Part 1 Noise Free:

We’re going to test if our Tx and Rx are working correctly before adding any noise:

The modulation techniques tested are:

BPSK QPSK 8PSK 16-QAM

So we made a function based-code:

Tx Mapper:

Code:

function [Tx\_Vector, Table] = mapper(bits, mod\_type)

% MAPPER Digital modulation mapper with explicit symbol table

% Inputs:

% bits - Binary input array (row vector)

% mod\_type - 'BPSK', 'QPSK', '8PSK', 'BFSK', '16QAM'

% Outputs:

% Tx\_Vector - Complex modulated symbols

% Table - Constellation points (M-ary symbols)

% Ensure bits are row vector

bits = bits(:)';

% Define modulation parameters

switch upper(mod\_type)

case 'BPSK'

n = 1; % bits per symbol

M = 2; % constellation size

Table = [-1, 1]; % BPSK symbols (real)

case 'QPSK'

n = 2;

M = 4;

Table = [-1-1j, -1+1j, 1-1j, 1+1j]; % QPSK symbols

case '8PSK'

n = 3;

M = 8;

angles =[0, 1, 3, 2, 7, 6, 4, 5]\*pi/4; % Gray-coded 8PSK

Table = exp(1j\*angles);

case 'BFSK'

error('BFSK requires time-domain implementation (see alternative)');

case '16-QAM'

n = 4;

M = 16;

% 16-QAM with unit average power (normalized)

Table = [-3-3j, -3-1j, -3+3j, -3+1j, ...

-1-3j, -1-1j, -1+3j, -1+1j, ...

3-3j, 3-1j, 3+3j, 3+1j, ...

1-3j, 1-1j, 1+3j, 1+1j];

otherwise

error('Unsupported modulation type: %s', mod\_type);

end

% Pad bits if not multiple of n

if mod(length(bits), n) ~= 0

bits = [bits zeros(1, n - mod(length(bits), n))];

end

% Reshape into n-bit groups

bit\_groups = reshape(bits, n, [])';

% Convert to decimal symbols (0 to M-1)

Array\_symbol = bi2de(bit\_groups, 'left-msb') + 1; % MATLAB uses 1-based indexing

% Map to constellation points

Tx\_Vector = Table(Array\_symbol);

end

For the Tx mapper, we just convert the bits into decimal values to index it with symbol table, which is grey-coded, from the complex constellations:

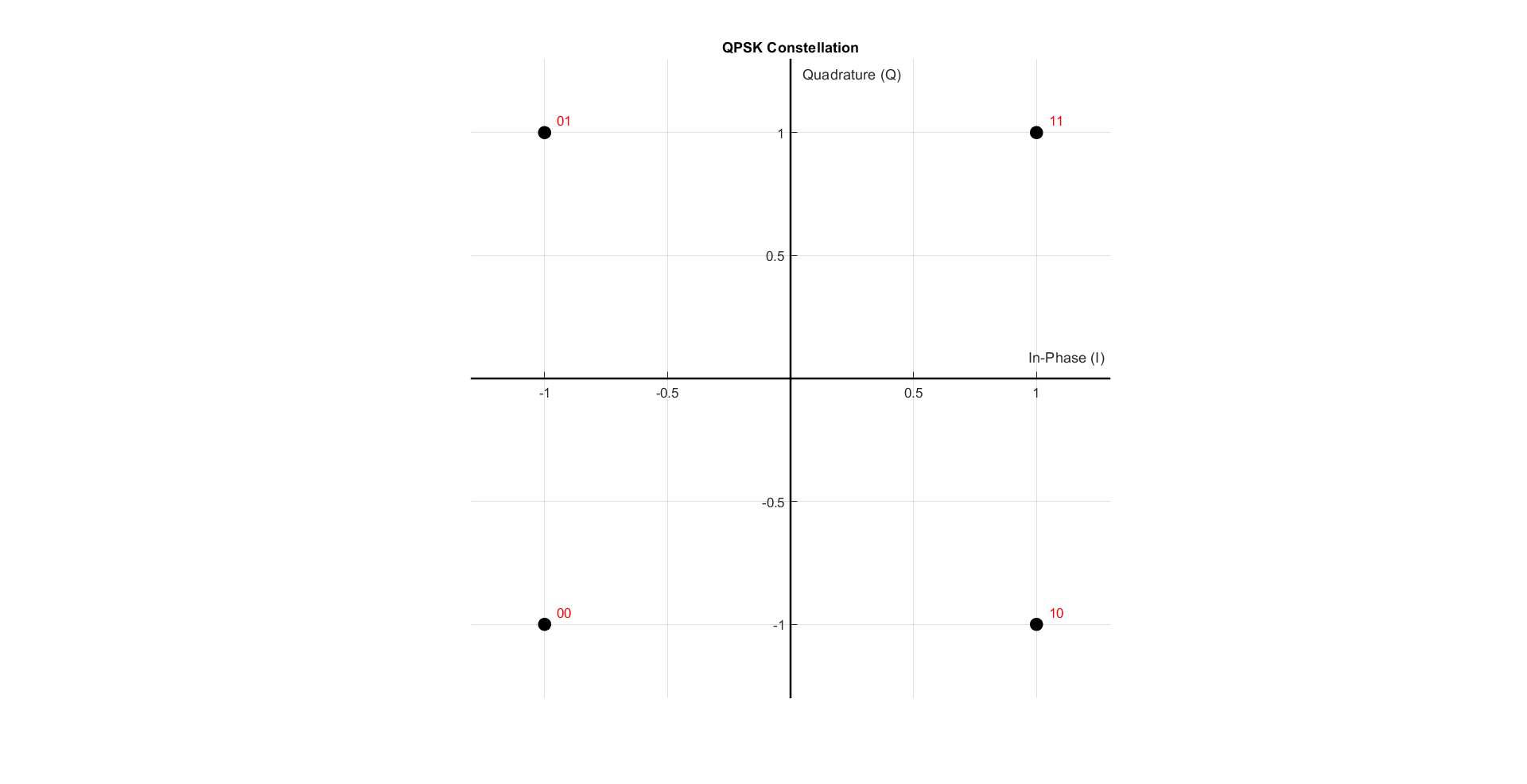
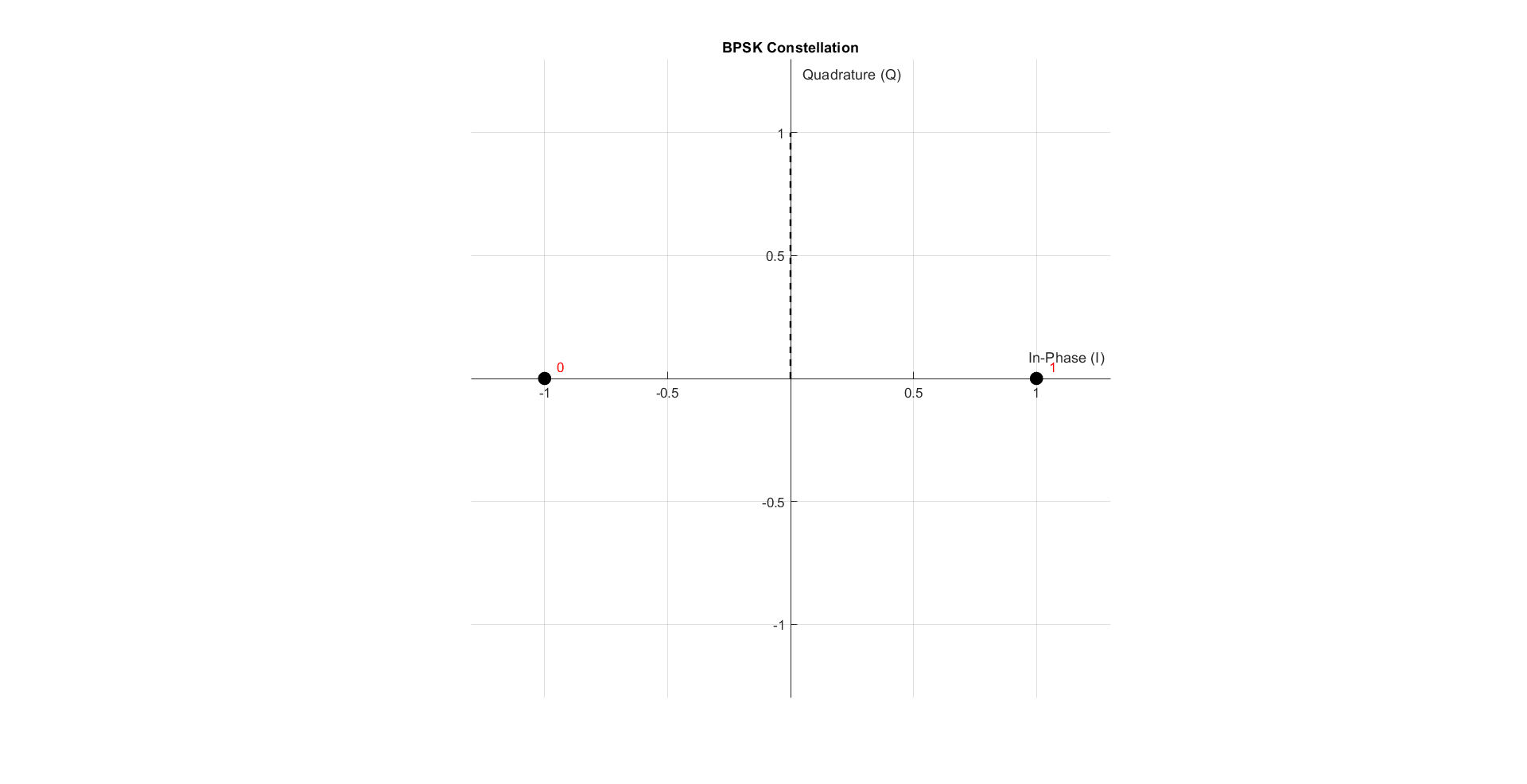


Figure 1 BPSK constellation

Figure 2 QPSK constellation

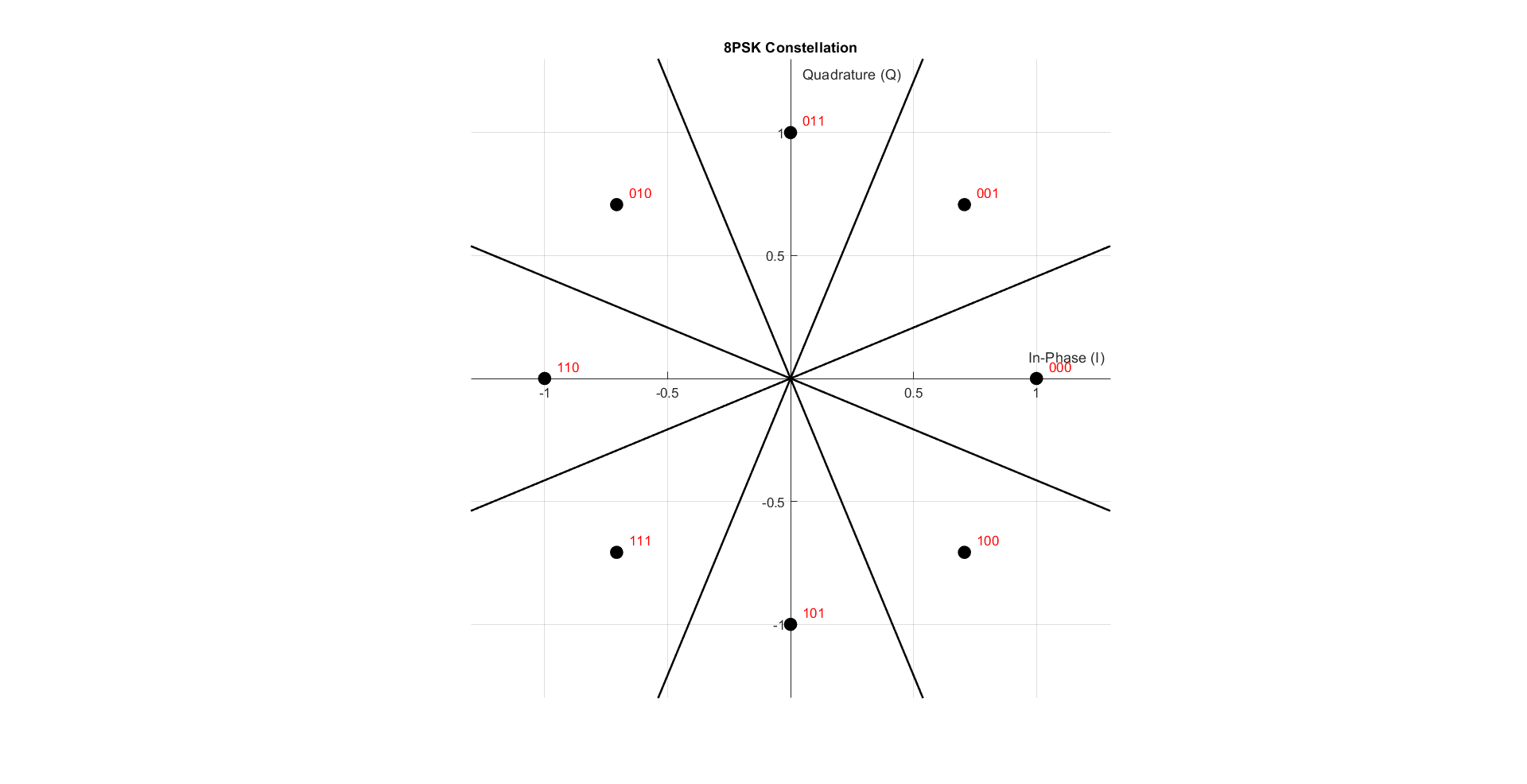
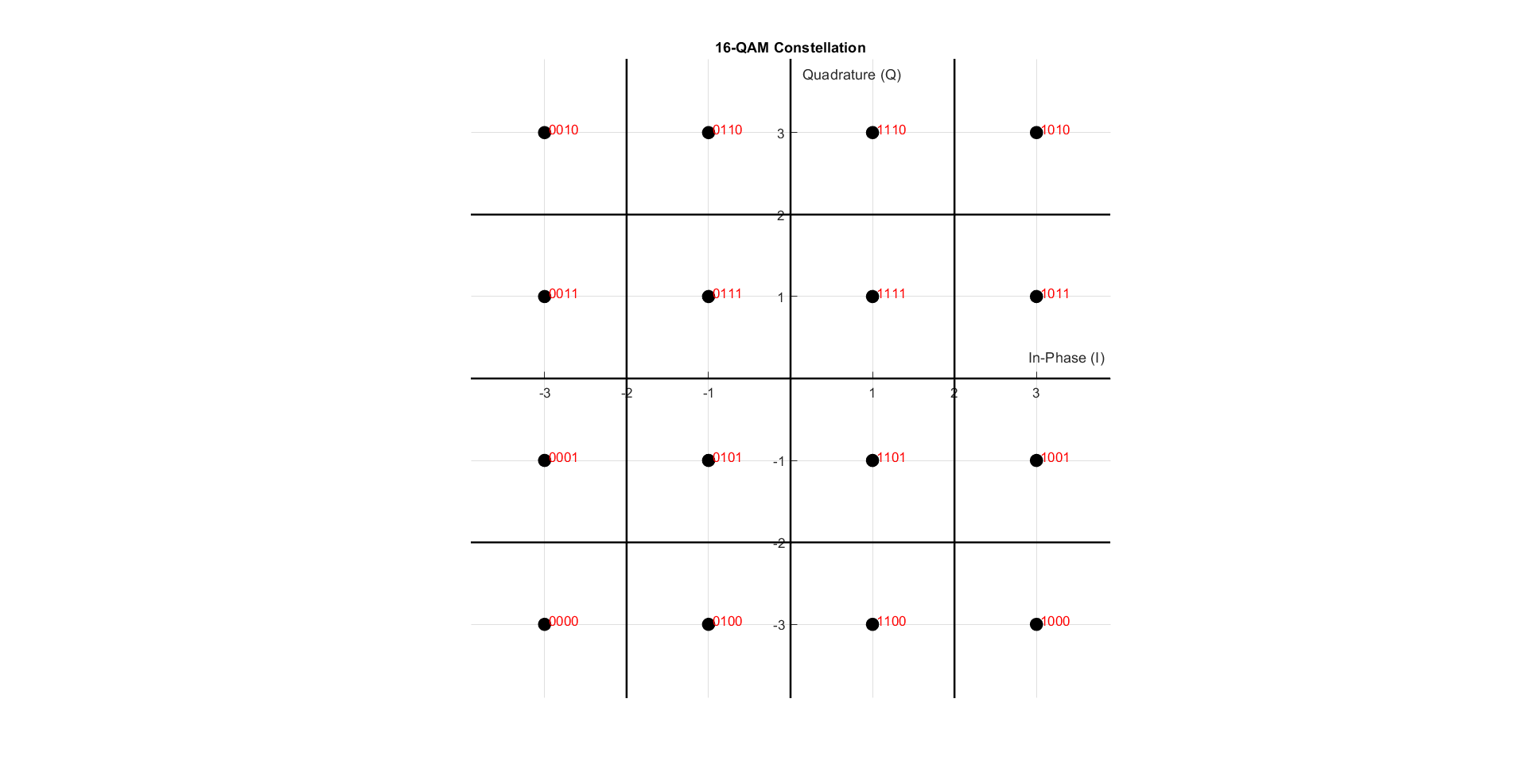


Figure 3 16-QAM constellation

Figure 4 8PSK constellation

As shown in the figures 1, 2, 3, and 4, we just make some linear algebra operations. As the I is the real part and Q is the imaginary part

XBB = XI + j XQ

Rx Demapper:

function [received\_bits] = demapper(received\_symbols, mod\_type)

% DEMAPPER Digital demodulation demapper

% Inputs:

% received\_symbols - Complex received symbols

% mod\_type - Modulation type ('BPSK', 'QPSK', etc.)

% Output:

% received\_bits - Demodulated bit stream

% Get constellation table from mapper

[~, Table] = mapper([1], mod\_type);

% Determine bits per symbol

switch upper(mod\_type)

case 'BPSK'

n = 1;

case 'QPSK'

n = 2;

case '8PSK'

n = 3;

case {'16QAM', '16-QAM'}

n = 4;

otherwise

error('Unsupported modulation type');

end

% Initialize output bits

received\_bits = zeros(1, length(received\_symbols)\*n);

% Demodulate each symbol

for i = 1:length(received\_symbols)

% Find nearest constellation point

[~, idx] = min(abs(received\_symbols(i) - Table));

% Convert to binary (0-based index)

bin\_str = dec2bin(idx-1, n);

% Store bits

received\_bits((i-1)\*n+1:i\*n) = bin\_str - '0';

end

end

For the Rx demapper, we just make inverse Tx mapper operation.

We check the nearest table symbol to the Rx symbol and get it’s index with this index we convert it into bits.

Simulation:

Now we will try a small noise free simulation to make sure that the Rx and Tx runs properly

Code:

clear; clc; close all;

%--------Part 1----------

% ========================

% Simulation Parameters

% ========================

bits\_Num = 48; % Number of bits to transmit

mod\_types = {'BPSK', 'QPSK', '8PSK', '16-QAM'}; % Cell array of modulation types

% Generate random bits (same for all modulations for fair comparison)

Tx\_bits = randi([0 1], 1, bits\_Num);

% Loop through all modulation types

for mod\_idx = 1:length(mod\_types)

mod\_type = mod\_types{mod\_idx};

fprintf('\n=== Testing %s Modulation ===\n', mod\_type);

% ========================

% 1. Mapping (Modulation)

% ========================

[tx\_symbols, constellation] = mapper(Tx\_bits, mod\_type);

% ========================

% 2. Display Constellation

% ========================

drawConstellation(constellation, mod\_type);

title(sprintf('%s Constellation', mod\_type));

% ========================

% 3. Add Channel Noise

% ========================

%rx\_symbols = awgn(tx\_symbols, SNR\_dB, 'measured');

rx\_symbols = tx\_symbols;

% ========================

% 4. Demapping (Demodulation)

% ========================

Rx\_bits = demapper(rx\_symbols, mod\_type);

% ========================

% 5. Display Results

% ========================

% Calculate BER

[BER, bit\_errors] = calculateBER(Tx\_bits, Rx\_bits);

% Display input/output comparison

fprintf('Original bits:\n');

disp(reshape(Tx\_bits, 16, [])'); % Display in 16-bit groups

fprintf('Received bits:\n');

disp(reshape(Rx\_bits(1:bits\_Num), 16, [])'); % Display in 16-bit groups

fprintf('Bit errors: %d\n', bit\_errors);

fprintf('BER: %.2e\n\n', BER);

end

In the simulation we’ll generate random bits and modulate it with each type and check if there’s an error

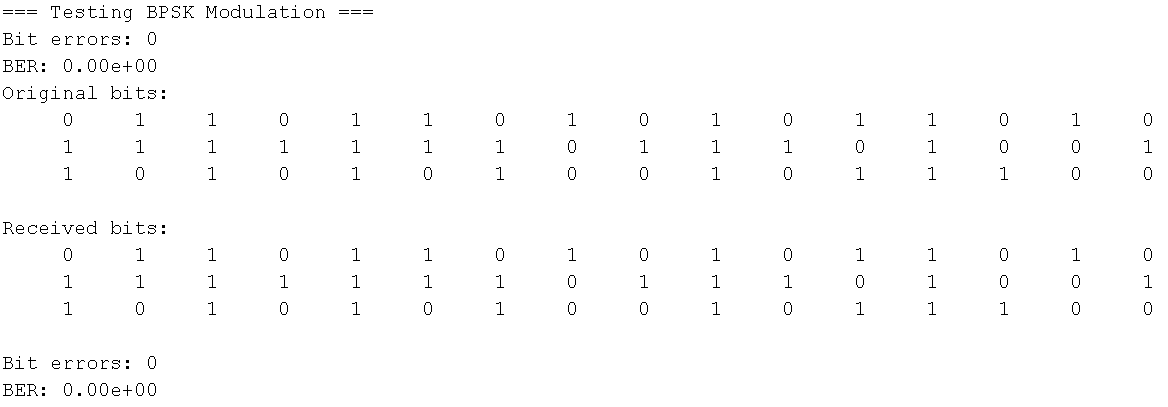
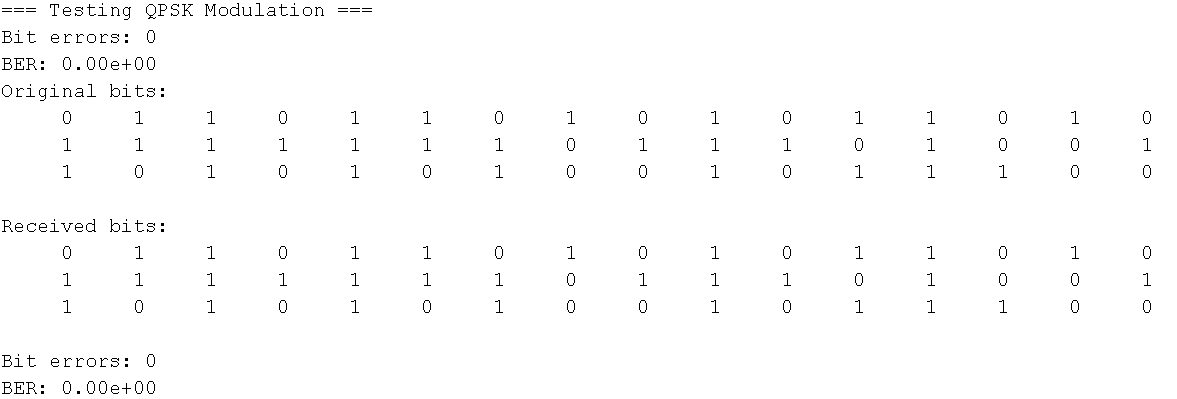
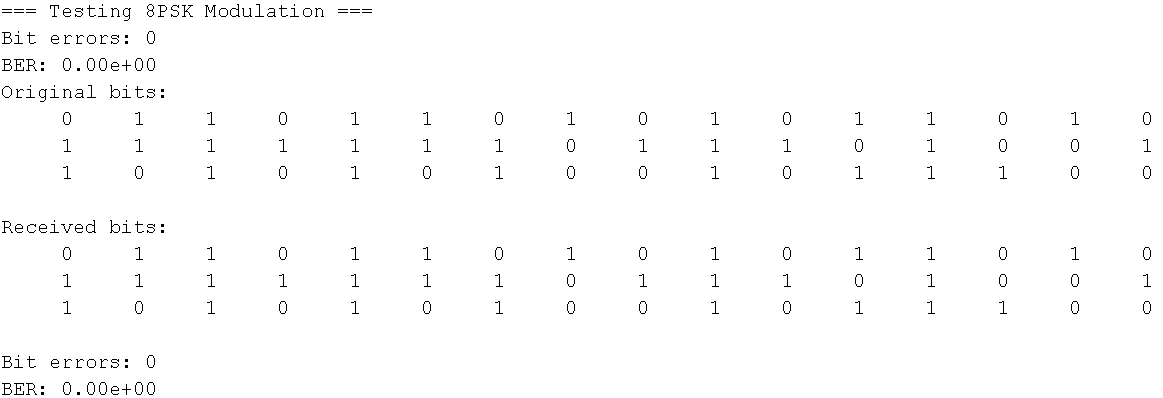
Results:

Figure 5 8PSK Test

Figure 6 QPSK Test

Figure 7 BPSK Test

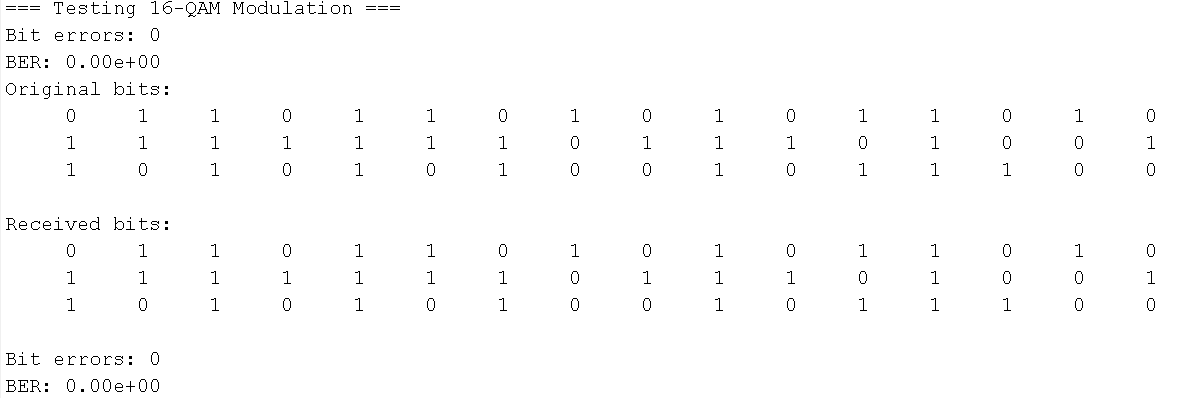


Figure 8 16-QAM Test

As shown in the figures 5, 6, 7 and 8, The noise free has zero error which means that the Tx and Rx are working properly.

Part 2 **AWGN channel**:

Now we’re going to add noise equivalent to the noise in real channel by using Average Energy Bit (Eb)

Code:

function noisy\_signals = addAWGNChannel(SNR\_range\_db, clean\_signal, Eb)

% ADDAGWNCHANNEL General AWGN channel noise adder

% Inputs:

% SNR\_range\_db - Array of SNR values in dB

% clean\_signal - Input signal (vector or matrix)

% Eb - Energy per bit

% Output:

% noisy\_signals - Cell array of noisy signals for each SNR

% Initialize output cell array

noisy\_signals = cell(length(SNR\_range\_db), 1);

% Get size of input signal

signal\_size = size(clean\_signal);

% Process each SNR point

for i = 1:length(SNR\_range\_db)

% Convert SNR from dB to linear scale

SNR\_linear = 10^(SNR\_range\_db(i)/10);

% Calculate noise power (N0)

N0 = 1 / SNR\_linear;

% Generate proper noise

if isreal(clean\_signal)

% Real noise for real signals

noise = sqrt(Eb\*N0/2) \* randn(signal\_size);

else

% Complex noise for complex signals

noise = sqrt(Eb\*N0/2) \* (randn(signal\_size) + 1j\*randn(signal\_size));

end

% Add noise to the signal

noisy\_signals{i} = clean\_signal + noise;

end

% If only one SNR point was requested, return array instead of cell

if length(SNR\_range\_db) == 1

noisy\_signals = noisy\_signals{1};

end

end

So the output is scattered on the constellation graph

Output: