

Código para modulação e demodulação de um sinal 4-pam, considerando que o sinal passou por um canal com ruído.

Forma geral sinal MPAM

$$S_i(t) = A \left(i - \frac{M-1}{2} \right) g(t)$$

Onde:

$$i = 0, 1, \dots, (M-1)$$

$g(t)$: funcao de modelagem de pulso

Sinal 4-PAM

$$S_i(t) = 2 \left(i - \frac{4-1}{2} \right) g(t)$$

M: número de níveis

$$M = 4 = 2^2$$

A: Amplitud

$$A = 2$$

$$S_i(t) = 2 \left(i - \frac{3}{2} \right)$$

$$i = 0 \rightarrow 2 \left(0 - \frac{3}{2} \right) = -3$$

$$i = 1 \rightarrow 2 \left(1 - \frac{3}{2} \right) = -1$$

$$i = 2 \rightarrow 2 \left(2 - \frac{3}{2} \right) = 1$$

$$i = 3 \rightarrow 2 \left(3 - \frac{3}{2} \right) = 3$$

$$S = \{-3, -1, 1, 3\}$$

Geração de sinal 4-PAM

Nesse código, a função **generate_4pam_signal** recebe uma sequência de bits como entrada e gera um sinal 4-PAM.

Primeiro, devemos mapear os bits para símbolos 4-PAM.

Para isso, criamos um dicionário **symbol_map** que mapeia bits para símbolos. Para isso, temos que dividir os bits em pares, pois temos apenas 4 níveis.

Dividimos a sequência de bits em pares de dois com o seguinte algoritmo que agrupa (**bits[i:i+2]**) e os converte em símbolos usando o dicionário de mapeamento. Esses símbolos representarão os níveis de amplitude do sinal.

Exemplo:

Si for $i = 0 \rightarrow$ bits [0:2]

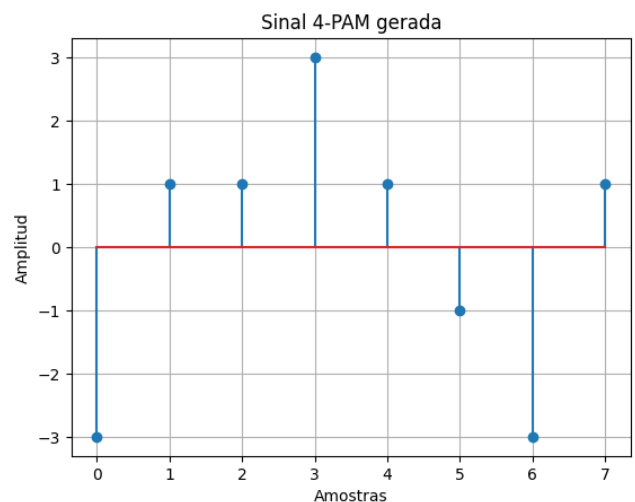
Resultado 00

Mapeamento 00 = -3

Si for $i = 2 \rightarrow$ bits [2:4]

Resultado 10

Mapeamento 00 = 1



```
def generate_4pam_signal(bits):
    #Mapeamento bit a símbolo 4-PAM
    symbol_map = {'00': -3, '01': -1, '10': 1, '11': 3}
    symbols=np.array([])
    # Converter bits em símbolos
    for i in range(0, len(bits), 2):
        b = bits[i:i+2]
        symbol = symbol_map[b]
        symbols = np.append(symbols, symbol)

    return symbols
print(symbols)
# Exemplo de uso
bits = "0010101110010010" # Exemplo de sequência de bits
symbols = generate_4pam_signal(bits)

plt.stem(symbols, use_line_collection=True)
plt.title("Sinal 4-PAM gerada")
plt.xlabel("Amostras")
plt.ylabel("Amplitud")
plt.grid(True)
plt.show()
```

Demodulação sem ruído

Repetimos o mesmo processo, mas na ordem inversa.

Os símbolos 4-PAM são mapeados para as combinações de bits correspondentes.

Em seguida, uma cadeia de **bits** vazia é inicializada. Cada símbolo é comparado com a lista de símbolos, usando a função **min** com uma função de **chave** que calcula o valor absoluto da diferença entre o símbolo de entrada e cada símbolo.

symbol_map_inverse.keys(): Retorna uma lista de todas as chaves no dicionário **symbol_map_inverse**, que, nesse caso, são os símbolos 4-PAM (-3, -1, 1, 3).

key=lambda x: abs(x - symbol): define uma função (lambda) que recebe um símbolo x e retorna o valor absoluto da diferença entre x e o símbolo de entrada. Essa função é usada para determinar qual símbolo está mais próximo do símbolo de entrada.

Neste exemplo, adicionamos ruído ao sinal, onde geramos um ruído aleatório com uma distribuição normal, atribuímos a ele um valor médio de 0 e uma variância de 0,5.

Por fim, a combinação de bits correspondente ao símbolo mais próximo é pesquisada. Esse é o motivo pelo qual nem sempre é possível obter uma boa modulação, devido ao fato de que, ao adicionar um ruído, é possível que, ao fazer a operação de diferença, obtenhamos um bit ao qual as informações iniciais não correspondiam.

O resultado da demodulação dependerá do ruído do nosso sinal.

```
def add_noise(signal, noise_variance):
    noise = np.random.normal(0, np.sqrt(noise_variance), len(signal))
    return signal + noise

def demodulate_4pam_signal(symbols, noise_variance):
    # Mapeamento inverso de símbolos para bits
    symbol_map_inverse = {-3: '00', -1: '01', 1: '10', 3: '11'}

    # Adicionando ruído aos símbolos
    symbols_noisy = add_noise(symbols, noise_variance)


    # Demodulação de símbolos
    bits = ""
    for symbol in symbols_noisy:
        # Encontre o símbolo mais próximo na ausência de ruído
        closest_symbol = min(symbol_map_inverse.keys(), key=lambda x: abs(x - symbol))
        # Converta o símbolo para o bit correspondente
        bits += symbol_map_inverse[closest_symbol]

    return bits
```

Nesse caso, obtivemos os mesmos valores

```
# Exemplo de uso
bits = "0010101110010010" # Exemplo de sequência de bits
symbols = generate_4pam_signal(bits)

plt.stem(symbols, use_line_collection=True)
plt.title("Sinal 4-PAM gerada")
plt.xlabel("Amostras")
plt.ylabel("Amplitud")
plt.grid(True)
plt.show()
```

 [-3. 1. 1. 3. 1. -1. -3. 1.]

```
# Exemplo de uso
noise_variance = 0.5 # Variação de ruído
received_bits = demodulate_4pam_signal(symbols, noise_variance)

print("Bits recibidos después de demodulación con ruido:", received_bits)
```

Bits recibidos después de demodulación con ruido: 0010101110010010
