Análise de Desempenho de Modulações Digitais

Sistemas de Comunicação I

Arthur Cadore Matuella Barcella

29 de Julho de 2024

Sumário

1. Introdução:

O objetivo deste relatório é analisar o desempenho das modulações digitais MPSK e MQAM em um sistema de comunicação digital. Para isso, foram simuladas diferentes modulações e analisados os resultados obtidos.

2. Desenvolvimento e Resultados:

O desenvolvimento deste relatório foi dividido em três diferentes seções, onde foi analisado o desempenho individual de cada modulação e em seguida um comparativo entre as duas diferentes modulações.

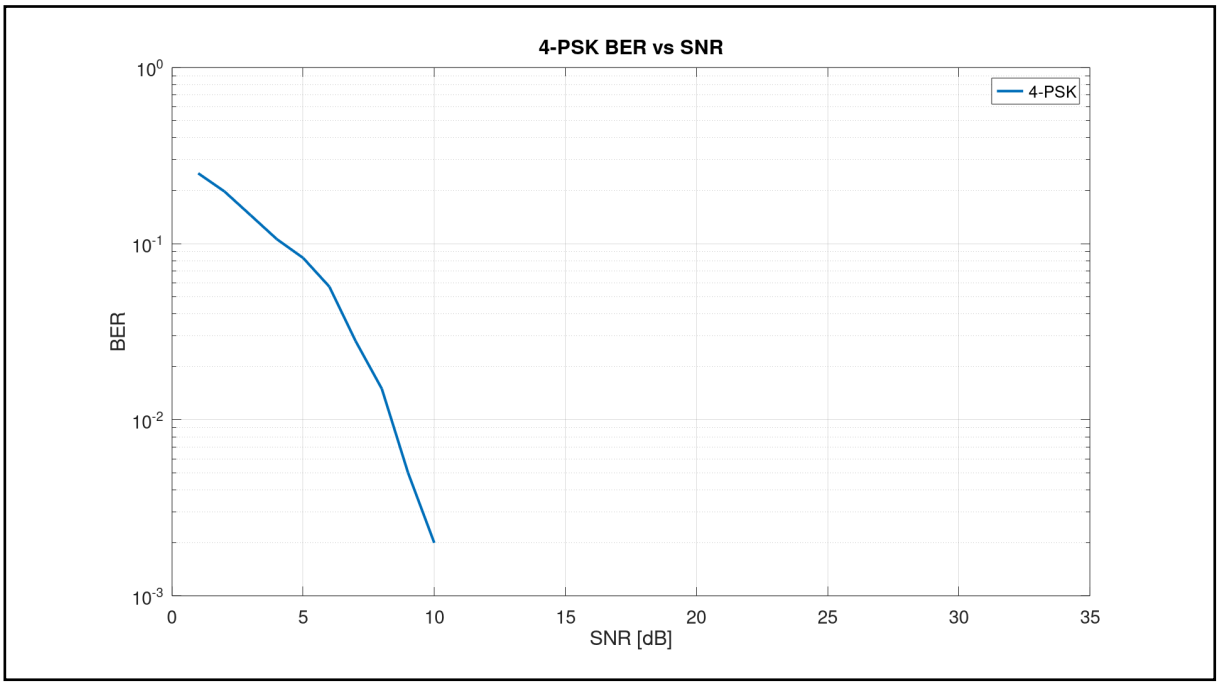
2.1. Desempenho de modulações MPSK

```
clear all; close all; clc
pkg load communications;

% Função para realizar a modulação PSK
function symbols = psk_modulate(data, M)
    phase_step = 2 * pi / M;
    phases = (0:M-1) * phase_step;
    symbols = exp(1i * phases(data + 1));
end

% Função para realizar a demodulação PSK
function demodulated_data = psk_demodulate(symbols, M)
    phase_step = 2 * pi / M;
    phases = (0:M-1) * phase_step;
    [~, demodulated_data] = min(abs(symbols(:) - exp(1i * phases)), [], 2);
    demodulated_data = demodulated_data - 1;
end

% Define a quantidade de dados a serem gerados
data_length = 1000;
data = randi([0 1], data_length, 1); % Gera dados aleatórios binários

for M = [4, 8, 16, 32]
    % Modulação PSK
    psk_symbols = psk_modulate(data, M);

    psk_ber = [];

    for snr = 1:34
        snr_linear = 10^(snr / 10);
        noise_variance = 1 / (2 * snr_linear);
           noise = sqrt(noise_variance) * (randn(size(psk_symbols)) + 1i * randn(size(psk_symbols)));

        psk_noisy = psk_symbols + noise;

        % Demodulação PSK
        psk_demodulated = psk_demodulate(psk_noisy, M);

        % Calcula o BER
```

```
        errors = sum(psk_demodulated ~= data);
        psk_ber = [psk_ber, errors / data_length];
    end

    % Plotando o gráfico para cada valor de M
    figure;
    semilogy(1:34, psk_ber, 'DisplayName', sprintf('%d-PSK', M));
    xlabel('SNR [dB]');
    ylabel('BER');
    title(sprintf('%d-PSK BER vs SNR', M));
    legend('show');
    grid on;
    set(gca, 'FontSize', 14);
end
```

2.1.1. 4-MPSK

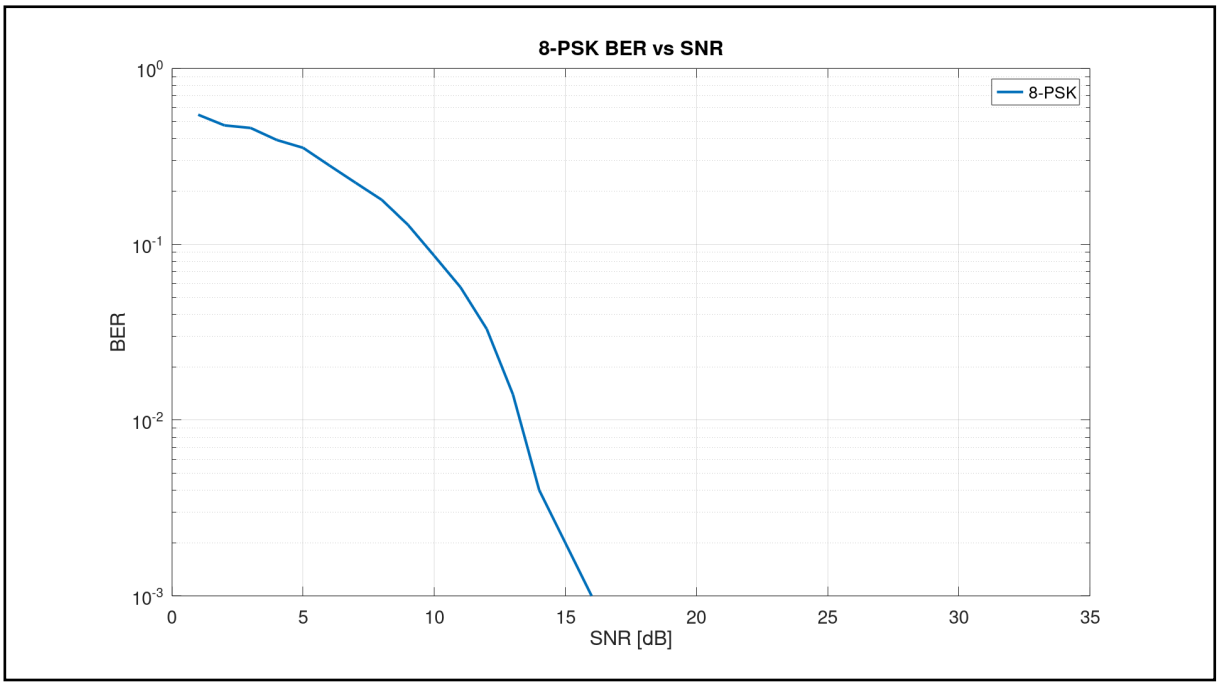
Figure 1: Elaborada pelo Autor



Taxa de erro de bit (BER) para modulação 4-PSK

2.1.2. 8-MPSK

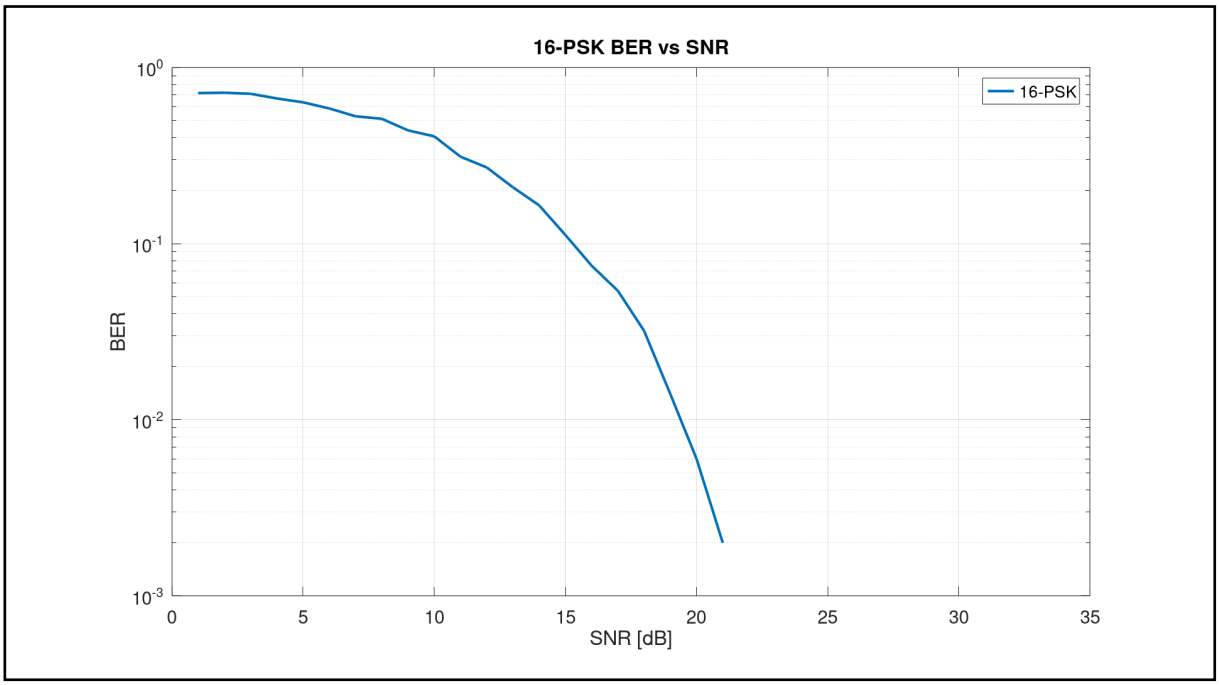
Figure 2: Elaborada pelo Autor



Taxa de erro de bit (BER) para modulação 8-PSK

2.1.3. 16-MPSK

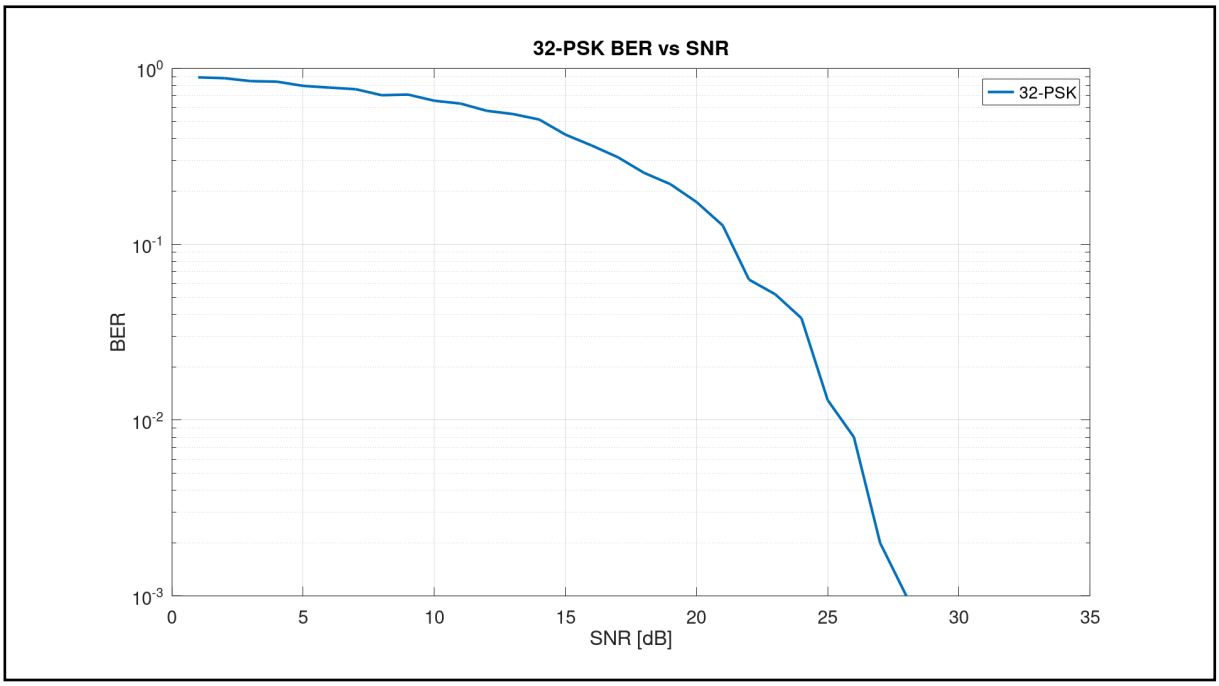
Figure 3: Elaborada pelo Autor



Taxa de erro de bit (BER) para modulação 16-PSK

2.1.4. 32-MPSK

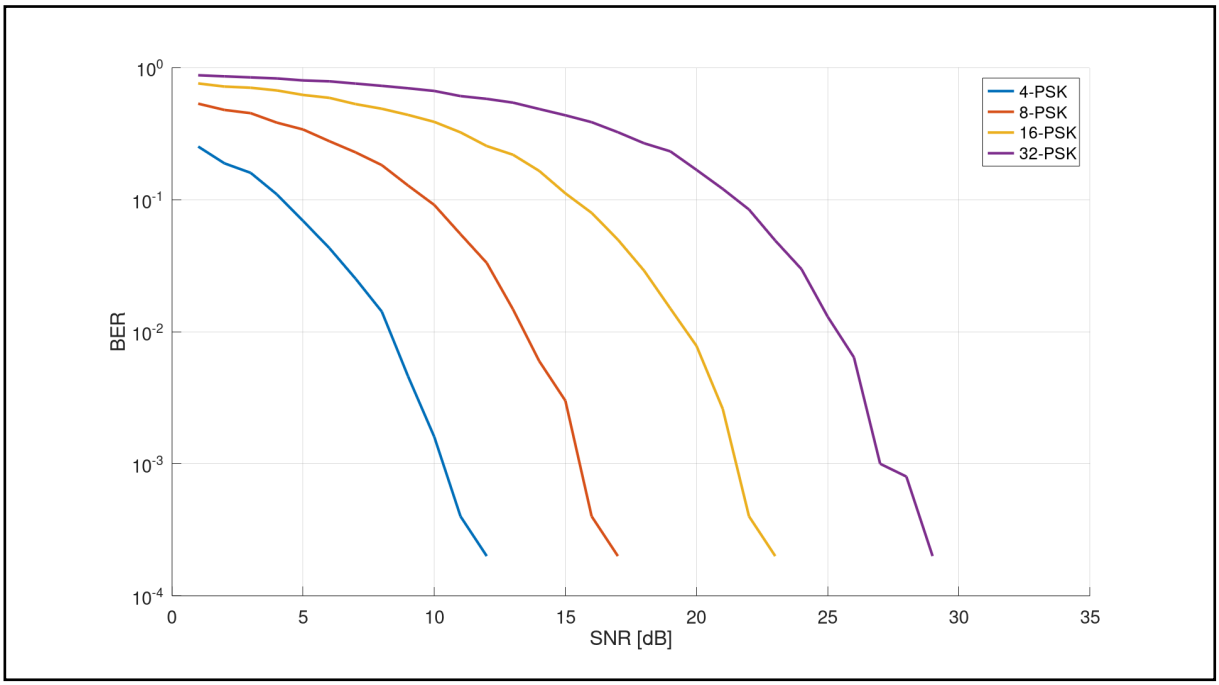
Figure 4: Elaborada pelo Autor



Taxa de erro de bit (BER) para modulação 32-PSK

2.1.5. Desempenho comparativo:

Figure 5: Elaborada pelo Autor



Taxa de erro de bit (BER) comparativa

```
Clear all; close all; clc;
pkg load communications;

% Função para realizar a modulação PSK
function symbols = psk_modulate(data, M)
```

```
    phase_step = 2 * pi / M;
    phases = (0:M-1) * phase_step;
    symbols = exp(1i * phases(data + 1));
end

% Função para realizar a demodulação PSK
function demodulated_data = psk_demodulate(symbols, M)
    phase_step = 2 * pi / M;
    phases = (0:M-1) * phase_step;
    [~, demodulated_data] = min(abs(symbols(:) - exp(1i * phases)), [], 2);
    demodulated_data = demodulated_data - 1;
end

figure;
hold on;
grid on;
set(gca, 'FontSize', 14);

for M = [4, 8, 16, 32]
    % Define a quantidade de dados a serem gerados
    data_length = 1000;
    data = randi([0 M-1], data_length, 1); % Gera dados aleatórios

    % Modulação PSK
    psk_symbols = psk_modulate(data, M);

    psk_ber = [];

    for snr = 1:34
        snr_linear = 10^(snr / 10);
        noise_variance = 1 / (2 * snr_linear);
           noise = sqrt(noise_variance) * (randn(size(psk_symbols)) + 1i
* randn(size(psk_symbols)));

        psk_noisy = psk_symbols + noise;

        % Demodulação PSK
        psk_demodulated = psk_demodulate(psk_noisy, M);

        % Calcula o BER
        errors = sum(psk_demodulated ~= data);
        psk_ber = [psk_ber, errors / data_length];
    end

    semilogy(1:34, psk_ber, 'DisplayName', sprintf('%d-PSK', M));
end

xlabel('SNR [dB]');
ylabel('BER');
legend('show');
hold off;
```

2.2. Desempenho de modulações MQAM

```
pkg load communications; % Carrega o pacote de comunicações

% Função para realizar a modulação QAM
```

```
function symbols = qam_modulate(data, M)
    % Define o número de bits por símbolo
    k = log2(M);
    % Define o tamanho da grade QAM
    n = sqrt(M);

    % Normaliza os dados
    data = mod(data, M);

    % Converte dados para uma matriz de símbolos
    symbols = zeros(size(data));
    for i = 1:numel(data)
        % Mapeia os dados para uma posição na matriz QAM
        x = mod(data(i), n) - (n-1)/2;
        y = floor(data(i) / n) - (n-1)/2;
        symbols(i) = x + 1i * y;
    end
end

% Função para realizar a demodulação QAM
function demodulated_data = qam_demodulate(symbols, M)
    % Define o número de bits por símbolo
    k = log2(M);
    % Define o tamanho da grade QAM
    n = sqrt(M);

    % Inicializa o vetor de dados demodulados
    demodulated_data = zeros(size(symbols));

    % Demodula cada símbolo
    for i = 1:numel(symbols)
        % Extrai a parte real e imaginária do símbolo
        x = real(symbols(i));
        y = imag(symbols(i));
        % Mapeia para a posição da matriz QAM
        x_idx = round(x + (n-1)/2);
        y_idx = round(y + (n-1)/2);
        % Converte para o índice de dados
        demodulated_data(i) = x_idx + n * y_idx;
    end
end

% Define a quantidade de dados a serem gerados
data_length = 1000;
data = randi([0 63], data_length, 1); % Gera dados aleatórios

% Loop para diferentes tamanhos de QAM
for M = [4, 16, 64]
    % Modulação QAM
    qam_symbols = qam_modulate(data, M);

    qam_ber = [];

    for snr = 1:14
        snr_linear = 10^(snr / 10);
        noise_variance = 1 / (2 * snr_linear);
```

```
            noise = sqrt(noise_variance) * (randn(size(qam_symbols)) + 1i
* randn(size(qam_symbols)));

        qam_noisy = qam_symbols + noise;

        % Demodulação QAM
        qam_demodulated = qam_demodulate(qam_noisy, M);

        % Calcula o BER
        errors = sum(qam_demodulated ~= data);
        qam_ber = [qam_ber, errors / data_length];
    end

    % Plotando o gráfico para cada valor de M
    figure;
    semilogy(1:14, qam_ber, 'DisplayName', sprintf('%d-QAM', M));
    xlabel('SNR [dB]');
    ylabel('BER');
    title(sprintf('%d-QAM BER vs SNR', M));
    legend('show');
    grid on;
    set(gca, 'FontSize', 14);
end
```

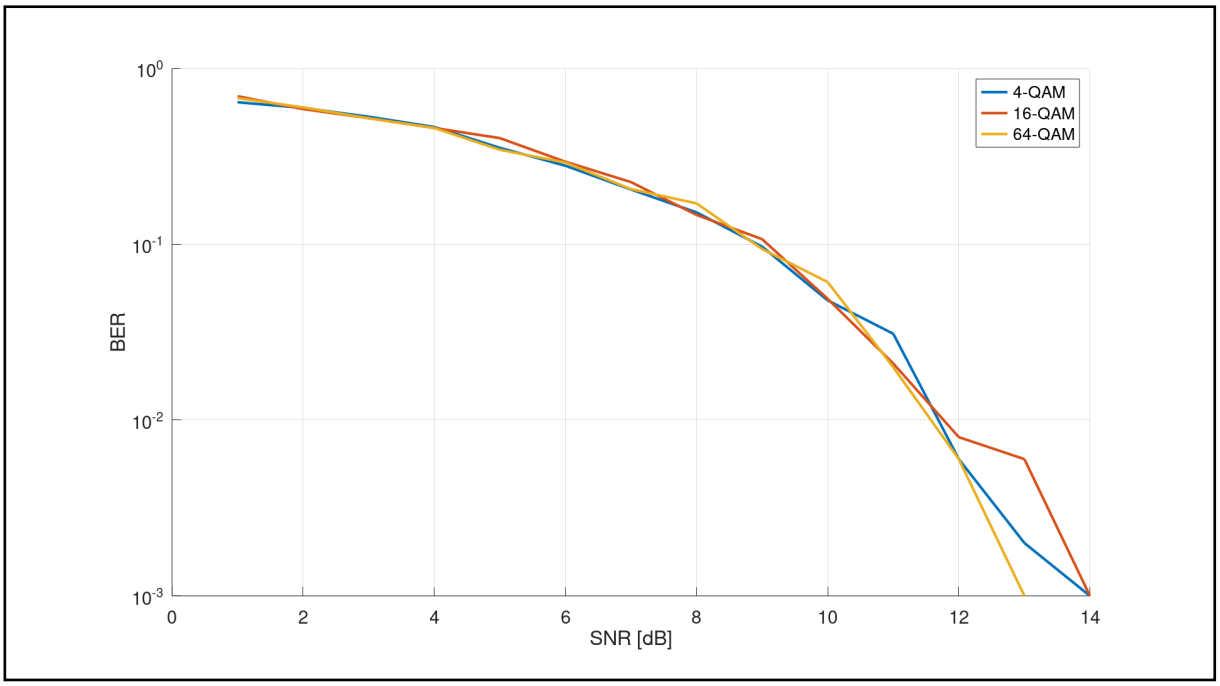
2.2.1. 4-MQAM

2.2.2. 16-MQAM

2.2.3. 64-MQAM

2.2.4. Desempenho comparativo:

Figure 6: Elaborada pelo Autor



Taxa de erro de bit (BER) comparativa

```
clear all; close all; clc;
pkg load communications;

% Função para realizar a modulação QAM
function symbols = qam_modulate(data, M)
    % Define o número de bits por símbolo
    k = log2(M);
    % Define o tamanho da grade QAM
    n = sqrt(M);

    % Normaliza os dados
    data = mod(data, M);

    % Converte dados para uma matriz de símbolos
    symbols = zeros(size(data));
    for i = 1:numel(data)
        % Mapeia os dados para uma posição na matriz QAM
        x = mod(data(i), n) - (n-1)/2;
        y = floor(data(i) / n) - (n-1)/2;
        symbols(i) = x + 1i * y;
    end
end

% Função para realizar a demodulação QAM
function demodulated_data = qam_demodulate(symbols, M)
    % Define o número de bits por símbolo
    k = log2(M);
    % Define o tamanho da grade QAM
    n = sqrt(M);

    % Inicializa o vetor de dados demodulados
    demodulated_data = zeros(size(symbols));

    % Demodula cada símbolo
    for i = 1:numel(symbols)
        % Extrai a parte real e imaginária do símbolo
        x = real(symbols(i));
        y = imag(symbols(i));
        % Mapeia para a posição da matriz QAM
        x_idx = round(x + (n-1)/2);
        y_idx = round(y + (n-1)/2);
        % Converte para o índice de dados
        demodulated_data(i) = x_idx + n * y_idx;
    end
end

figure;
hold on;
grid on;
set(gca, 'FontSize', 14);

for M = [4, 16, 64]
    % Define a quantidade de dados a serem gerados
    data_length = 1000; % Número reduzido de amostras
    data = randi([0 M-1], data_length, 1); % Gera dados aleatórios

    % Modulação QAM
    qam_symbols = qam_modulate(data, M);
```

```
59
60      qam_ber = [];
61
62      for snr = 1:14
63          snr_linear = 10^(snr / 10);
64          noise_variance = 1 / (2 * snr_linear);
65             noise = sqrt(noise_variance) * (randn(size(qam_symbols)) + 1i
    * randn(size(qam_symbols)));
66
67          qam_noisy = qam_symbols + noise;
68
69          % Demodulação QAM
70          qam_demodulated = qam_demodulate(qam_noisy, M);
71
72          % Calcula o BER
73          errors = sum(qam_demodulated ~= data);
74          qam_ber = [qam_ber, errors / data_length];
75      end
76
77      semilogy(1:14, qam_ber, 'DisplayName', sprintf('%d-QAM', M));
78  end
79
80  xlabel('SNR [dB]');
81  ylabel('BER');
82  legend('show');
83  hold off;
```

2.3. Comparação de modulações PSK vs. QAM

2.3.1. 4-PSK vs. 4-QAM

2.3.2. 16-PSK vs. 16-QAM

2.3.3. 64-PSK vs. 64-QAM

3. Conclusão:

4. Referências Bibliográficas:

Para o desenvolvimento deste relatório, foi utilizado o seguinte material de referência:

- Software Defined Radio Using MATLAB & Simulink and the RTL-SDR, de Robert W. Stewart