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:

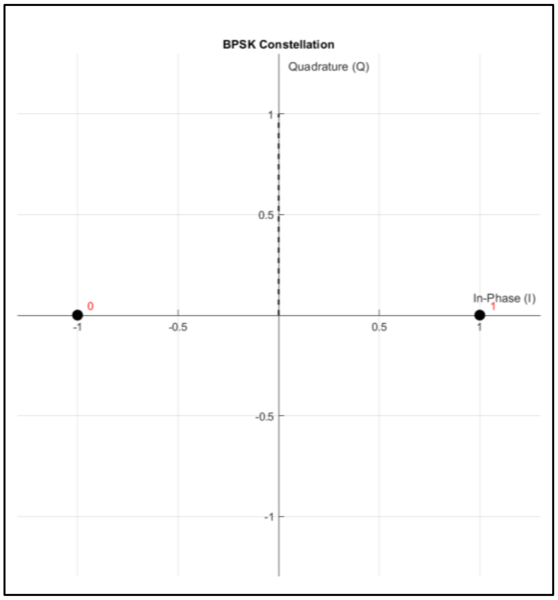

Figure 1 BPSK constellation

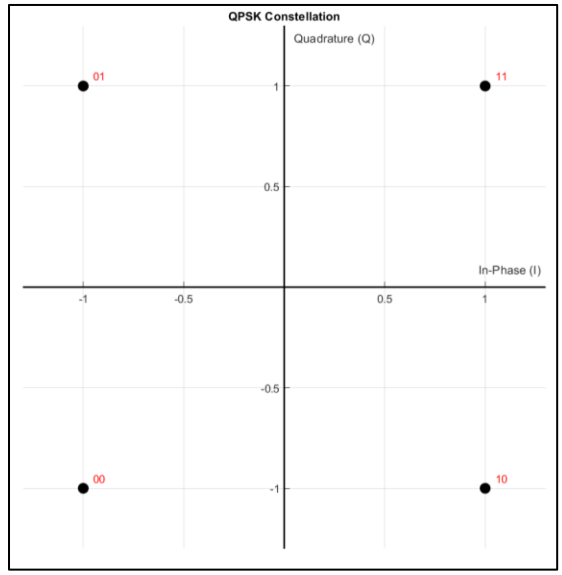

Figure 2 QPSK constellation

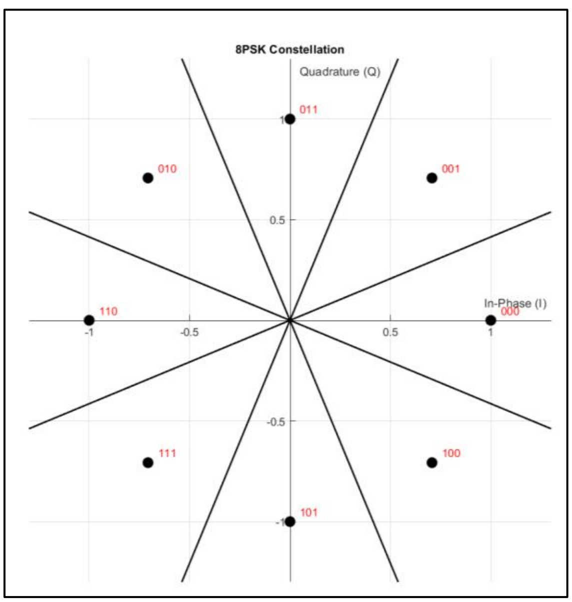

Figure 4 8PSK constellation

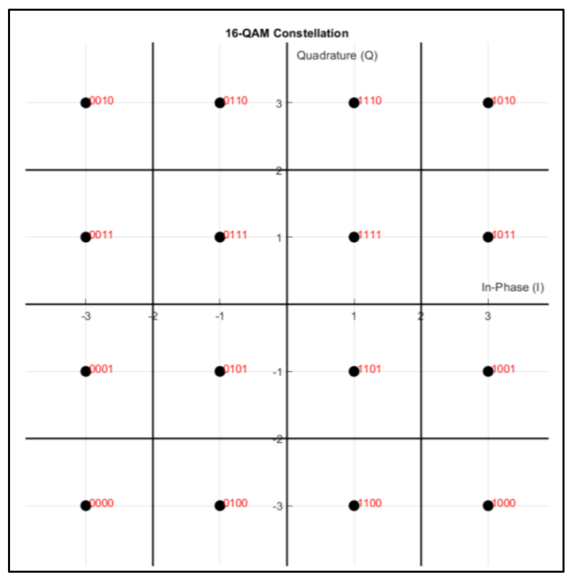

Figure 3 16-QAM constellation

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

$X_{BB} = X_I + j\,X_Q$

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:

```
=== Testing BPSK Modulation ===
```

```
=== Testing 16-QAM Modulation ===
Bit errors: 0
BER: 0.00e+00
Original bits:
     0     1     1     0     1     1     0     1     0     1     0     1     1     0     1     0
     1     1     1     1     1     1     1     0     1     1     1     0     1     0     0     1
     1     0     1     0     1     0     1     0     0     1     0     1     1     1     0     0

Received bits:
     0     1     1     0     1     1     0     1     0     1     0     1     1     0     1     0
     1     1     1     1     1     1     1     0     1     1     1     0     1     0     0     1
     1     0     1     0     1     0     1     0     0     1     0     1     1     1     0     0

Bit errors: 0
BER: 0.00e+00
```

Figure 7 BPSK Test

Figure 8 16-QAM Test

```
=== Testing QPSK Modulation ===
Bit errors: 0
BER: 0.00e+00
Original bits:
     0     1     1     0     1     1     0     1     0     1     0     1     1     0     1     0
     1     1     1     1     1     1     1     0     1     1     1     0     1     0     0     1
     1     0     1     0     1     0     1     0     0     1     0     1     1     1     0     0

Received bits:
     0     1     1     0     1     1     0     1     0     1     0     1     1     0     1     0
     1     1     1     1     1     1     1     0     1     1     1     0     1     0     0     1
     1     0     1     0     1     0     1     0     0     1     0     1     1     1     0     0

Bit errors: 0
BER: 0.00e+00
```

Figure 6 QPSK Test

```
=== Testing 8PSK Modulation ===
Bit errors: 0
BER: 0.00e+00
Original bits:
     0     1     1     0     1     1     0     1     0     1     0     1     1     0     1     0
     1     1     1     1     1     1     1     0     1     1     1     0     1     0     0     1
     1     0     1     0     1     0     1     0     0     1     0     1     1     1     0     0

Received bits:
     0     1     1     0     1     1     0     1     0     1     0     1     1     0     1     0
     1     1     1     1     1     1     1     0     1     1     1     0     1     0     0     1
     1     0     1     0     1     0     1     0     0     1     0     1     1     1     0     0

Bit errors: 0
BER: 0.00e+00
```

Figure 5 8PSK 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:

BPSK:

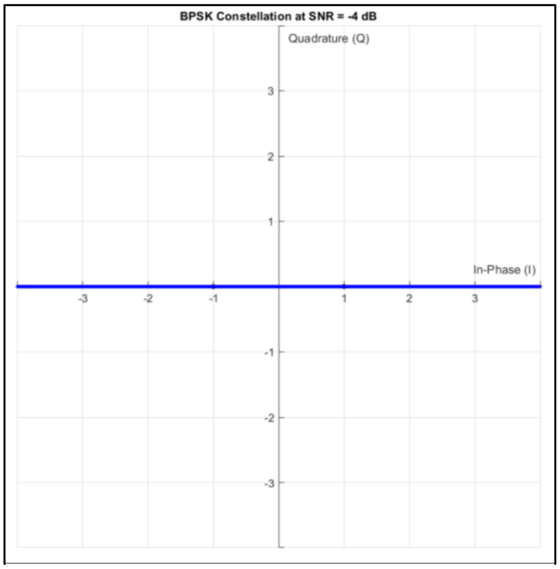

Figure 9 Noise on BPSK with SNR = -4 dB

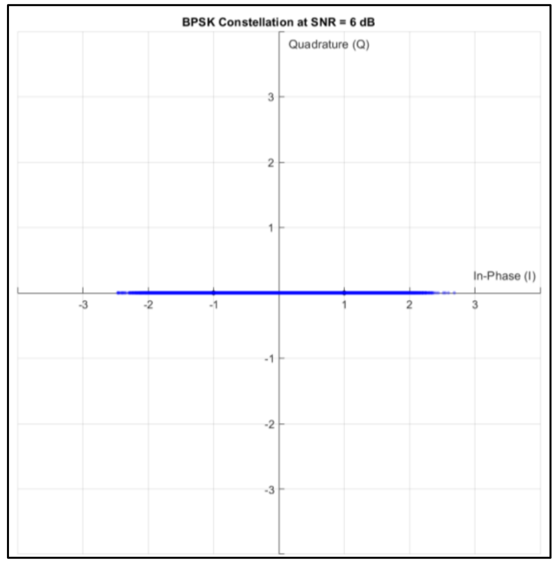

Figure 10 Noise on BPSK with SNR = 6 dB

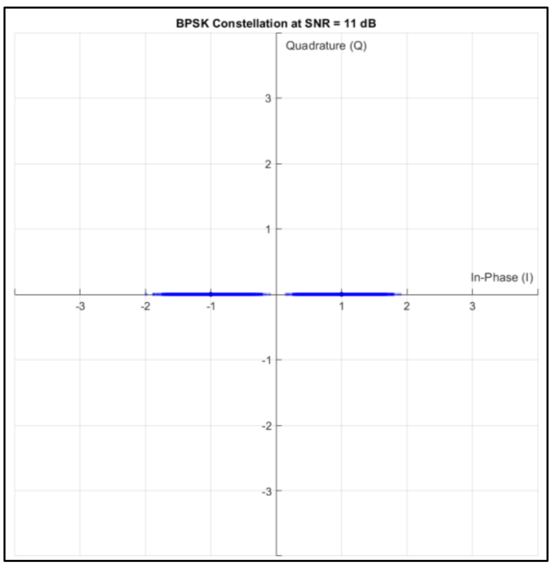

Figure 12 Noise on BPSK with SNR = 11 dB

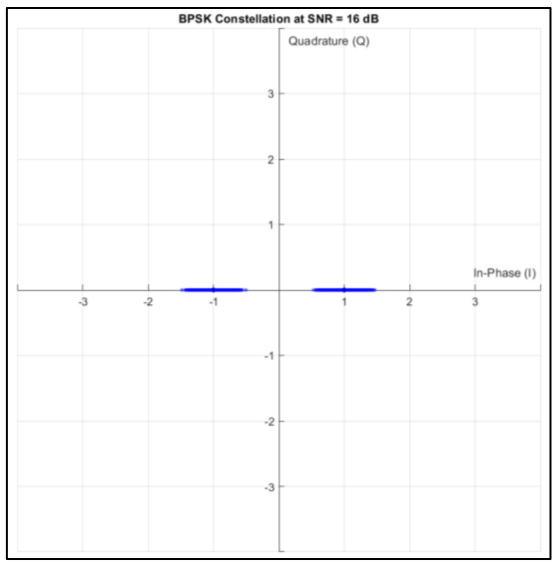

Figure 11 Noise on BPSK with SNR = 16 dB

QPSK:

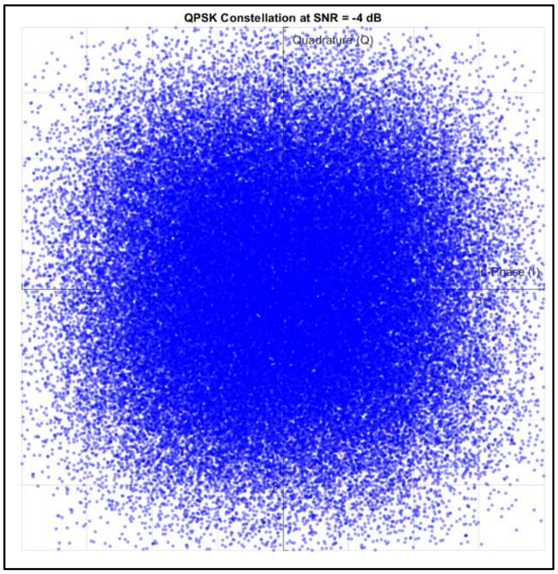

Figure 13 Noise on QPSK with SNR = -4 dB

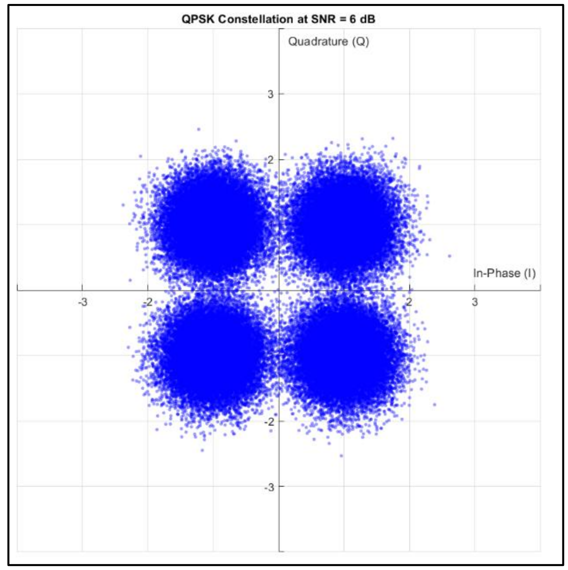

Figure 14 Noise on QPSK with SNR = 6 dB

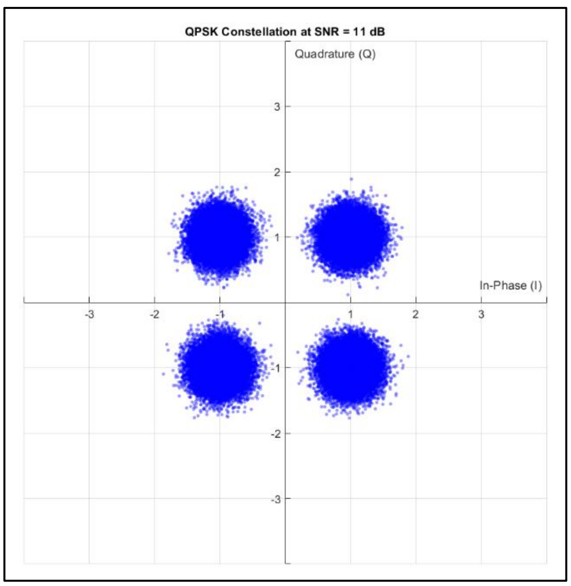

Figure 16 Noise on QPSK with SNR = 11 dB

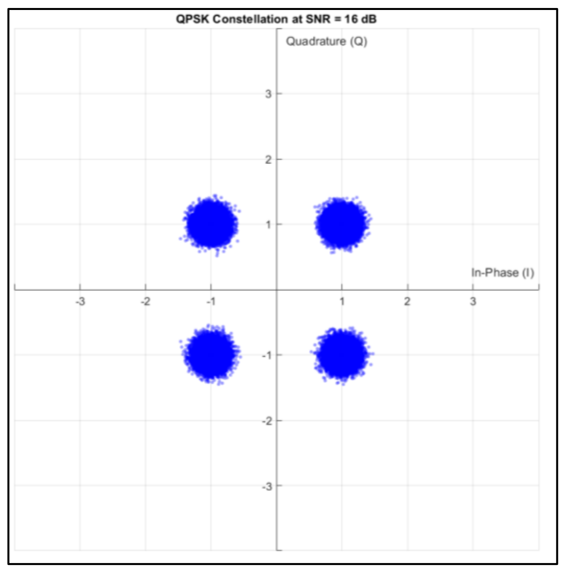

Figure 15 Noise on QPSK with SNR = 16 dB

8PSK:

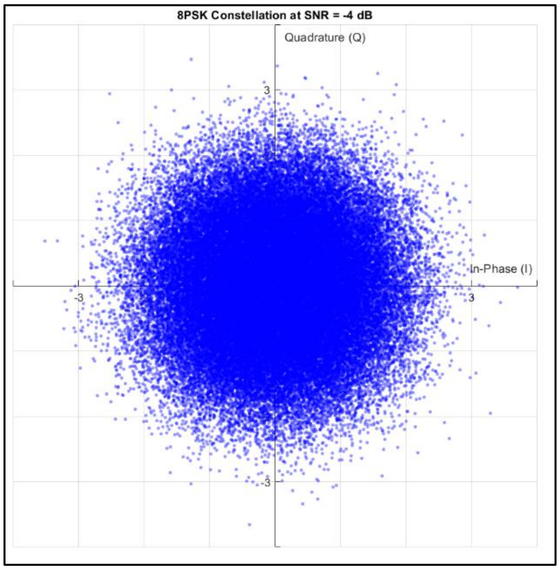

Figure 17 Noise on 8PSK with SNR = -4 dB

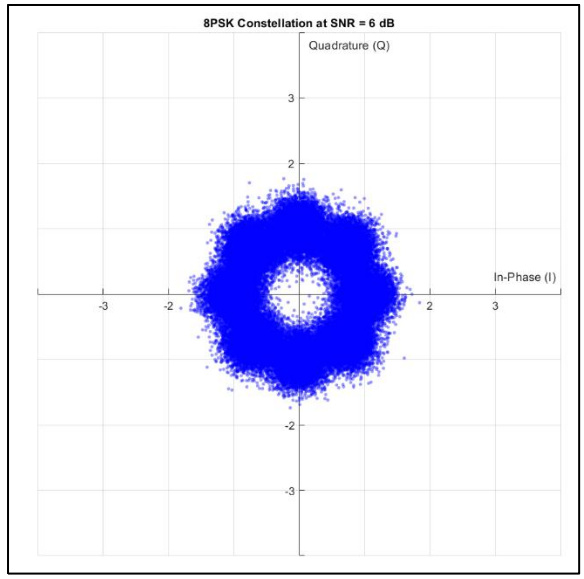

Figure 18 Noise on 8PSK with SNR = 6 dB

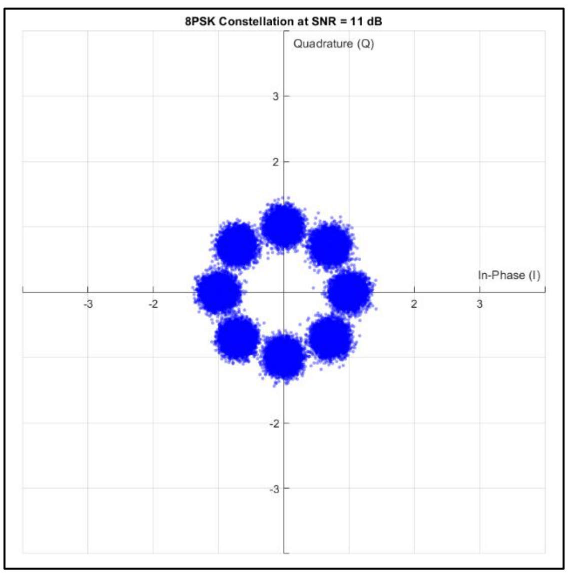

Figure 19 Noise on 8PSK with SNR = 11 dB

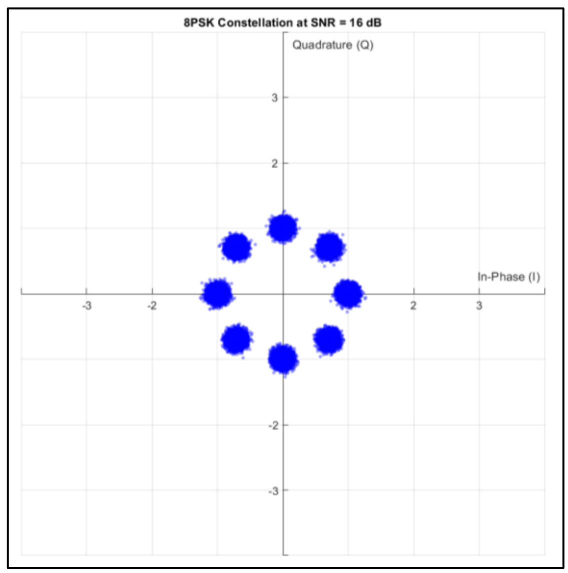

Figure 20 Noise on 8PSK with SNR = 16 dB

16QAM:

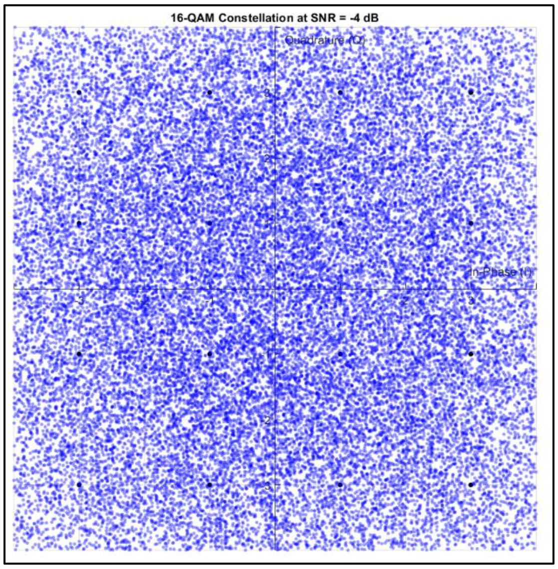

Figure 22 Noise on 16QAM with SNR = -4 dB

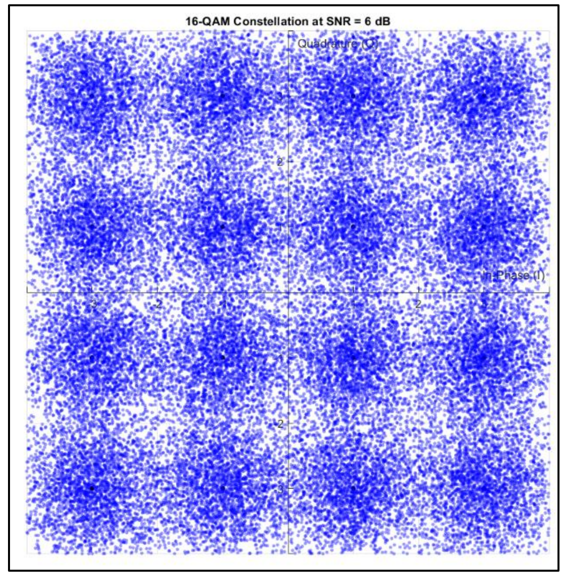

Figure 21 Noise on 16QAM with SNR = 1 dB

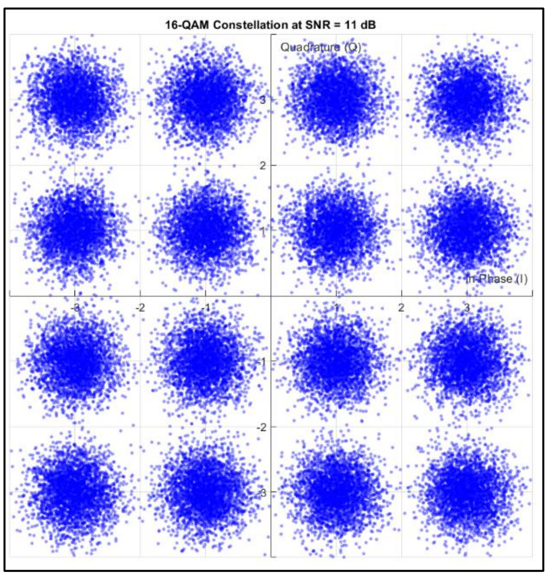

Figure 24 Noise on 16QAM with SNR = 11 dB

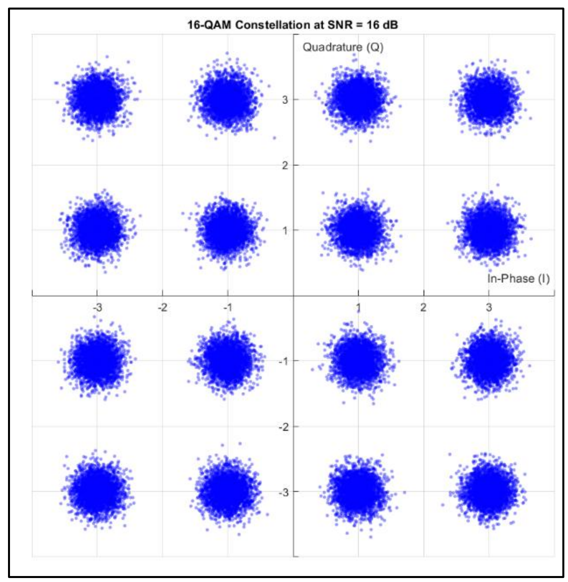

Figure 23 Noise on 16QAM with SNR = 16 dB

It is obvious that noise affects the location of sent symbols on constellation and from the plots we can estimate how good the BER for each scheme based on how good the symbols are well separated where BPSK<QPSK<8PSK16QAM<BFSK.

Tasks

Task 1

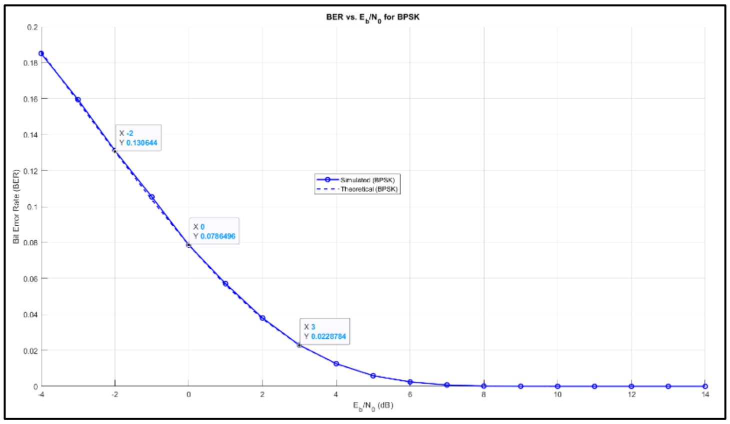

Figure 28 Simulated vs Theoretical BER for BPSK

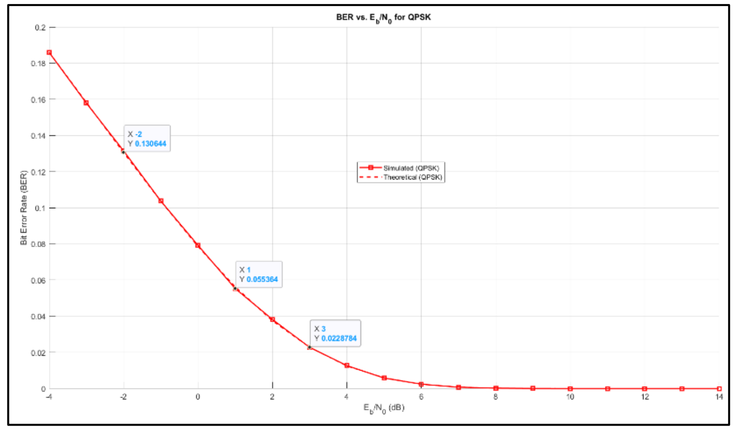

Figure 27 Simulated vs Theoretical BER for QPSK

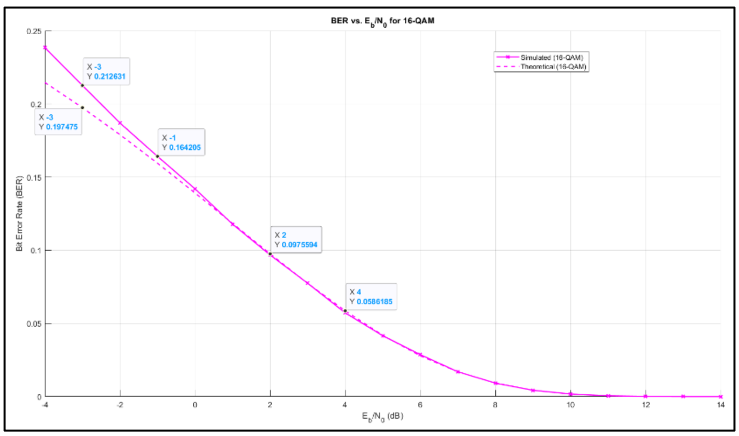

Figure 26 Simulated vs Theoretical BER for 16QAM

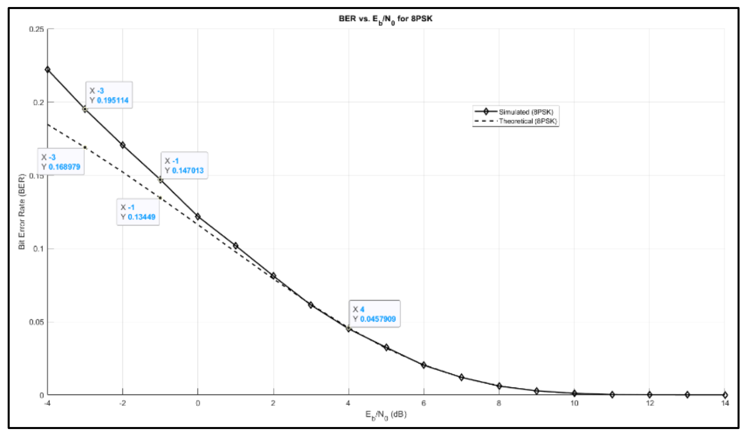

Figure 25 Simulated vs Theoretical BER for 8PSK

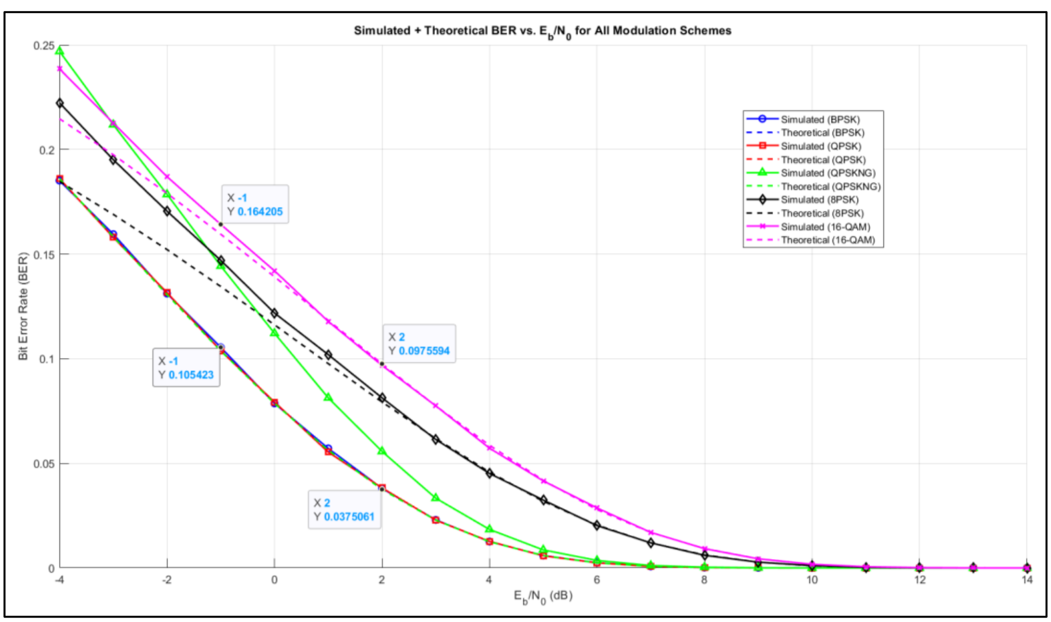

Figure 29 Simulated and Theoritical BER for BPSK, QPSK, 8PSK and 16QAM

Task 2

QPSK not Grey

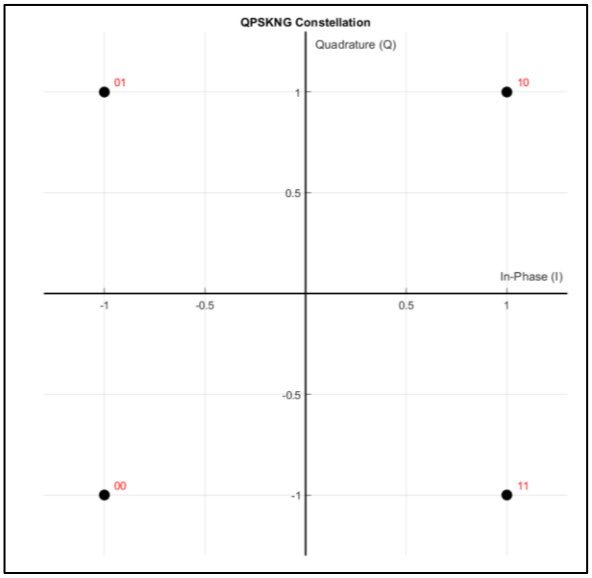

Figure 30 QPSKNG constellation

Output:

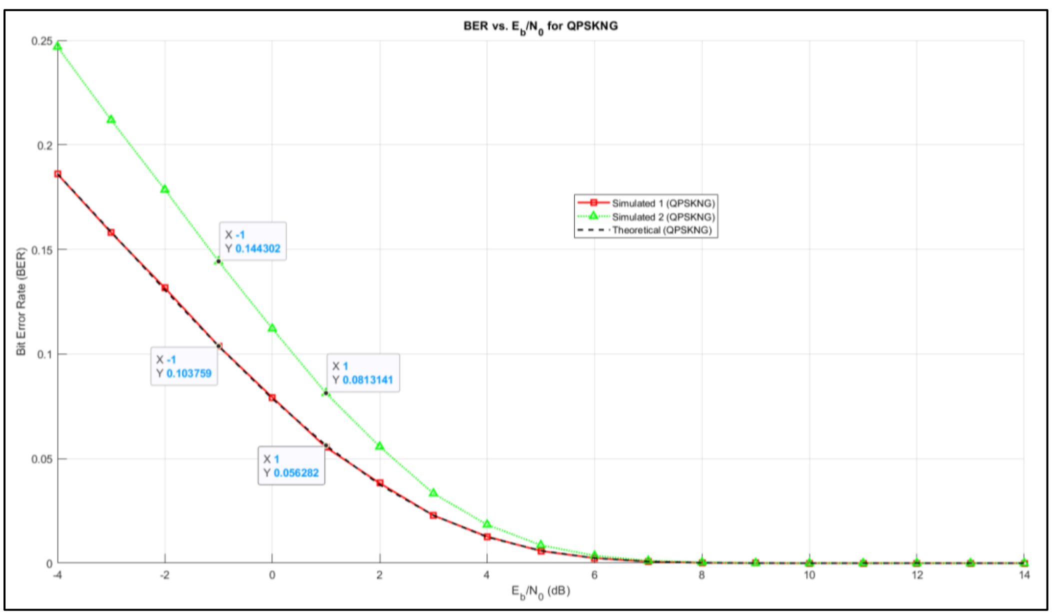

Figure 31 QPSK vs QPSKNG BER