Simulating Communications Systems in MATLAB

Qi Wang

October 13, 2010

MATLAB is an interactive system for doing numerical computations. It is widely used to simulate communications systems. General introductions of MATLAB can be easily found by typing `matlab introduction` in any search engine. In this document, we will give a brief introduction on how to simulate a communication system.

1 Before the Start

Help and information on MATLAB commands can be found in several ways:

- from the command line by using the command `help <function name>`:

```
>> help sqrt
 SQRT   Square root.
    SQRT(X) is the square root of the elements of X. Complex
    results are produced if X is not positive.

    See also sqrtm, realsqrt, hypot.

    Overloaded methods:
       codistributed/sqrt
       sym/sqrt

    Reference page in Help browser
       doc sqrt

>>
```

- from the product Help window found under the menu Help or using the command `doc` from the command line.
- placing the cursor on a command of interest and pressing `F1`.

2 Calculation, Variables, Vectors, Matrices

In this section, we provide a list of notations that are commonly used in MATLAB . It will be easier to follow if you have MATLAB by side and just learn by doing. **You should type in commands shown below after the prompt: >>.**

2.1 Matlab as a Calculator

```
>> 2+3^2+4*(5-8)*sin(0.1)
ans =
    9.8020
```

Other built-in functions for calculation include `abs, sqrt, exp, log, log10, sin, cos, ...`

2.2 Variables

The result of the calculation is assigned to the variable `ans` by default. Commands are separated by a ",". The output of commands is supressed by a ";".

Legal variable names consist of any combination of letters and digits, starting with a letter. However, you should avoid using already predefined names such as `eps, pi, i, j`.

```
>> eps, pi, i, j, 3+5, a=3+5, b=3+5;
ans = 2.2204e-016
ans = 3.1416
ans = 0 + 1.0000i
ans = 0 + 1.0000i
ans =
     8
a =
     8
>>
```

In the exercises, you are required to use names that represent the meaning of the variables.

2.3 Vectors and Matrices

When defining a vector, entries must be enclosed in square brackets. Row elements are separated with a space or a ",". Column elements are separated with a ";".

```
>> a = [1 5 3 8 -4]           %% Comments are written like this.
a =
     1     5     3     8    -4

>> a = [1, 5, 3, 8, -4]       %% a row vector
a =
     1     5     3     8    -4

>> a = [1; 5; 3; 8; -4]       %% a column vector
a =
     1
     5
     3
     8
    -4

>> length(a)                  %% get the length of the vector
ans =
     5

>> b = rand(5,1)              %% define a vector with uniformly
                              %% distributed entries.
b =
    0.8147
    0.9058
    0.1270
    0.9134
    0.6324

>> c = a + j*b                %% build a complex-valued vector
c =
   1.0000 + 0.8147i
   5.0000 + 0.9058i
   3.0000 + 0.1270i
   8.0000 + 0.9134i
  -4.0000 + 0.6324i
```

```
>> d = c.'                     %% transpose
d =
   1.0000 + 0.8147i   5.0000 + 0.9058i   3.0000 + 0.1270i
 8.0000 + 0.9134i  -4.0000 + 0.6324i

>> e = conj(d)                 %% conjugate
e =
   1.0000 - 0.8147i   5.0000 - 0.9058i   3.0000 - 0.1270i
 8.0000 - 0.9134i  -4.0000 - 0.6324i

>> f = c'                      %% Hermitian -- transpose + conjugate
f =
   1.0000 - 0.8147i   5.0000 - 0.9058i   3.0000 - 0.1270i
 8.0000 - 0.9134i  -4.0000 - 0.6324i

>> e == f                      %% logical operations (>,<,~=,>=,&,|)
                               %% zero--false or one--true

ans =
     1     1     1     1     1

>> norm(f)                     %% norm of a vector
ans =
   10.8506
>> help norm                   %% check different norms

>> k = ones(5,3)               %% define an all-one matrix with
                               %% specified size
                               %% same with zeros() and nan()
k =
     1     1     1
     1     1     1
     1     1     1
     1     1     1
     1     1     1

>> size(k)                    %% return the size of a matrix
                              %% size(k,1) to return only the first dimension
ans =
     5     3
```

```
>> k(5,2)                       %% the element at 5th row, 2nd column in k
ans =
     1

>> k(5,:)                       %% the 5th row in k
ans =
     1     1     1

>> k(end,:)                     %% the last row
ans =
     1     1     1

>> k(1:3:end,:)                 %% every 3th row
ans =
     1     1     1
     1     1     1

>> rank(k)                      %% rank of a matrix
ans =
     1

>> n = eye(3)                   %% an identity matrix
n =
     1     0     0
     0     1     0
     0     0     1

>> diag(n)                     %% diagonal elements of a matrix
ans =
     1
     1
     1

>> trace(n)                    %% trace of a matrix
ans =
     3

>> p = d*k                      %% product of two matrices
                                %% dimensions of the two have to match.
p =
```

```
  13.0000 + 3.3933i  13.0000 + 3.3933i  13.0000 + 3.3933i
```

```
>> clc                           %% clear the command window
>> clear                         %% clear the workspace
```

3 Batch Files and Functions

[1] If you don't want the current result of a command at the spot, the line has to be ended with ";". In order to execute a number of command at once, we write them in a m-file.

- Open the editor using the command `edit` or from the menu bar.
- Write your command line by line and save as `<yourfilename>.m`.
- Press `F5` or type `<yourfilename>` in the command window.

To define a function, the procedure is similar, check the product Help.

4 A Typical Communications System

In a communications system, messages are transmitted over a certain channel. An example is shown in Fig. 1.

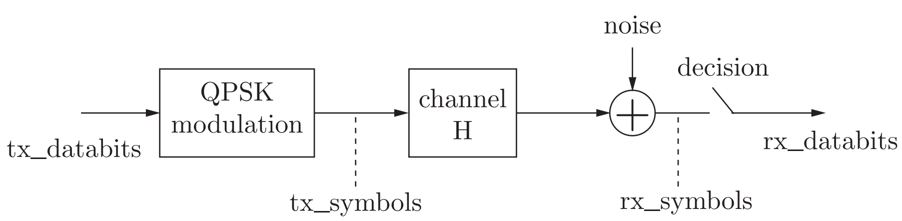

Figure 1: a typical transmission system

In the following, an example will be given to show how each step is implemented in MATLAB .

[1]If you already have some experience with MATLAB , this section can be skipped.

Table 1: **QPSK symbol mapping**

integer number	0	1	2	3
LSB	0	1	0	1
MSB	0	0	1	1
symbol alphabet	$\frac{1}{\sqrt{2}}(1+j)$	$\frac{1}{\sqrt{2}}(-1+j)$	$\frac{1}{\sqrt{2}}(1-j)$	$\frac{1}{\sqrt{2}}(-1-j)$

4.1 Generate Databits

The data information is a stream of binary bits. The occurrence of `0` and `1` are of equal probability. Assume a data stream of length `50000`, it can be generated in MATLAB as shown below:

```
tx_databits = round(rand(1,50000));
```

The function `rand(m,n)` returns a matrix of size $m \times n$ with its entries uniformly distributed within the interval $[0,1]$. The operation `round` rounds the arguments to the nearest integers, resulting a vector of integer number 0 and 1.

4.2 QPSK Modulation

For QPSK modulation, the mapping pattern is shown in Table. 1. It can be implemented in MATLAB as shown below:

```
symbol_alphabet = [ 1+1j, -1+1j, 1-1j, -1-1j]/sqrt(2);
M = log2(length(symbol_alphabet));
symbols_int = 2.^[0:M-1]*reshape(tx_databits, M, []);
tx_symbols = symbol_alphabet(symbols_int+1);
```

In this example, the power per symbol σ_s^2 is normalized to 1.

4.3 Noise Definition

In most of the simulations, the noise is assumed to be complex-valued additive white Gaussian. The function `randn(m,n)` returns a $m \times n$ matrix of real numbers which follows the standard normal distribution $N(0,1)$. Therefore, a complex-valued Gaussian vector with average power per symbol `1` can be built by `(randn(m,n) + j*randn(m,n))/sqrt(2)`.

In order to plot a Bit-Error-Ratio (BER)/Signal-to-Noise (SNR) curve, the transmission scheme is simulated for different SNR level. This is usually done by fixing the transmitted symbol power to 1 and varying the average noise power σ_n^2 according to SNR levels.

The SNR utilizes a dB notation and can be defined as shown below:

$$\text{SNR in dB} = 10\log_{10}\frac{\sigma_s^2}{\sigma_n^2} = 20\log_{10}\frac{|\sigma_s|}{E\{|\sigma_n|\}}.$$

Therefore, the average amplitude of the noise realization can be found by

$$E\{|\sigma_n|\} = 10^{-\frac{\text{SNR in dB}}{20}},$$

which will be applied to the noise definition.

In summary, the noise definition part is implemented as shown below:

```
SNR = 20;   % an example: SNR = 20 dB
sigma_v = 10^(-SNR/20);
noise = sigma_v*(randn(size(tx_symbols))
                        +j*randn(size(tx_symbols)))/sqrt(2);
```

4.4 Channel

In this example, the channel is assumed to be AWGN, which is implemented as following:

```
rx_symbols = tx_symbols + noise;
```

4.5 Check the Scatterplot using `scatter` (optional)

You can check the constellation of the received symbol by

```
scatter(real(rx_symbols), imag(rx_symbols));
```

Sometimes, this is a useful tool for debugging.

4.6 Demodulation by a Slicer

For demodulation, we consider hard decision made by a slicer, which is implemented as shown below:

```
rx_databits_temp = [real(rx_symbols)<0; imag(rx_symbols)<0];
rx_databits = reshape(rx_databits_temp, 1, []);
```

4.7 Calculate Bit Error Ratio

The Bit Error Ratio (BER) is defined as

$$\text{BER} = \frac{\text{number of error bits}}{\text{number of transmit bits}},$$

which is calculated for each SNR value. It can be implemented in MATLAB as

```
bit_errors = sum(tx_databits ~= rx_databits);
ber = bit_errors/length(rx_databits);
```

4.8 Loop over SNRs

Up to Section 4.7, the transmission is complete for one SNR value. In order to obtain the BER for different SNR levels, a loop structure using `for` command is shown below.

```
% define a SNR vector you want to simulate
SNR_vec = -10:2:20;

% assign a vector for bit error ratio result.
% note that is very important to predefine the vector in which
% the results are stored. otherwise, if the vector is large, it
% is increased in size while running the loop, which can be
% incredibly slow.

ber = nan(1,length(SNR_vec));


% main loop over different SNR value
for ii = 1:length(SNR_vec)      % Attention: do not use i,j as loop index!
                                % because i^2=-1 by default.

    % specify the SNR for current loop
    SNR = SNR_vec(ii);

    ... ...

    ber(ii) = bit_errors/length(rx_databits);
end
```

4.9 Show the Plot

After the simulation over different SNR values, a vector of BER is obtained with respect to the SNR vector previously defined. A simple plot can be generated using the commands below. BER curves are usually plotted in logarithmic scale by `semilogy`.

```
figure
semilogy(SNR_vec, ber, 'b-')
xlabel('SNR [dB]')
ylabel('Bit Error Ratio')
title('QPSK AWGN')
grid on
```

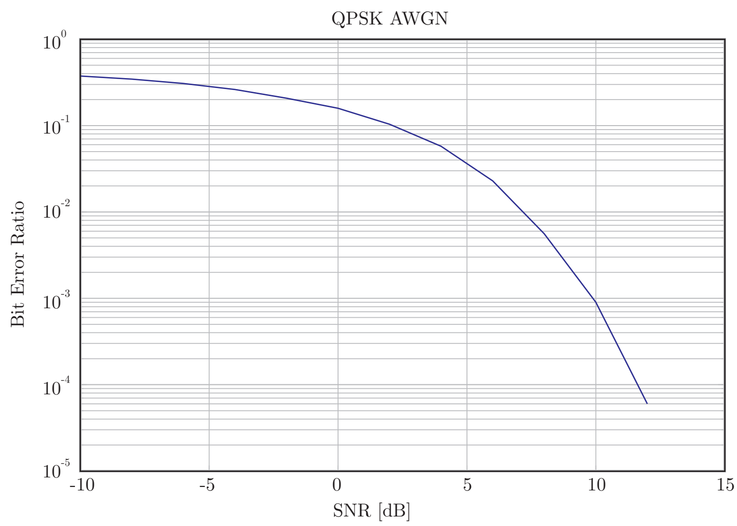

Figure 2: a Bit Error Ratio curve

If you want to plot more than one curve in one figure, use the command `hold`. For more details, read the help file of the command `plot`.