



**INSTITUTO
FEDERAL**

Santa Catarina

Câmpus
São José

Análise de Desempenho de Modulações Digitais

Sistemas de Comunicação I

Arthur Cadore Matuella Barcella

29 de Julho de 2024

Sumário

1. Introdução:	3
2. Desenvolvimento e Resultados:	3
2.1. Desempenho de modulações MPSK	3
2.1.1. 4-MPSK	4
2.1.2. 8-MPSK	4
2.1.3. 16-MPSK	5
2.1.4. 32-MPSK	5
2.1.5. Desempenho comparativo:	6
2.2. Desempenho de modulações MQAM	7
2.2.1. 4-MQAM	9
2.2.2. 16-MQAM	9
2.2.3. 64-MQAM	9
2.2.4. Desempenho comparativo:	9
2.3. Comparação de modulações PSK vs. QAM	11
2.3.1. 4-PSK vs. 4-QAM	11
2.3.2. 16-PSK vs. 16-QAM	11
2.3.3. 64-PSK vs. 64-QAM	11
3. Conclusão:	11
4. Referências Bibliográficas:	11

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

```
1 clear all; close all; clc
2 pkg load communications;
3
4 % Função para realizar a modulação PSK
5 function symbols = psk_modulate(data, M)
6     phase_step = 2 * pi / M;
7     phases = (0:M-1) * phase_step;
8     symbols = exp(1i * phases(data + 1));
9 end
10
11 % Função para realizar a demodulação PSK
12 function demodulated_data = psk_demodulate(symbols, M)
13     phase_step = 2 * pi / M;
14     phases = (0:M-1) * phase_step;
15     [~, demodulated_data] = min(abs(symbols(:) - exp(1i * phases)), [], 2);
16     demodulated_data = demodulated_data - 1;
17 end
18
19 % Define a quantidade de dados a serem gerados
20 data_length = 1000;
21 data = randi([0 1], data_length, 1); % Gera dados aleatórios binários
22
23 for M = [4, 8, 16, 32]
24     % Modulação PSK
25     psk_symbols = psk_modulate(data, M);
26
27     psk_ber = [];
28
29     for snr = 1:34
30         snr_linear = 10^(snr / 10);
31         noise_variance = 1 / (2 * snr_linear);
32         noise = sqrt(noise_variance) * (randn(size(psk_symbols)) + 1i
33 * randn(size(psk_symbols)));
34
35         psk_noisy = psk_symbols + noise;
36
37         % Demodulação PSK
38         psk_demodulated = psk_demodulate(psk_noisy, M);
39
40         % Calcula o BER
```

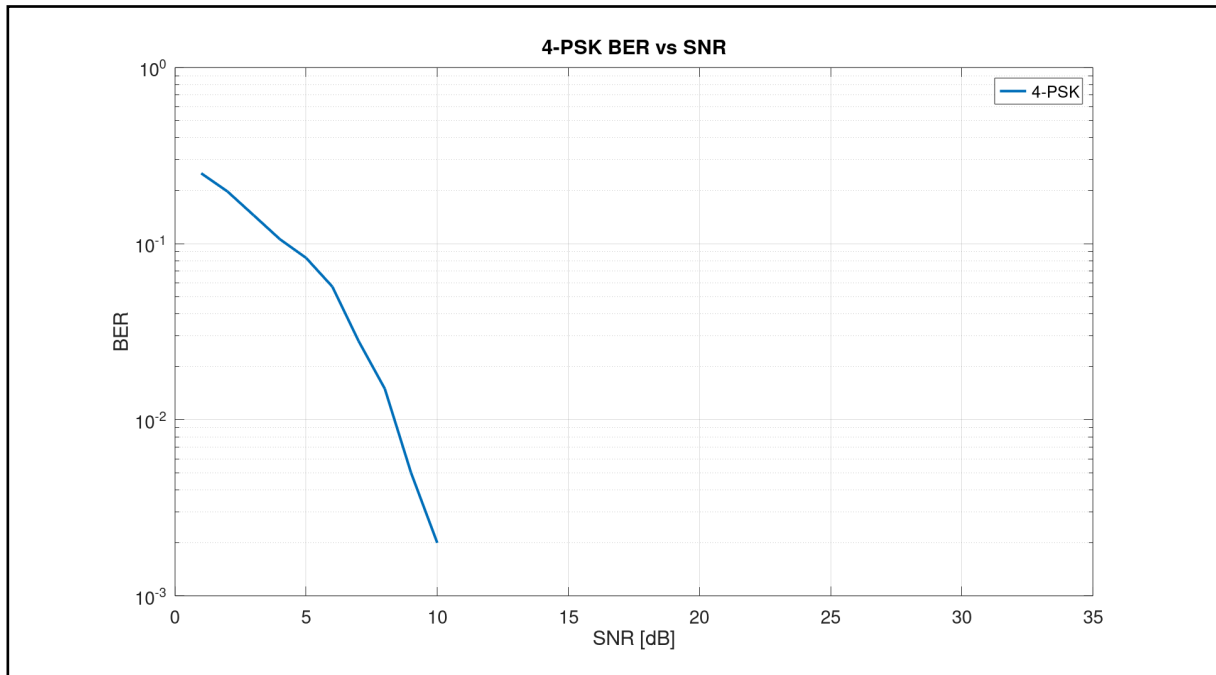
```

40     errors = sum(psk_demodulated ~= data);
41     psk_ber = [psk_ber, errors / data_length];
42 end
43
44 % Plotando o gráfico para cada valor de M
45 figure;
46 semilogy(1:34, psk_ber, 'DisplayName', sprintf('%d-PSK', M));
47 xlabel('SNR [dB]');
48 ylabel('BER');
49 title(sprintf('%d-PSK BER vs SNR', M));
50 legend('show');
51 grid on;
52 set(gca, 'FontSize', 14);
53 end

```

2.1.1. 4-MPSK

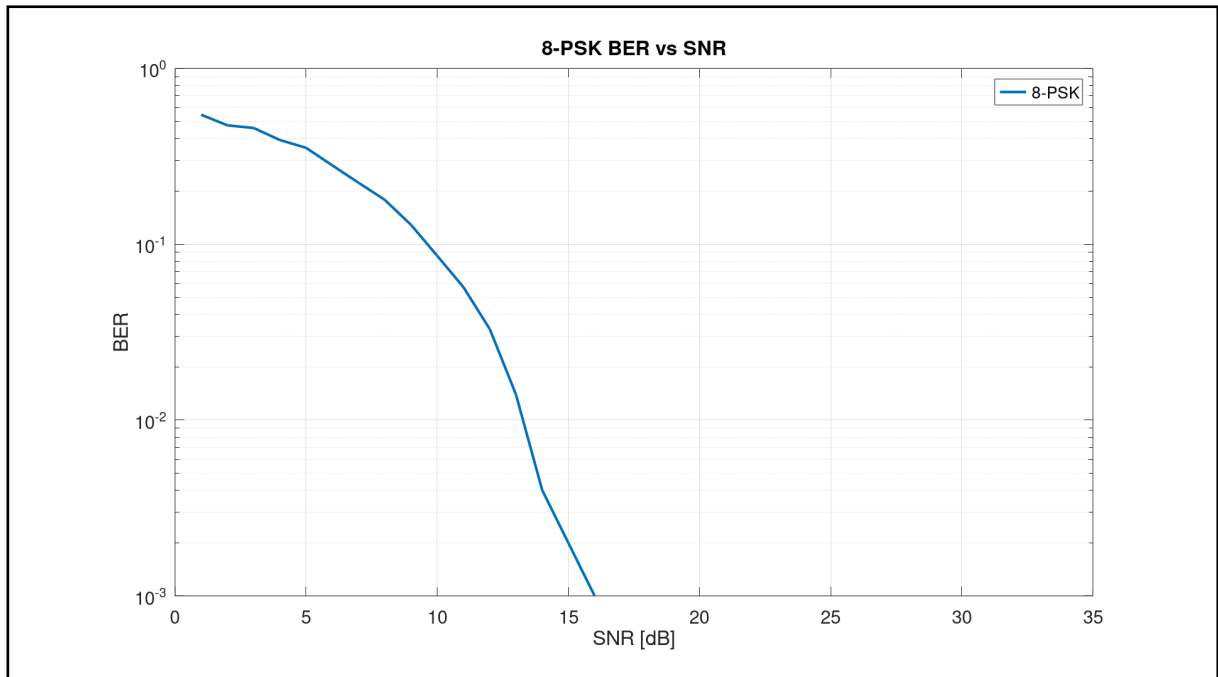
Figure 1: Elaborada pelo Autor



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

2.1.2. 8-MPSK

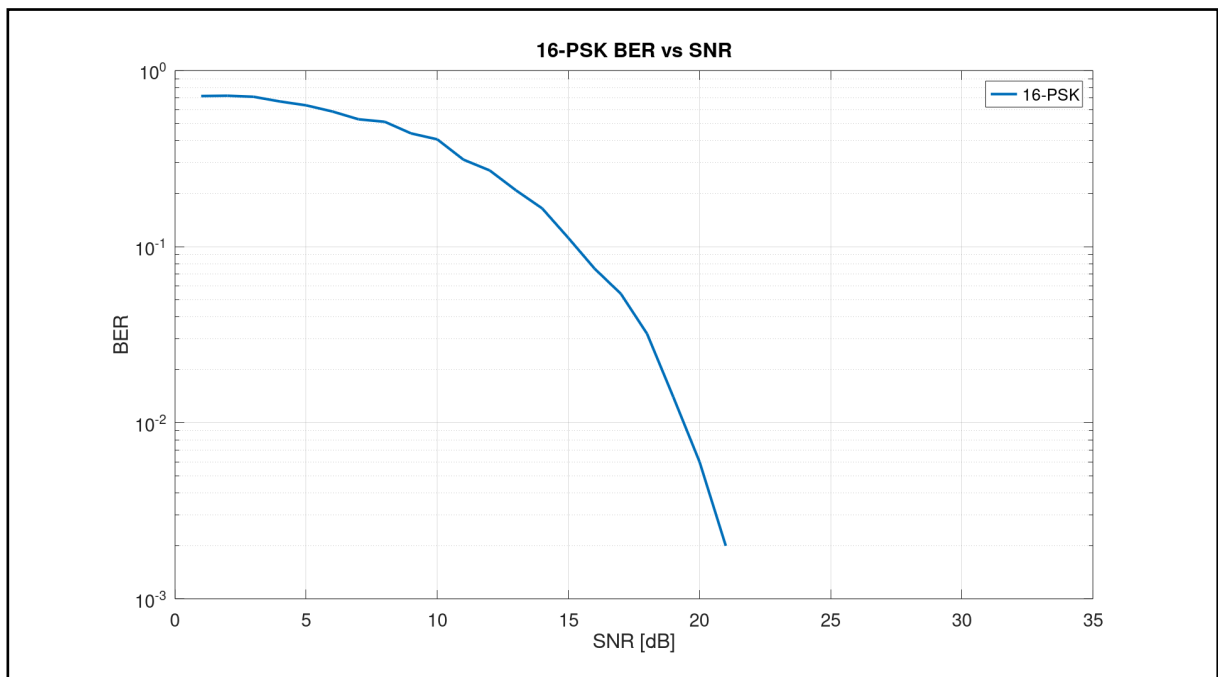
Figure 2: Elaborada pelo Autor



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

2.1.3. 16-MPSK

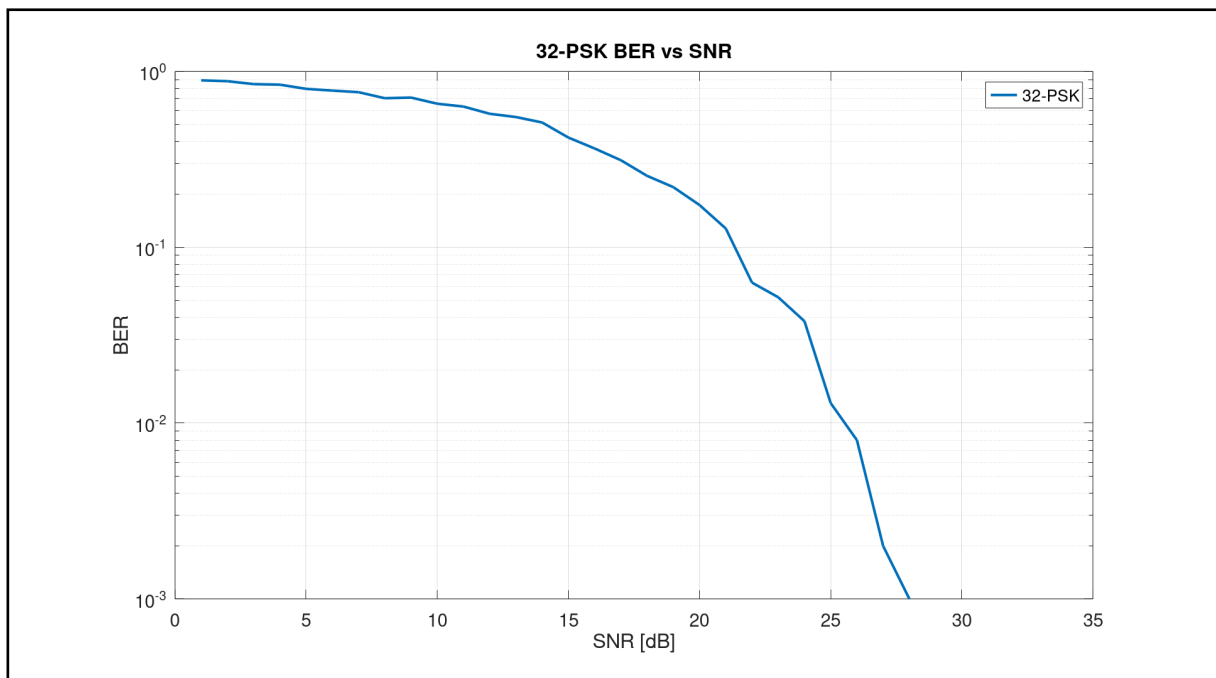
Figure 3: Elaborada pelo Autor



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

2.1.4. 32-MPSK

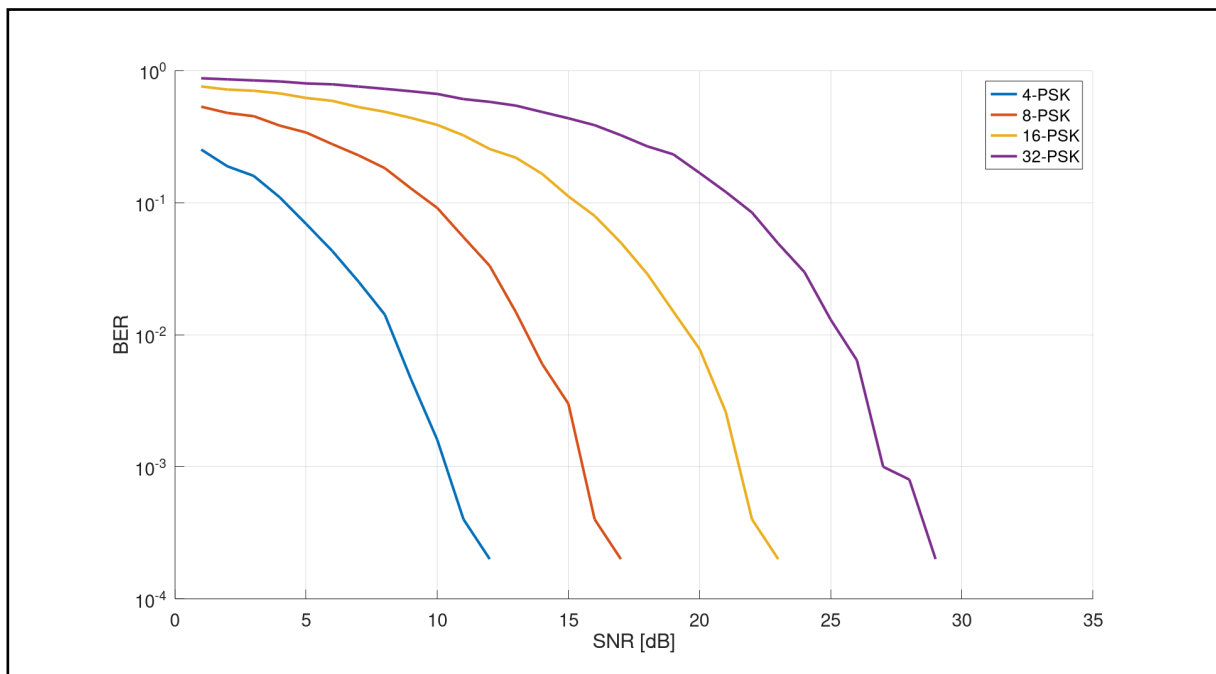
Figure 4: Elaborada pelo Autor



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

```

1 Clear all; close all; clc;
2 pkg load communications;
3
4 % Função para realizar a modulação PSK
5 function symbols = psk_modulate(data, M)

```

```

6     phase_step = 2 * pi / M;
7     phases = (0:M-1) * phase_step;
8     symbols = exp(1i * phases(data + 1));
9 end
10
11 % Função para realizar a demodulação PSK
12 function demodulated_data = psk_demodulate(symbols, M)
13     phase_step = 2 * pi / M;
14     phases = (0:M-1) * phase_step;
15     [~, demodulated_data] = min(abs(symbols(:) - exp(1i * phases)), [], 2);
16     demodulated_data = demodulated_data - 1;
17 end
18
19 figure;
20 hold on;
21 grid on;
22 set(gca, 'FontSize', 14);
23
24 for M = [4, 8, 16, 32]
25     % Define a quantidade de dados a serem gerados
26     data_length = 1000;
27     data = randi([0 M-1], data_length, 1); % Gera dados aleatórios
28
29     % Modulação PSK
30     psk_symbols = psk_modulate(data, M);
31
32     psk_ber = [];
33
34     for snr = 1:34
35         snr_linear = 10^(snr / 10);
36         noise_variance = 1 / (2 * snr_linear);
37         noise = sqrt(noise_variance) * (randn(size(psk_symbols)) + 1i
38 * randn(size(psk_symbols)));
39
40         psk_noisy = psk_symbols + noise;
41
42         % Demodulação PSK
43         psk_demodulated = psk_demodulate(psk_noisy, M);
44
45         % Calcula o BER
46         errors = sum(psk_demodulated ~= data);
47         psk_ber = [psk_ber, errors / data_length];
48     end
49     semilogy(1:34, psk_ber, 'DisplayName', sprintf('%d-PSK', M));
50 end
51
52 xlabel('SNR [dB]');
53 ylabel('BER');
54 legend('show');
55 hold off;

```

2.2. Desempenho de modulações MQAM

```

1 pkg load communications; % Carrega o pacote de comunicações
2
3 % Função para realizar a modulação QAM

```

```

4 function symbols = qam_modulate(data, M)
5     % Define o número de bits por símbolo
6     k = log2(M);
7     % Define o tamanho da grade QAM
8     n = sqrt(M);
9
10    % Normaliza os dados
11    data = mod(data, M);
12
13    % Converte dados para uma matriz de símbolos
14    symbols = zeros(size(data));
15    for i = 1:numel(data)
16        % Mapeia os dados para uma posição na matriz QAM
17        x = mod(data(i), n) - (n-1)/2;
18        y = floor(data(i) / n) - (n-1)/2;
19        symbols(i) = x + 1i * y;
20    end
21 end
22
23 % Função para realizar a demodulação QAM
24 function demodulated_data = qam_demodulate(symbols, M)
25     % Define o número de bits por símbolo
26     k = log2(M);
27     % Define o tamanho da grade QAM
28     n = sqrt(M);
29
30     % Inicializa o vetor de dados demodulados
31     demodulated_data = zeros(size(symbols));
32
33     % Demodula cada símbolo
34     for i = 1:numel(symbols)
35         % Extrai a parte real e imaginária do símbolo
36         x = real(symbols(i));
37         y = imag(symbols(i));
38         % Mapeia para a posição da matriz QAM
39         x_idx = round(x + (n-1)/2);
40         y_idx = round(y + (n-1)/2);
41         % Converte para o índice de dados
42         demodulated_data(i) = x_idx + n * y_idx;
43     end
44 end
45
46 % Define a quantidade de dados a serem gerados
47 data_length = 1000;
48 data = randi([0 63], data_length, 1); % Gera dados aleatórios
49
50 % Loop para diferentes tamanhos de QAM
51 for M = [4, 16, 64]
52     % Modulação QAM
53     qam_symbols = qam_modulate(data, M);
54
55     qam_ber = [];
56
57     for snr = 1:14
58         snr_linear = 10^(snr / 10);
59         noise_variance = 1 / (2 * snr_linear);

```



```

60         noise = sqrt(noise_variance) * (randn(size(qam_symbols)) + 1i
* randn(size(qam_symbols)));
61
62         qam_noisy = qam_symbols + noise;
63
64         % Demodulação QAM
65         qam_demodulated = qam_demodulate(qam_noisy, M);
66
67         % Calcula o BER
68         errors = sum(qam_demodulated ~= data);
69         qam_ber = [qam_ber, errors / data_length];
70     end
71
72     % Plotando o gráfico para cada valor de M
73     figure;
74     semilogy(1:14, qam_ber, 'DisplayName', sprintf('%d-QAM', M));
75     xlabel('SNR [dB]');
76     ylabel('BER');
77     title(sprintf('%d-QAM BER vs SNR', M));
78     legend('show');
79     grid on;
80     set(gca, 'FontSize', 14);
81 end

```

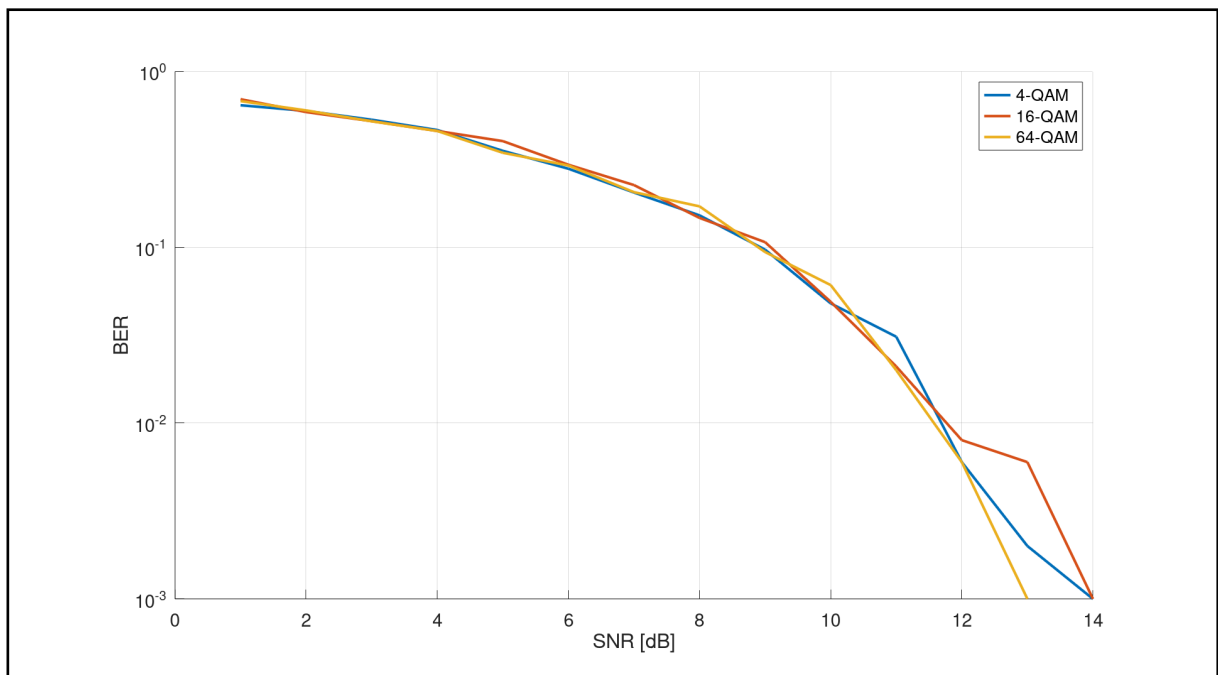
2.2.1. 4-MQAM

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

```

1  clear all; close all; clc;
2  pkg load communications;
3
4  % Função para realizar a modulação QAM
5  function symbols = qam_modulate(data, M)
6      % Define o número de bits por símbolo
7      k = log2(M);
8      % Define o tamanho da grade QAM
9      n = sqrt(M);
10
11     % Normaliza os dados
12     data = mod(data, M);
13
14     % Converte dados para uma matriz de símbolos
15     symbols = zeros(size(data));
16     for i = 1:numel(data)
17         % Mapeia os dados para uma posição na matriz QAM
18         x = mod(data(i), n) - (n-1)/2;
19         y = floor(data(i) / n) - (n-1)/2;
20         symbols(i) = x + 1i * y;
21     end
22 end
23
24 % Função para realizar a demodulação QAM
25 function demodulated_data = qam_demodulate(symbols, M)
26     % Define o número de bits por símbolo
27     k = log2(M);
28     % Define o tamanho da grade QAM
29     n = sqrt(M);
30
31     % Inicializa o vetor de dados demodulados
32     demodulated_data = zeros(size(symbols));
33
34     % Demodula cada símbolo
35     for i = 1:numel(symbols)
36         % Extrai a parte real e imaginária do símbolo
37         x = real(symbols(i));
38         y = imag(symbols(i));
39         % Mapeia para a posição da matriz QAM
40         x_idx = round(x + (n-1)/2);
41         y_idx = round(y + (n-1)/2);
42         % Converte para o índice de dados
43         demodulated_data(i) = x_idx + n * y_idx;
44     end
45 end
46
47 figure;
48 hold on;
49 grid on;
50 set(gca, 'FontSize', 14);
51
52 for M = [4, 16, 64]
53     % Define a quantidade de dados a serem gerados
54     data_length = 1000; % Número reduzido de amostras
55     data = randi([0 M-1], data_length, 1); % Gera dados aleatórios
56
57     % Modulação QAM
58     qam_symbols = qam_modulate(data, M);

```

```

59     qam_ber = [];
60
61     for snr = 1:14
62         snr_linear = 10^(snr / 10);
63         noise_variance = 1 / (2 * snr_linear);
64         noise = sqrt(noise_variance) * (randn(size(qam_symbols)) + 1i
65 * 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