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

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. Definicação 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 Eb/N0 em dB, o número de iterações para cada Eb/N0, o comprimento do código de espalhamento e a modulação 8-PSK.

```
# Número de flip-flops para o código de sequência máxima
numFlipFlops = 6

# Número de símbolos transmitidos por iteração
numSimbolos = 1000

# Intervalo de Eb/N0 em dB
Eb_N0_dB = np.arange(0, 24, 1)

# Número de iterações para cada Eb/N0
numIteracoes = 10000

# Comprimento do código de espalhamento
comprimentoCodigo = (2**numFlipFlops) - 1

# Modulação 8-PSK
M = 8
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.

```
# Inicializando vetores de Registro de Deslocamento de Sequência Máxima
estadoReg = np.array([0, 1, 1, 1, 1, 1])
codigoBase = np.zeros(comprimentoCodigo)

# Gera o código de sequência máxima
for i in range(comprimentoCodigo):
    novoBit = (estadoReg[0] + estadoReg[5]) % 2
    estadoReg = np.roll(estadoReg, 1)
```

```
    estadoReg[0] = novoBit
    codigoBase[i] = estadoReg[5]

# Converte para {-1, +1} e normaliza para energia unitária
codigoBase = 2 * codigoBase - 1
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.

```
# Definição de parâmetros para simulação:

# Atribuição de códigos aos usuários
codigoUsuario1 = codigoBase
codigoUsuario2 = np.roll(codigoBase, 5)
codigoUsuario3 = np.roll(codigoBase, 10)

# Inicializa vetor para BER, individual por usuário.
BER_Usuario1 = np.zeros(len(Eb_N0_dB))
BER_Usuario2 = np.zeros(len(Eb_N0_dB))
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.

```
# laço de repetiação, para varrer todos os valores do vetor de SNR.
for idx, Eb_N0 in enumerate(Eb_N0_dB):

    # Cálculo da variância do ruído
    EbN0_linear = 10**(Eb_N0 / 10)
    sigma = np.sqrt(1 / (2 * bitsPorSimbolo * EbN0_linear))

    # Inicializa vetor para contagem de erros, individual por usuário
    numErros = np.zeros(3)
    numBitsTotal = numSimbolos * bitsPorSimbolo * numIteracoes

  # Cálculo da BER para cada usuário
    BER_Usuario1[idx], BER_Usuario2[idx], BER_Usuario3[idx] =
    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.

```
# Laço de repetição, mantendo o valor de SNR e testando diversas interações
for _ in range(numIteracoes):

    # Gera bits aleatórios para os usuários
    bitsUsuarios = np.random.randint(0, 2,
                   (3, numSimbolos * bitsPorSimbolo))

    simbolos = np.exp(1j * 2 * np.pi * np.dot(bitsUsuarios.reshape(
    3, -1, bitsPorSimbolo),(2**np.arange(bitsPorSimbolo)[::-1])) / M)

    # Transmissão e recepção
    TX_total = sum(np.outer(codigoBase, simbolos[i, :]).flatten() for i,
    codigoBase in enumerate(
    [codigoUsuario1,
     codigoUsuario2,
     codigoUsuario3]))

    RX_total = TX_total + sigma * (np.random.randn(*TX_total.shape)
    + 1j * np.random.randn(*TX_total.shape))

    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.

```
# Desespalhamento e decisão de símbolos
       for i, codigoBase in enumerate(
             [codigoUsuario1,
              codigoUsuario2,
              codigoUsuario3]):

           # Verifica se o código do usuário está presente
           sinaisDesespalhados = np.sum(RX_matriz *
           codigoBase[:, None], axis=0)

           fasesRecebidas = np.angle(sinaisDesespalhados)

           decisao = np.round(fasesRecebidas / (2 * np.pi / M)) % M

           # Converter símbolos para bits corretamente
           bitsRecebidos = np.array([list(np.binary_repr(int(d),
           width=bitsPorSimbolo)) for d in decisao], dtype=int).flatten()

            # Comparar apenas os bits relevantes
           numErros[i] += np.sum(bitsRecebidos[
             :len(bitsUsuarios[i, :])] != bitsUsuarios[i, :])
```

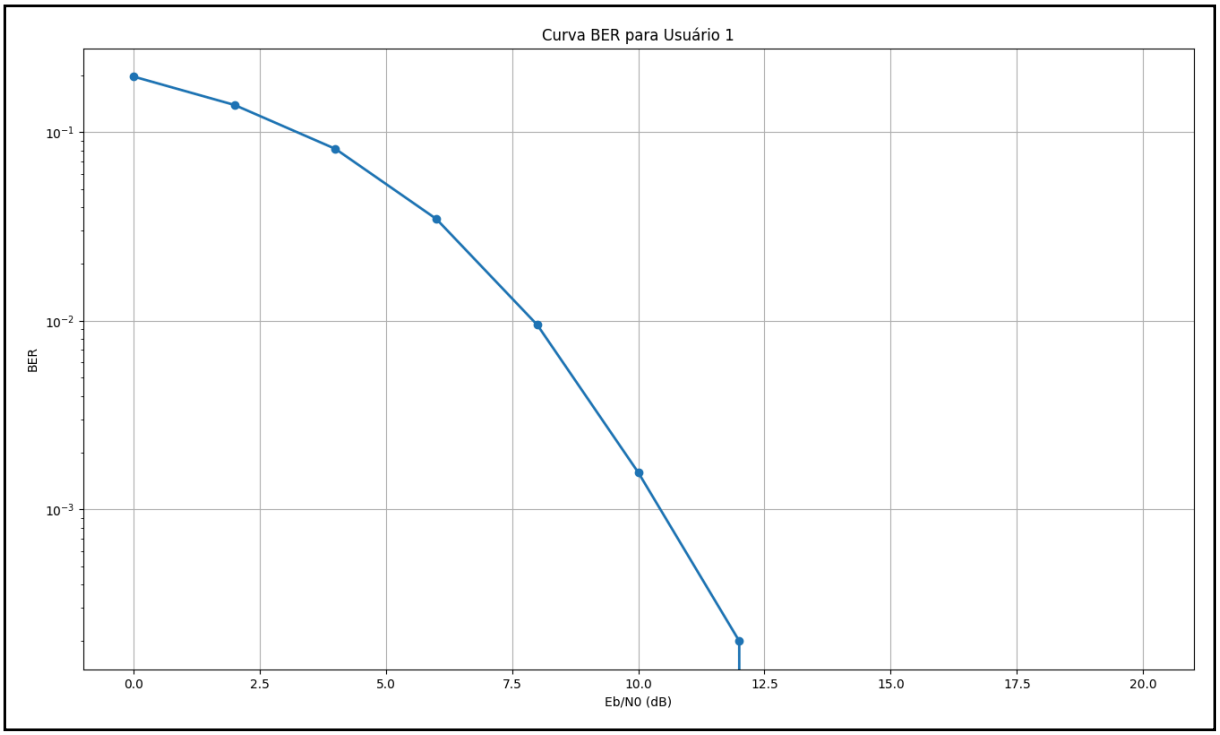
2.5. Resultados

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 Eb/N0 em dB, e o eixo y representa a taxa de erro de bits (BER).

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

Figura 1: Elaborada pelo Autor



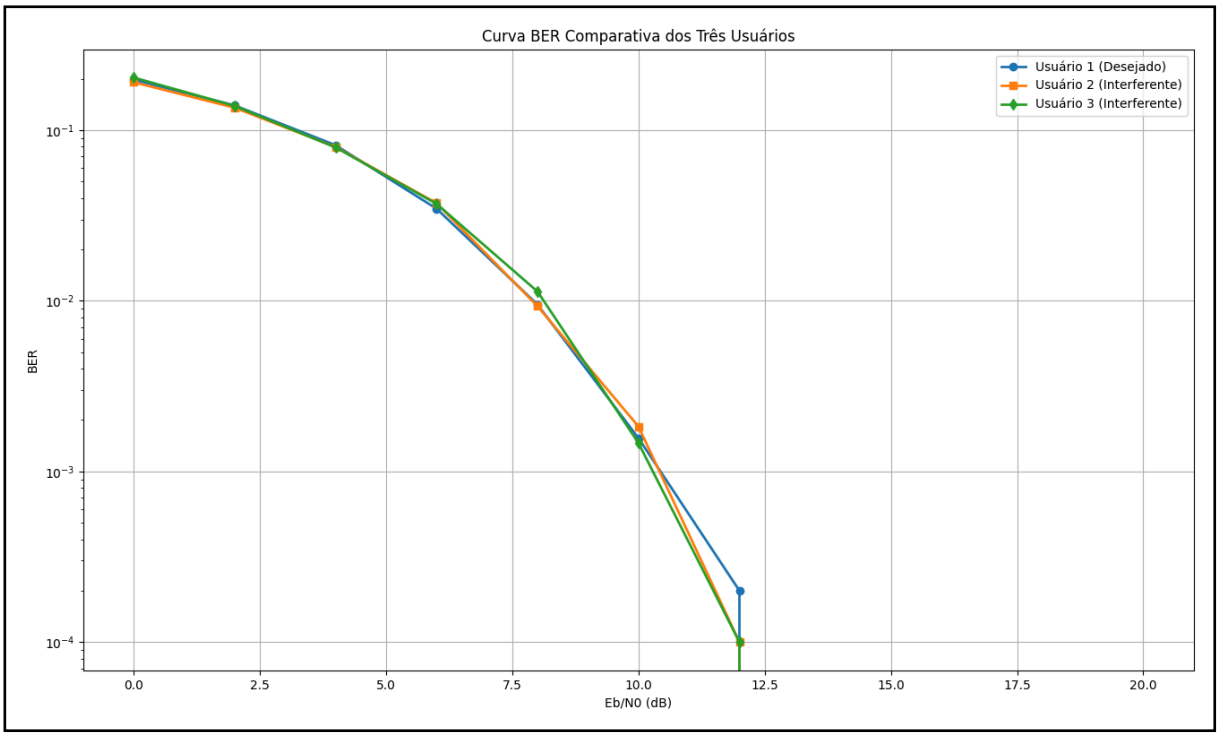
2.5.2. Curva BER para o Conjunto de Usuários

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 Eb/N0 em dB.

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

```
plt.semilogy(Eb_N0_dB, BER_Usuario2, 's-', linewidth=2, label='Usuário 2 (Interferente)')
plt.semilogy(Eb_N0_dB, BER_Usuario3, 'd-', linewidth=2, label='Usuário 3 (Interferente)')
plt.xlabel('Eb/N0 (dB)')
plt.ylabel('BER')
plt.title('Curva BER Comparativa dos Três Usuários')
plt.legend()
plt.grid(True)
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 Eb/N0 aumenta, e que a BER é menor para o usuário desejado em comparação com os usuários interferentes.