

UNIVERSIDADE FEDERAL DE SANTA MARIA

**SIMULAÇÃO DE UM SISTEMA DE
COMUNICAÇÃO DIGITAL**

TELECOMUNICAÇÕES ELC1120

Pedro Augusto Bandeira

Santa Maria, RS, Brasil

2024

1 DESENVOLVIMENTO

Este trabalho visa a simulação de um canal de transmissão de comunicação digital e segue o diagrama da Figura 1.1.

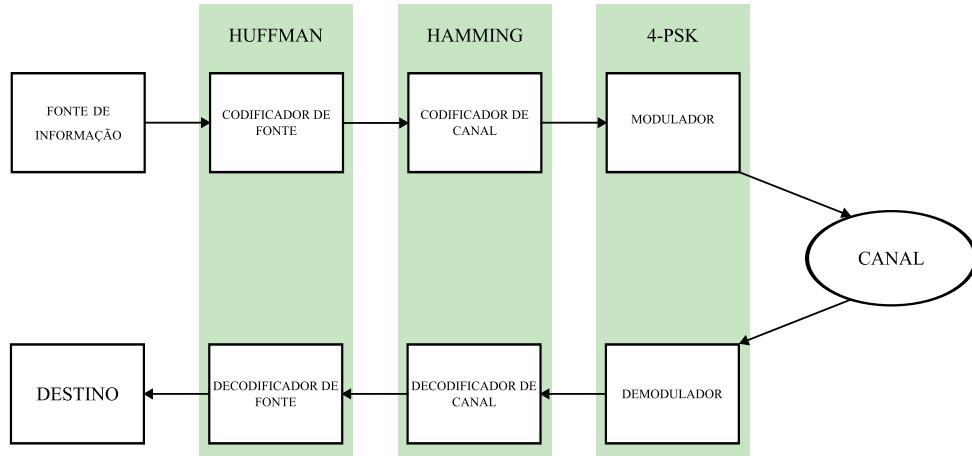


Figura 1.1: Diagrama do processo

1.1 Fonte de informação

A fonte de informação é o livro Alice no País das Maravilhas, disponibilizado pelo professor.

1.2 Codificador de fonte

Foi utilizado o código de *Huffman* desenvolvido no trabalho anterior.

1.3 Codificador de canal

Para o codificador de canal, utilizou-se o código de *Hamming* com a matriz de paridade abaixo.

$$H = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

O código utilizado é um (13,8) com 8 bits de dados e 5 de paridade. A distância mínima para esta matriz é 3, o que confere a ela uma capacidade de detecção de 2 bits e a capacidade de correção de 1 bit.

A implementação desta etapa é simples: para gerar as palavras-código, basta multiplicar a informação original por uma matriz G . A matriz G é formada pela concatenação de uma matriz identidade, com o mesmo número de colunas que a matriz H , e pela transposição da submatriz P , que, neste caso, corresponde às 8 primeiras colunas de H .

$$H = [P \mid I] \rightarrow G = [I \mid P^T]$$

Desta forma, a sequência binária produzida pelo codificador de fonte é separada em blocos de 8 bits, que são então multiplicados pela matriz G .

1.4 Modulação

A modulação utilizada foi a 4-PSK, sua constelação pode ser observada na Figura 1.2.

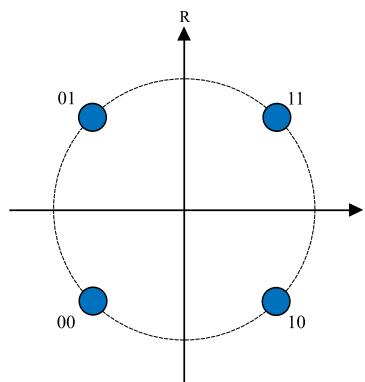


Figura 1.2: Constelação 4-PSK

Modular o sinal consiste em dividir o sinal dos códigos gerados por *Hamming* em fatias de 2 bits, que são atribuídas a uma coordenada no plano complexo.

```

1 for i = 1:2:N % Gera números ímpar
2 par = palavra_codigo(i:i+1); % Faz a fatia de 2 bits. Ex: i=1 -> (1:2),
   i=3 -> (3:4)
3 if isequal(par, [1; 1])
4     cod_mod((i+1)/2) = (1 + 1i); % 11 -> (1, 1)
5 elseif isequal(par, [1; 0])
6     cod_mod((i+1)/2) = (1 - 1i); % 10 -> (1, -1)
7 elseif isequal(par, [0; 1])
8     cod_mod((i+1)/2) = (-1 + 1i); % 01 -> (-1, 1)
9 else
10    cod_mod((i+1)/2) = (-1 - 1i); % 00 -> (-1, -1)
11 end
12 end

```

Na Figura 1.3, é possível observar a constelação gerada sob diferentes níveis de ruído. À medida que o nível de ruído aumenta, os pontos da constelação se dispersam em torno de

suas posições ideais, resultando em uma maior probabilidade de erro na detecção dos símbolos. Essas figuras ilustram como o ruído afeta a integridade do sinal modulado, tornando mais difícil a distinção entre os diferentes símbolos no receptor. Essa dispersão dos pontos é uma representação visual direta da relação entre o ruído no canal e a taxa de erro de *bits* (BER).

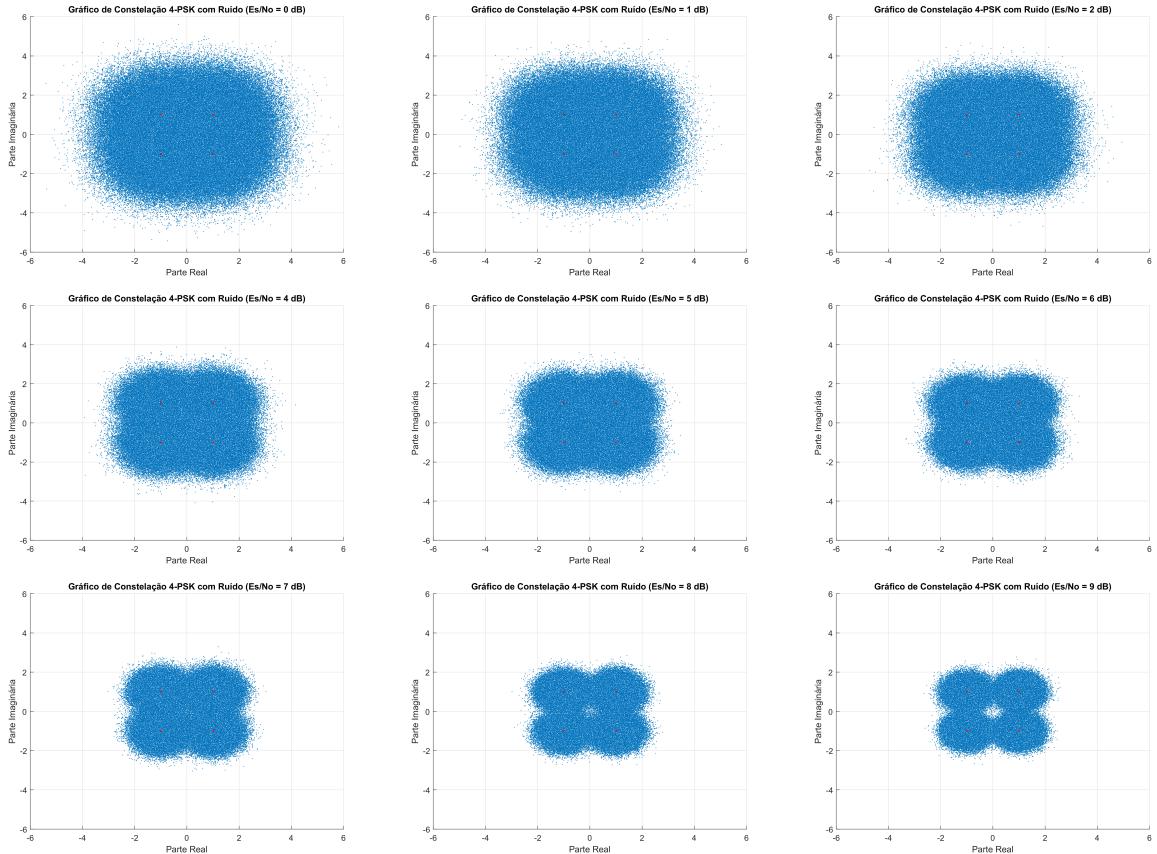


Figura 1.3: Constelação com diferentes níveis de ruído

Além dos níveis de ruído variando de 0 a 9 dB, foram simulados dois níveis adicionais até que o erro se tornasse nulo em 11 dB. Como resultado, na Figura 1.4, podemos observar uma distinção mais clara entre os pontos na constelação.

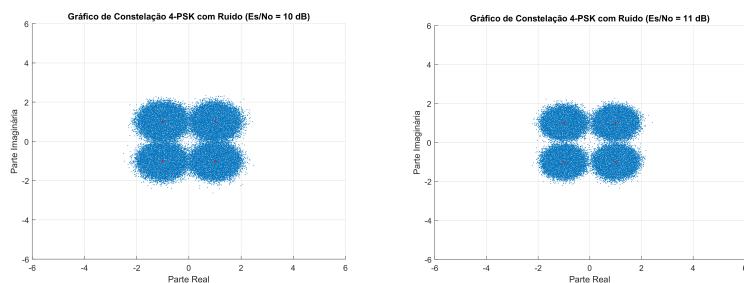


Figura 1.4: Constelações das simulações adicionais para BER = 0

1.5 Demodulação

O processo de demodulação consiste em converter os símbolos modulados recebidos de volta aos *bits* originais transmitidos. Inicialmente, os símbolos recebidos, que foram corrompidos pelo ruído do canal, são comparados com os pontos da constelação de modulação para determinar o ponto mais próximo. Essa comparação é feita calculando a distância euclidiana entre o símbolo recebido e cada ponto da constelação. O ponto da constelação com a menor distância é selecionado como o símbolo correspondente. Em seguida, esse ponto é mapeado para o par de *bits* original que ele representa, restaurando a informação original.

```

1   for i = 1:length(cod_ruido)
2       % Encontra o ponto da constelação mais próximo do símbolo recebido
3       [~, index] = min(abs(cod_ruido(i) - pontos_constel));
4       % Mapeia o índice do ponto encontrado para o par de bits
5       % correspondente
6       recebido(2*i-1:2*i) = bit_map(index, :);
7   end

```

Nesta etapa, a BER (*Bit Error Rate*) sem correção é calculada, comparando os *bits* transmitidos com os bits recebidos. A Figura 1.5 apresenta o gráfico semilog da BER em função do ruído.

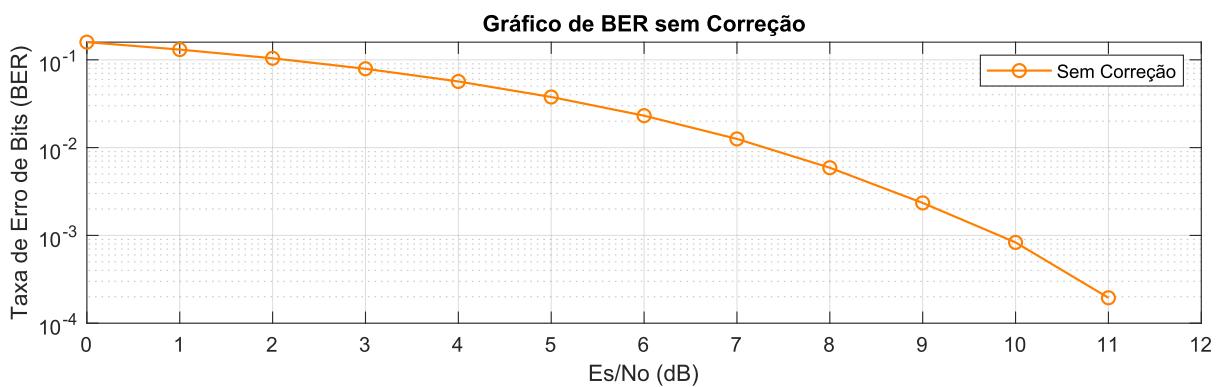


Figura 1.5: Gráfico BER sem correção X Ruído (dB)

1.6 Decodificador de canal

O decodificador de canal é responsável pela correção dos erros introduzidos durante a transmissão e pela tradução das palavras-código à sequência original de *bits*.

Inicialmente, os *bits* demodulados são reorganizados em blocos de tamanho adequado, compatíveis com o código de *Hamming* utilizado, 13 *bits*. Em seguida, é calculado o síndrome, multiplicando a palavra código transposta pela matriz H .

Se o síndrome indica que um erro ocorreu, ou seja, quando é diferente de um vetor nulo, o próximo passo é localizar o *bit* exato que foi corrompido. Isso é feito comparando o síndrome com as colunas da matriz H . Quando o *bit* corrompido é identificado, ele é corrigido invertendo o seu valor.

```

1 % Corrigir erros com base no síndrome
2 for i = 1:size(sindrome, 1)
3     if any(sindrome(i, :))
4         for j = 1:size(H, 2)
5             if isequal(sindrome(i, :), H(:, j)') %Comparacao do
6                 sindrome com as colunas de H
7                 entrada_hamming(i, j) = mod(entrada_hamming(i, j) + 1,
8                     2); %Flip de correcao
9             break;
10        end
11    end
12 end

```

Finalmente, os bits de dados são extraídos selecionando os primeiros 8 *bits* da palavra código corrigida.

A Figura 1.6 apresenta uma comparação entre a BER com correção e sem correção em função do nível de ruído (Es/No).

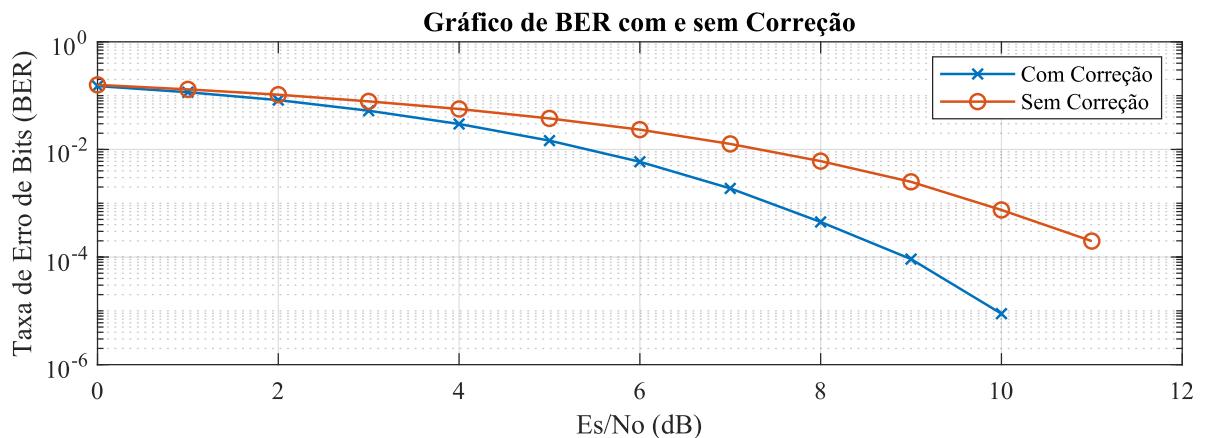


Figura 1.6: Gráfico BER com e sem correção X Ruído (dB)

Além disso, foi gerada uma Tabela 1.1 que exibe o número de erros correspondentes em cada caso, permitindo uma análise detalhada do desempenho do sistema em diferentes condições de ruído. Estes erros são relativos aos bits transmitidos e corrigidos em relação ao binário correspondente ao arquivo original codificado por Huffman, ou seja, não representam diretamente a diferença entre os textos, mas passam uma ideia da integridade do texto transmitido.

Es/No (dB)	Erros sem Correção	Erros com Correção
0	1.0751e+05	1.019e+05
1	88747	78571
2	70639	56135
3	53597	36208
4	38464	20494
5	25669	9852
6	15801	4012
7	8579	1295
8	4025	280
9	1635	48
10	495	3
11	153	0

Tabela 1.1: Comparação entre o número de erros dos bits com e sem correção para diferentes níveis de Es/No (dB).

1.7 Decodificador de fonte

Foi utilizado a função de decodificação do trabalho anterior.

```

1 function texto = decodTexto(codificar, dicionario)
2 texto = '';
3 codigo = '';
4 for i = 1:length(codificar)
5     codigo = [codigo codificar(i)];
6     id = find(strcmp({dicionario.codigo}, codigo));
7     if ~isempty(id)
8         texto = [texto dicionario(id).simbolo];
9         codigo = '';
10    end
11 end
12 end

```

2 ANEXO A

```

1 % Huffman
2 Alice = fopen('alice_in_wonderland.txt', 'r'); % Abre o arquivo para
   leitura
3 Dados = fread(Alice, '*char'); % Lê o conteúdo do arquivo como caracteres
4 fclose(Alice); % Fecha o arquivo
5 Total = length(Dados); % Calcula o tamanho total dos dados
6
7 % Construir a tabela de frequência
8 i = 1;
9 B = [];
10 while i <= Total
11     if JaExiste(Dados(i), B)
12         B(end+1, 1) = double(Dados(i)); % Adiciona o caracter ao array B
13         B(end, 2) = double(sum(Dados == Dados(i))); % Conta a frequência do
           caracter
14     end
15     i = i + 1;
16 end
17
18 ordCar = sortrows(B, -2); % Ordena os caracteres pela frequência em ordem
   decrescente
19 Simbolos = char(ordCar(:, 1)); % Obtém os símbolos
20 ordCar(:, 2) = ordCar(:, 2) / Total; % Normaliza as frequências
21 numSimb = numel(ordCar(:, 1)); % Número de símbolos
22 ordSimb = transpose(ordCar(:, 1)); % Transpõe os símbolos
23 Nos = num2cell(char(ordSimb)); % Converte os símbolos em células
24 ordProb = transpose(ordCar(:, 2)); % Transpõe as probabilidades
25
26 for j = 1:numSimb-1
27     [~, indice] = min(ordProb); % Encontra o índice do menor valor de
       probabilidade
28     prob1 = ordProb(indice); % Probabilidade do menor valor
29     nol = Nos{indice}; % Nó do menor valor
30     ordProb(indice) = []; % Remove o menor valor de probabilidade
31     Nos(indice) = []; % Remove o menor nó
32
33     [~, indice] = min(ordProb); % Repete para o segundo menor valor
34     prob2 = ordProb(indice); % Probabilidade do segundo menor valor
35     no2 = Nos{indice}; % Nó do segundo menor valor
36     ordProb(indice) = []; % Remove o segundo menor valor de probabilidade
37     Nos(indice) = []; % Remove o segundo menor nó
38
39     novoNo = {nol, no2}; % Cria um novo nó combinando os dois menores nós
40     novaProb = prob1 + prob2; % Soma as probabilidades
41
42     id = find(ordProb >= novaProb, 1, 'last'); % Encontra a posição onde
       inserir o novo nó
43
44     if isempty(id)
45         id = numel(ordProb) + 1; % Se não encontrar, insere no final
46     end
47
48     ordProb = [ordProb(1:id-1), novaProb, ordProb(id:end)]; % Insere a nova
       probabilidade
49     Nos = [Nos(1:id-1), {novoNo}, Nos(id:end)]; % Insere o novo nó

```

```

50 end
51
52 Arvore = Nos{1}; % rvore de Huffman
53 dicionario = mapear(Arvore, ' '); % Cria o dicionário de Huffman
54
55 TextoCod = codTexto(Dados, dicionario); % Codifica o texto usando Huffman
56
57 %%
58
59 % Define matrizes de código de Hamming
60 H = [ 1 1 0 1 1 0 1 0 1 0 0 0 0;
61     1 0 1 1 0 1 1 0 0 1 0 0 0;
62     0 1 1 1 0 0 0 1 0 0 1 0 0;
63     0 0 0 0 1 1 1 1 0 0 0 1 0;
64     1 1 1 0 1 1 0 1 0 0 0 0 1];
65
66
67 c = size(H, 2); % Comprimento do código codificado
68 d = c - size(H, 1); % Comprimento dos bits de dados
69
70 % Define a matriz geradora G
71 % G =[ I | P' ]
72 I_k = eye(d);
73 P = H(:, 1:d); % Extrai a parte da matriz H que representa P
74 G = [I_k, P'];
75
76 % Converte string codificada de Huffman para bits
77 bits = double(TextoCod) - double('0');
78 bits = bits(:); % Transpõe para vetor coluna
79
80 % Adiciona padding para que o comprimento seja múltiplo de 8 (para o código
81 % Hamming (13,8))
82
83 % Codifica usando código Hamming (13,8)
84 dados = reshape(bits, d, []); %Faz fatias de 8 bits e a transposição
85 palavra_codigo = mod(double(dados) * G, 2); % Geração das palavras código
86 palavra_codigo = palavra_codigo';
87
88 % Converte os bits codificados em um vetor coluna
89 palavra_codigo = palavra_codigo(:);
90
91 %%
92 % Modulação 4-PSK
93 N = length(palavra_codigo);
94 cod_mod = zeros(N/2, 1);
95 for i = 1:2:N % Gera números ímpar
96     par = palavra_codigo(i:i+1); % Faz a fatia de 2 bits. Ex: i=1 -> (1:2),
97     % i=3 -> (3:4)
98     if isequal(par, [1; 1])
99         cod_mod((i+1)/2) = (1 + 1i); % 11 -> (1, 1)
100    elseif isequal(par, [1; 0])
101        cod_mod((i+1)/2) = (1 - 1i); % 10 -> (1, -1)
102    elseif isequal(par, [0; 1])
103        cod_mod((i+1)/2) = (-1 + 1i); % 01 -> (-1, 1)
104    else
105        cod_mod((i+1)/2) = (-1 - 1i); % 00 -> (-1, -1)
106    end
107 end

```

```

106 % Plotar a constelação
107 figure;
108 scatter(real(cod_mod), imag(cod_mod), 'filled');
109 title('Gráfico_de_Constelação_4-PSK');
110 xlabel('Parte_Real');
111 ylabel('Parte_Imaginária');
112 xlim([-6 6])
113 ylim([-6 6])
114 grid on;
115
116
117 % Demodulação 4-PSK
118 recebido = zeros(N, 1);
119 pontos_constel = [1+1i, 1-1i, -1+1i, -1-1i];
120 bit_map = [1 1; 1 0; 0 1; 0 0];
121
122 % Adicionar ruído e realizar demodulação para diferentes Es/No
123 Es_No_dB = 0:11; % Valores de Es/No em dB
124 num_snr = length(Es_No_dB);
125
126 ber_sem_correcao = zeros(length(Es_No_dB), 1);
127 ber_com_correcao = zeros(length(Es_No_dB), 1);
128 num_erros_matriz = zeros(length(Es_No_dB), 2); % Matriz para registrar número de erros
129
130 num_text_erros = zeros(length(Es_No_dB), 1);
131 for k = 1:num_snr
132     % Adicionar ruído usando a função add_awgn_noise
133     cod_ruido = add_awgn_noise(cod_mod, Es_No_dB(k));
134
135     % Plotar a constelação com ruído
136     figure;
137     scatter(real(cod_ruido), imag(cod_ruido), 1, 'filled');
138     hold on;
139     scatter(real(pontos_constel), imag(pontos_constel), 5, 'r', 'filled');
140     title(['Gráfico_de_Constelação_4-PSK_com_Ruído_(Es/No=' num2str(
141         Es_No_dB(k)) '_dB')]);
142     xlabel('Parte_Real');
143     ylabel('Parte_Imaginária');
144     xlim([-6 6]);
145     ylim([-6 6]);
146     grid on;
147     hold off;
148
149     % Demodulação 4-PSK
150     recebido = zeros(N, 1);
151     for i = 1:length(cod_ruido)
152         % Encontra o ponto da constelação mais próximo do símbolo recebido
153         [~, index] = min(abs(cod_ruido(i) - pontos_constel));
154         % Mapeia o índice do ponto encontrado para o par de bits correspondente
155         recebido(2*i-1:2*i) = bit_map(index, :);
156     end
157
158     % Calcular BER antes da correção
159     uncorrected_bits = reshape(recebido, c, []).';
160     uncorrected_bits = uncorrected_bits(:, 1:d).';

```

```

161 uncorrected_bits = uncorrected_bits(:);
162 uncorrected_bits = uncorrected_bits(1:length(bits));
163
164 num_errors_no_correction = sum(uncorrected_bits ~= bits);
165 ber_sem_correcao(k) = num_errors_no_correction / length(bits);
166
167
168 % Reshape os bits recebidos para decodificação de Hamming
169 entrada_hamming = reshape(recebido, c, []).';
170
171 % Calcular o síndrome
172 sindrome = mod(entrada_hamming * H.', 2);
173
174 % Corrigir erros com base no síndrome
175 for i = 1:size(sindrome, 1)
176     if any(sindrome(i, :))
177         for j = 1:size(H, 2)
178             if isequal(sindrome(i, :), H(:, j)') %Comparacao do
179                 sindrome com as colunas de H
180                 entrada_hamming(i, j) = mod(entrada_hamming(i, j) + 1,
181                     2); %Flip de correção
182                 break;
183             end
184         end
185     end
186
187 % Extraír bits originais corrigidos
188 corrected_bits = entrada_hamming(:, 1:d).';
189 corrected_bits = corrected_bits(:, );
190
191 % Ajustar o tamanho dos bits corrigidos para corresponder ao tamanho
192 % dos bits originais
193 corrected_bits = corrected_bits(1:length(bits));
194
195 % Calcular BER após correção
196 num_errors_with_correction = sum(corrected_bits ~= bits);
197 ber_com_correcao(k) = num_errors_with_correction / length(bits);
198
199 % Registrar o número de erros na matriz
200 num_erros_matriz(k, 1) = num_errors_no_correction;
201 num_erros_matriz(k, 2) = num_errors_with_correction;
202
203 % Decodificar o texto corrigido usando Huffman
204 TextoCod = char(corrected_bits' + '0');
205 TextoDecod = decodTexto(TextoCod, dicionario);
206
207 % Salvar o texto decodificado
208 Decodificado = fopen(['Decodificado_ ' num2str(Es_No_dB(k)) '.txt'], 'w'
209 );
210 disp(['Número_de_erros_no_texto_decodificado_para_Es/No_=_' num2str(
211 Es_No_dB(k)) '_dB:' num2str(num_text_errors(k))]);
212 fprintf(Decodificado, '%s', TextoDecod);
213 fclose(Decodificado);
214
215 disp(['Texto_decodificado_salvo_em_Decodificado_ ' num2str(Es_No_dB(k))
216 '.txt']);
217
218 end

```

```

213
214
215 % Exibir matriz de erros
216 disp('Matriz_de_erros_bits_(Número_de_Erros_sem_Correção,_Número_de_Erros_
217 com_Correção)');
217 disp(array2table(num_erro_matri, 'VariableNames', {'Erros_sem_Correcão',
218 'Erros_com_Correcão'}, 'RowNames', cellstr(num2str(Es_No_dB(:))));;
219 disp('Matriz_de_erros_no_texto_decodificado:');
220 disp(array2table(num_text_errors, 'VariableNames', {'Erros_no_Texto'}, 'RowNames',
221 cellstr(num2str(Es_No_dB(:))));;
222 figure; % Aumenta a largura da figura
223 semilogy(Es_No_dB, ber_sem_correcao, '-o', 'Color', [1 0.5 0], 'DisplayName'
224 ', 'Sem_Correção');
225 xlabel('Es/No_(dB)');
226 ylabel('Taxa_de_Erro_de_Bits_(BER)');
227 title('Gráfico_de_BER_sem_Correção');
228 legend('Location', 'best');
229 grid on;
230 pbaspect([4 1 1]);
231
232 % Gráfico de BER com Correção
233 figure; % Aumenta a largura da figura
234 semilogy(Es_No_dB, ber_com_correcao, '-x', 'DisplayName', 'Com_Correção');
235 xlabel('Es/No_(dB)');
236 ylabel('Taxa_de_Erro_de_Bits_(BER)');
237 title('Gráfico_de_BER_com_Correção');
238 legend('Location', 'best');
239 grid on;
240 pbaspect([4 1 1]);
241 % Gráfico Combinado
242 figure; % Aumenta a largura da figura
243 hold on;
244 semilogy(Es_No_dB, ber_com_correcao, '-x', 'DisplayName', 'Com_Correção');
245 semilogy(Es_No_dB, ber_sem_correcao, '-o', 'DisplayName', 'Sem_Correção');
246 xlabel('Es/No_(dB)');
247 ylabel('Taxa_de_Erro_de_Bits_(BER)');
248 title('Gráfico_de_BER_com_e_sem_Correção');
249 legend('Location', 'best');
250 grid on;
251 pbaspect([4 1 1]);
252 disp('Processo_concluído.');
253 %
254
255 % Funções Auxiliares
256
257 function checa = JaExiste(Dado, B)
258     n = 1;
259     while n <= length(B)
260         if Dado == B(n)
261             checa = false;
262             return;
263         end
264         n = n + 1;
265     end
266     checa = true;
267 end

```

```

267
268 function mapa = mapear(arvore, prefixo)
269     if iscell(arvore)
270         mapa = [mapear(arvore{1}, [prefixo '0']); mapear(arvore{2}, [
271             prefixo '1'])];
272     else
273         mapa = struct('simbolo', arvore, 'codigo', prefixo);
274     end
275 end
276
277 function codificar = codTexto(Dados, dicionario)
278     codificar = [];
279     for i = 1:length(Dados)
280         simbolo = Dados(i);
281         id = find([dicionario.simbolo] == simbolo);
282         codificar = [codificar dicionario(id).codigo];
283     end
284 end
285
286 function texto = decodTexto(codificar, dicionario)
287     texto = '';
288     codigo = '';
289     for i = 1:length(codificar)
290         codigo = [codigo codificar(i)];
291         id = find(strcmp({dicionario.codigo}, codigo));
292         if ~isempty(id)
293             texto = [texto dicionario(id).simbolo];
294             codigo = '';
295         end
296     end
297 end
298 % Função para adicionar ruído AWGN
299 function [r, n, N0] = add_awgn_noise(s, SNRdB, L)
300     s_temp = s;
301     if iscolumn(s), s = s.'; end % Para retornar o resultado no mesmo
302     % formato de 's'
303     gamma = 10^(SNRdB/10); % SNR para escala linear
304
305     if nargin == 2, L = 1; end % Se o terceiro argumento não for dado,
306     % define como 1
307
308     if isvector(s)
309         P = L * sum(abs(s).^2) / length(s); % Potência real no vetor
310     else % Para sinais multidimensionais como MFSK
311         P = L * sum(sum(abs(s).^2)) / length(s); % Se s é uma matriz [MxN]
312     end
313
314     N0 = P / gamma; % Encontra a densidade espectral do ruído
315     if isreal(s)
316         n = sqrt(N0 / 2) * randn(size(s)); % Ruído calculado
317     else
318         n = sqrt(N0 / 2) * (randn(size(s)) + 1j * randn(size(s))); % Ruído
319         % calculado
320     end
321
322     r = s + n; % Sinal recebido

```

```
321     if iscolumn(s_temp), r = r.'; end % Retorna r no formato original como  
322     s  
end
```