the receiver in the last exercise of this assignment. Notice that the impulse response of the root raised cosine filter is infinite in length, so we will be using a truncated FIR approximation. Observe also that at this point you need to fix the relationship between the symbol period and the sampling period, to which we shall refer as *upsamling factor*.

Note: If you have time, we recommend that you "play" with the `firrcos` function (or `rcosine` / `rcosdesign`) to verify that you get what you expect. For instance, the help says that "The coefficients of B are normalized so that the nominal passband gain is always equal to one." From this information you should be able to figure out by how much you need to scale the coefficients to obtain a normalized pulse.

- Get the impulse response of the root raised cosine filter for different values of the rolloff factor (for instance $\beta = 0,\ 0.5,\ 1$). Observe the effect of β both on the plots of the impulse response and of the frequency response.
- Normalize your pulse and verify that it is orthogonal to its shift by appropriate amounts.

Hint about the order: For a FIR filter, the order is the order of its transfer-function polyomial. Hence it is the length of the impulse response minus 1. For instance, if the impulse response is $b_1, b_2, b_3 = -1, 0, 2$ then the transfer-function polynomial is $b_1 + b_2 z^{-1} + b_3 z^{-2}$ and the order is 2.

6. Implement the function `my_symbols2samples` defined as follows:

```
% MY_SYMBOLS2SAMPLES Produces the samples of the modulated signal
% Z = MY_SYMBOLS2SAMPLES(Y, H, USF) produces the samples of a
% modulated pulse train. The sampled pulse is given in vector H,
% and the symbols modulating the pulse are contained in vector Y.
% USF is the upsampling factor, i.e., the number of samples per symbol.
% It is the the ratio between the sampling frequency (Fs)
% and the symbol frequency (Fd), USF=Fs/Fd.
```

Hint: `Z` is conveniently obtained by the discrete-time convolution of a modified symbol sequence and `H`. The modified symbol sequence is obtained from the actual symbol sequence `Y` by inserting USF-1 zeros between each sample. You can use the function `upsample` to introduce zeros in between any two elements of `Y`.
Hint: use `conv` or `filter` to generate the samples of the modulated signal from the upsampled symbol vector.

7. Implement the function `my_sufficientstatistics` defined as follows:

```
% MY_SUFFICIENTSTATISTICS Processes the output of the channel to generate
% sufficient statistics about the transmitted symbols.
% X = MY_SUFICIENTSTATISTICS(R, H, USF) produces sufficient statistics
% about the transmitted symbols, given the signal received in vector R,
% the impulse response H of the basic pulse (transmitting filter), and
% the integer USF, which is the ratio between the sampling rate and the
% symbol rate (upsampling factor)
```

To put things into perspective, recall that there are two discrete-time channel models: the one that operates at the sample level and the one that operates at the symbol level. This function takes the sample-level channel output and delivers the symbol-level channel output.
Conceptually, this function implements inner products, but it is more efficient to do a matched filter implementation.

8. **Optional:** Implement the function `my_bi2de` defined as follows:

```
% MY_BI2DE Converts binary vectors to decimal numbers
%    D = MY_BI2DE(B) converts a binary vector B to a decimal value D. When
%    B is a matrix, the conversion is performed row-wise and the output D
%    is a column vector of decimal values. The default orientation of the
%    binary input is Right-MSB: the first element in B represents the least
%    significant bit.
%
%    In addition to the input matrix, an optional parameter MSBFLAG can be
%    given:
%
%    D = MY_BI2DE(B, MSBFLAG) uses MSBFLAG to determine the input
%    orientation. MSBFLAG has two possible values, 'right-msb' and
%    'left-msb'. Giving a 'right-msb' MSBFLAG does not change the
%    function's default behavior. Giving a 'left-msb' MSBFLAG flips the
%    input orientation such that the MSB is on the left.
```

You should not use MATLAB functions `bi2de` or `bin2dec`.

If the function is called with invalid arguments, for example if the argument `B` has values other than 1 or 0, or if `MSBFLAG` is an invalid string, then an error must be displayed.

Examples:

```
>> my_bi2de([0 1 0 1])
ans =
    10

>> my_bi2de([0 1 0 1; 1 0 1 0])
ans =
    10
     5

>> my_bi2de([1 1 0; 1 0 1; 0 0 1], 'left-msb')
ans =
     6
     5
     1
```

9. **Optional:** Implement the function `my_de2bi` defined as follows:

```
% MY_DE2BI Converts decimal numbers to binary numbers
%    B = MY_DE2BI(D) converts a vector D of nonnegative integers from base 10
%    to binary matrix B. Each row of the binary matrix B corresponds to one
%    element of D. The default orientation of the binary output is
%    Right-MSB, i.e., the first element in a row of B represents the least
%    significant bit. If D is a matrix rather than a row or column vector, the
%    matrix is first converted to a vector (column-wise).
%
%    In addition to the vector input, two optional parameters can be given:
%
%    B = MY_DE2BI(D,MSBFLAG) uses MSBFLAG to determine the output
%    orientation. MSBFLAG has two possible values, 'right-msb' and
%    'left-msb'. Giving a 'right-msb' MSBFLAG does not change the
%    function's default behavior. Giving a 'left-msb' MSBFLAG flips the
%    output orientation to display the MSB to the left.
%
%    B = MY_DE2BI(D,MSBFLAG,N) uses N to define how many binary digits (columns)
%    are output. The number of bits must be large enough to represent the
%    largest number in D.
```

You should not use MATLAB functions `de2bi` or `dec2bin`.

Hint: you can use `bitshift` and `bitand` to efficiently implement this function.
Hint: you can implement this function without using any loops.

Examples:

```
>> my_de2bi([12 5])
ans =
     0     0     1     1
     1     0     1     0

>> my_de2bi([12 5], 'right-msb', 5)
ans =
         0       0       1       1       0
         1       0       1       0       0

>> my_de2bi([12 5], 'left-msb')
ans =
         1       1       0       0
         0       1       0       1
```

If the function is called with invalid arguments, for example if `N` is too small to represent the largest number in `D`, or if `MSBFLAG` is an invalid string, then an error must be displayed[1].

Exercise 2. Using the functions you have written for the different parts of Exercise 1, write a script that simulates the transmission of a sequence of bits over the waveform channel with additive white Gaussian noise. The script must also implement the receiver side, and compute at the end the BER (Bit Error Rate) and SER (Symbol Error Rate) of the transmission.

Design the RRC filter as in Part 5 of Exercise 1, using a roll-off factor of $\beta = 0.22$.

At the beginning of the script the user should be able to easily configure the following parameters:

- Number of bits N to transmit[2].
- Modulation type and order (4-QAM, 8-PSK, . . .)
- Value of E_s/σ^2 (ratio of symbol energy to noise variance at the matched filter output)

If you have not completed the optional `my_bi2de` and `my_de2bi` functions, you can use those provided by Matlab, `bi2de` and `de2bi`.

Hint: To generate the random bit vector to be transmitted, you can use the `randi` function.
Hint: To simulate the AWGN channel, you may use the `awgn` function from Matlab Communication Toolbox or, if not available, the standard Matlab function `randn`. In this last case you will need two calls to `randn` to generate the real and imaginary parts of the complex noise vector, and you will need to multiply the output by an appropriate factor so that the resulting variance is in accordance with the desired value of E_s/σ^2.
Hint: The ratio of symbol energy to noise variance at the matched filter output is not the same as the ratio of symbol energy to noise variance at the matched filter input!
Hint: You can validate your implementation in several ways. For QPSK/4-QAM, you can use the exact formulas[3] $P_b = Q(x)$ and $P_s = 2Q(x) - Q(x)^2$ to compute respectively the BER and SER, where $x = \sqrt{E_s/\sigma^2}$. For other modulation schemes, you can use the `berawgn` function from Matlab Communication Toolbox[4].
Hint: Plotting the constellation of received symbols at the output of the matched filter, after downsampling, can help to debug problems with your implementation. Use a fairly high value of E_s/σ^2.

In order to help you with the implementation, we provide on the course website the compiled versions of the functions you have to write.

[1]using the function `error`

[2]use $N > 10^5$ or $N > 10^6$ to get accurate approximation of the true BER down to 10^{-4} or 10^{-5}. Initially use smaller values of N to more easily debug your implementation.

[3]Older versions of Matlab do not have an implementation of the $Q(\cdot)$ function, but you can use the complementary error function and the relationship $Q(x) = \frac{1}{2}\operatorname{erfc}\left(x/\sqrt{2}\right)$. In newer versions of Matlab with the Communication Toolbox installed you can directly use `qfunc`.

[4]You should be able to reproduce exactly the SER results. Except for QPSK/4-QAM, your results of BER will be slightly worse that the theoretical results reported by `berawgn` because this function assumes a Gray mapping of bits to symbols, while we are using a different suboptimal mapping