



**INSTITUTO
FEDERAL**

Santa Catarina

Câmpus
São José

Sistema de comunicação com espalhamento espectral por sequência direta DSSS

Sistemas de Comunicação II

Arthur Cadore Matuella Barcella

03 de Fevereiro de 2024

Engenharia de Telecomunicações - IFSC-SJ

Sumário

1. Introdução	3
2. Simulação	3
2.1. Definição dos Parâmetros do Sistema	3
2.2. Gerando o Código de Sequência Direta Maxima	3
2.3. Definição dos Parâmetros de Usuário	4
2.4. Laço de Simulação	4
2.5. Resultados	6
2.5.1. Curva BER para o Primeiro Usuário	6
2.5.2. Curva BER para o Conjunto de Usuários	6
3. Conclusão	7

1. Introdução

O objetivo deste trabalho é simular um sistema de comunicação com espalhamento espectral por sequência direta DSSS (Direct Sequence Spread Spectrum), que é um tipo de técnica de modulação digital que é usada para espalhar o sinal de dados antes da transmissão.

2. Simulação

2.1. Definição dos Parâmetros do Sistema

Inicialmente, definimos os parâmetros do sistema, como o número de flip-flops, o número de símbolos transmitidos por iteração, o intervalo de E_b/N_0 em dB, o número de iterações para cada E_b/N_0 , o comprimento do código de espalhamento e a modulação 8-PSK.

```
1 # Número de flip-flops para o código de sequência máxima
2 numFlipFlops = 6
3
4 # Número de símbolos transmitidos por iteração
5 numSimbolos = 1000
6
7 # Intervalo de  $E_b/N_0$  em dB
8 Eb_N0_dB = np.arange(0, 24, 1)
9
10 # Número de iterações para cada  $E_b/N_0$ 
11 numIteracoes = 10000
12
13 # Comprimento do código de espalhamento
14 comprimentoCodigo = (2**numFlipFlops) - 1
15
16 # Modulação 8-PSK
17 M = 8
18 bitsPorSimbolo = int(np.log2(M))
```

Também neste paço calculamos o número de bits por símbolo, que é igual a 3, pois a modulação 8-PSK possui 8 símbolos, e 3 bits são necessários para representar cada símbolo.

2.2. Gerando o Código de Sequência Direta Maxima

Agora, geramos o código de sequência direta máxima, que é um código de espalhamento que é usado para espalhar o sinal de dados antes da transmissão. O código de sequência máxima é gerado a partir de um registro de deslocamento de sequência máxima, que é um tipo de registrador de deslocamento que é usado para gerar sequências pseudoaleatórias.

```
1 # Inicializando vetores de Registro de Deslocamento de Sequência Máxima
2 estadoReg = np.array([0, 1, 1, 1, 1, 1])
3 codigoBase = np.zeros(comprimentoCodigo)
4
5 # Gera o código de sequência máxima
6 for i in range(comprimentoCodigo):
7     novoBit = (estadoReg[0] + estadoReg[5]) % 2
8     estadoReg = np.roll(estadoReg, 1)
```

```

9     estadoReg[0] = novoBit
10    codigoBase[i] = estadoReg[5]
11
12    # Converte para {-1, +1} e normaliza para energia unitária
13    codigoBase = 2 * codigoBase - 1
14    codigoBase /= np.sqrt(comprimentoCodigo)

```

2.3. Definição dos Parâmetros de Usuário

Em seguida, para iniciar o laço de simulação, definimos os códigos de sequência de cada usuário, que são obtidos a partir do código de sequência máxima, deslocando-o em 5 e 10 posições, respectivamente. Portanto, temos três códigos de sequência de usuário, que são usados para espalhar o sinal de dados antes da transmissão. Abaixo também inicializamos vetores para armazenar a BER individual de cada usuário.

```

1  # Definição de parâmetros para simulação:
2
3  # Atribuição de códigos aos usuários
4  codigoUsuario1 = codigoBase
5  codigoUsuario2 = np.roll(codigoBase, 5)
6  codigoUsuario3 = np.roll(codigoBase, 10)
7
8  # Inicializa vetor para BER, individual por usuário.
9  BER_Usuario1 = np.zeros(len(Eb_N0_dB))
10 BER_Usuario2 = np.zeros(len(Eb_N0_dB))
11 BER_Usuario3 = np.zeros(len(Eb_N0_dB))

```

2.4. Laço de Simulação

O laço de simulação é dividido em três diferentes sessões, sendo elas apresentadas abaixo:

1. Cálculo da variância do ruído e inicialização do vetor para contagem de erros, individual por usuário. Nesta etapa também é calculada a BER para cada usuário.

```

1  # laço de repetição, para varrer todos os valores do vetor de SNR.
2  for idx, Eb_N0 in enumerate(Eb_N0_dB):
3
4      # Cálculo da variância do ruído
5      EbN0_linear = 10**(Eb_N0 / 10)
6      sigma = np.sqrt(1 / (2 * bitsPorSimbolo * EbN0_linear))
7
8      # Inicializa vetor para contagem de erros, individual por usuário
9      numErros = np.zeros(3)
10     numBitsTotal = numSimbolos * bitsPorSimbolo * numIteracoes
11
12     # Cálculo da BER para cada usuário
13     BER_Usuario1[idx], BER_Usuario2[idx], BER_Usuario3[idx] =
14     numErros / numBitsTotal

```

Em seguida, mantendo esses parâmetros fixos, é iniciado um novo laço de repetição, que varre todas as iterações para cada valor de Eb/N0. Neste laço, são gerados bits aleatórios para os

usuários, que são convertidos em símbolos 8-PSK. Os símbolos são transmitidos e recebidos, e então é feito o desespalhamento e a decisão de símbolos.

```
1 # Laço de repetição, mantendo o valor de SNR e testando diversas interações
2 for _ in range(numIteracoes):
3
4     # Gera bits aleatórios para os usuários
5     bitsUsuarios = np.random.randint(0, 2,
6                                       (3, numSimbolos * bitsPorSimbolo))
7
8     simbolos = np.exp(1j * 2 * np.pi * np.dot(bitsUsuarios.reshape(
9         3, -1, bitsPorSimbolo), (2**np.arange(bitsPorSimbolo)[::-1])) / M)
10
11     # Transmissão e recepção
12     TX_total = sum(np.outer(codigoBase, simbolos[i, :]).flatten() for i,
13                       codigoBase in enumerate(
14                           [codigoUsuario1,
15                            codigoUsuario2,
16                            codigoUsuario3]))
17
18     RX_total = TX_total + sigma * (np.random.randn(*TX_total.shape)
19                                     + 1j * np.random.randn(*TX_total.shape))
20
21     RX_matriz = RX_total.reshape(comprimentoCodigo, numSimbolos)
```

E dentro deste laço de repetição, mantendo os parâmetros os mesmos, é feito o desespalhamento e a decisão de símbolos. Para isso, é calculado o produto interno entre a matriz de recepção e o código do usuário, e então é feita a decisão de símbolos, a decisão é feita em um terceiro laço, onde os símbolos são convertidos em bits, e então é feita a comparação dos bits recebidos com os bits dos usuários.

```
1 # Desespalhamento e decisão de símbolos
2     for i, codigoBase in enumerate(
3         [codigoUsuario1,
4          codigoUsuario2,
5          codigoUsuario3]):
6
7         # Verifica se o código do usuário está presente
8         sinaisDesespalhados = np.sum(RX_matriz *
9                                       codigoBase[:, None], axis=0)
10
11         fasesRecebidas = np.angle(sinaisDesespalhados)
12
13         decisao = np.round(fasesRecebidas / (2 * np.pi / M)) % M
14
15         # Converter símbolos para bits corretamente
16         bitsRecebidos = np.array([list(np.binary_repr(int(d),
17                                       width=bitsPorSimbolo)) for d in decisao], dtype=int).flatten()
18
19         # Comparar apenas os bits relevantes
20         numErros[i] += np.sum(bitsRecebidos[
21                                 :len(bitsUsuarios[i, :])] != bitsUsuarios[i, :])
```

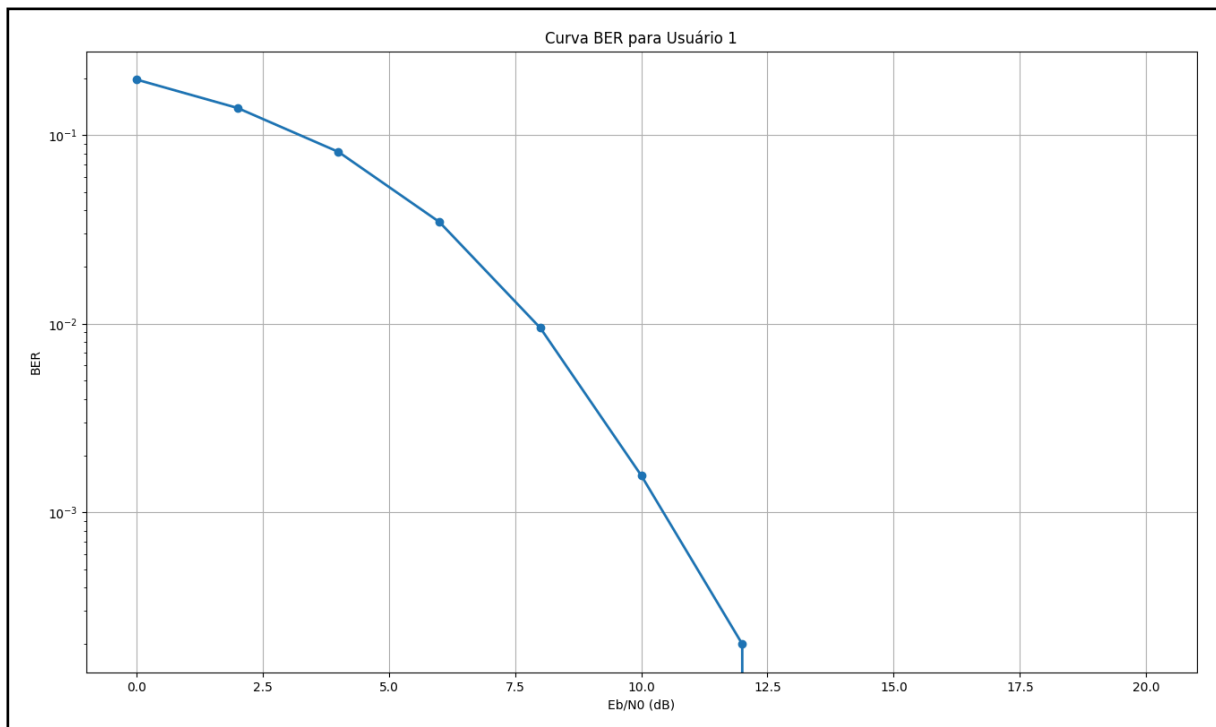
2.5. Resultados

2.5.1. Curva BER para o Primeiro Usuário

Abaixo é apresentado o gráfico da curva BER para o primeiro usuário, que é o usuário desejado. A curva BER é plotada em escala logarítmica, onde o eixo x representa a relação sinal-ruído E_b/N_0 em dB, e o eixo y representa a taxa de erro de bits (BER).

```
1 # Plot da Curva BER para o Usuário 1
2 plt.figure(figsize=(16,9))
3 plt.semilogy(Eb_N0_dB, BER_Usuario1, 'o-', linewidth=2)
4 plt.xlabel('Eb/N0 (dB)')
5 plt.ylabel('BER')
6 plt.title('Curva BER para Usuário 1')
7 plt.grid(True)
8 plt.show()
```

Figura 1: Elaborada pelo Autor



2.5.2. Curva BER para o Conjunto de Usuários

Abaixo é apresentado o gráfico comparativo da curva BER para os três usuários, onde é possível observar a taxa de erro de bits para cada usuário em relação à relação sinal-ruído E_b/N_0 em dB.

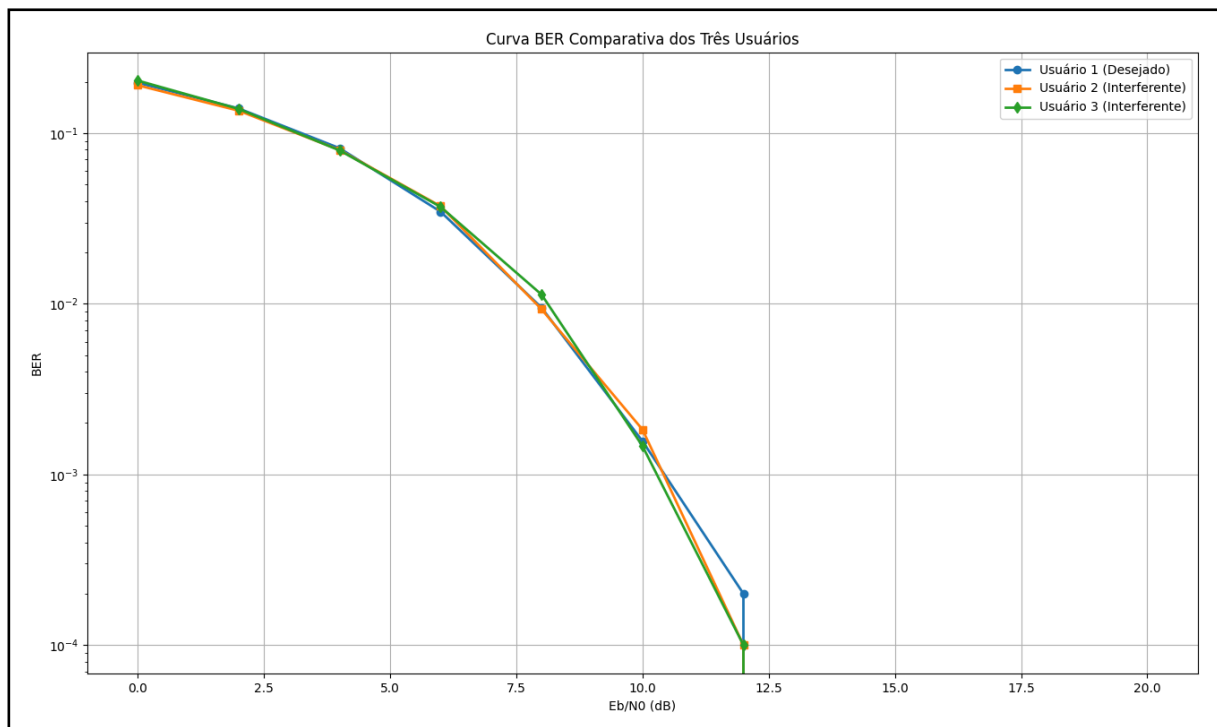
```
1 # Plot Comparativo BER para os Três Usuários
2 plt.figure(figsize=(16,9))
3 plt.semilogy(Eb_N0_dB, BER_Usuario1, 'o-', linewidth=2, label='Usuário 1
  (Desejado)')
```

```

4 plt.semilogy(Eb_N0_dB, BER_Usuario2, 's-', linewidth=2, label='Usuário 2
  (Interferente)')
5 plt.semilogy(Eb_N0_dB, BER_Usuario3, 'd-', linewidth=2, label='Usuário 3
  (Interferente)')
6 plt.xlabel('Eb/N0 (dB)')
7 plt.ylabel('BER')
8 plt.title('Curva BER Comparativa dos Três Usuários')
9 plt.legend()
10 plt.grid(True)
11 plt.show()

```

Figura 2: Elaborada pelo Autor



3. Conclusão

Neste trabalho, foi simulado um sistema de comunicação com espalhamento espectral por sequência direta DSSS, que é um tipo de técnica de modulação digital que é usada para espalhar o sinal de dados antes da transmissão.

Foram definidos os parâmetros do sistema, gerado o código de sequência direta máxima, definidos os códigos de sequência de cada usuário, e realizado o laço de simulação para calcular a BER para cada usuário. Os resultados mostraram que a taxa de erro de bits (BER) diminui à medida que a relação sinal-ruído E_b/N_0 aumenta, e que a BER é menor para o usuário desejado em comparação com os usuários interferentes.