

Relatório de Simulação

Karen dos Anjos Arcoverde

Introdução

Nesta seção apresenta-se a avaliação do desempenho de distintos esquemas de modulação digital (M-PAM, M-QAM e M-PSK) em canal AWGN, tanto em configuração banda-base (processamento direto dos símbolos) quanto em banda-passante (transmissão sobre portadora), empregando pulso Raiz-Cosseno Levantado e filtragem casada. Além disso, são examinadas as constelações Tx/Rx no plano I/Q para cada valor de E_b/N_0 . O objetivo principal é comparar, para diversas ordens de modulação e valores de E_b/N_0 .

Resultados Obtidos

Questão 1

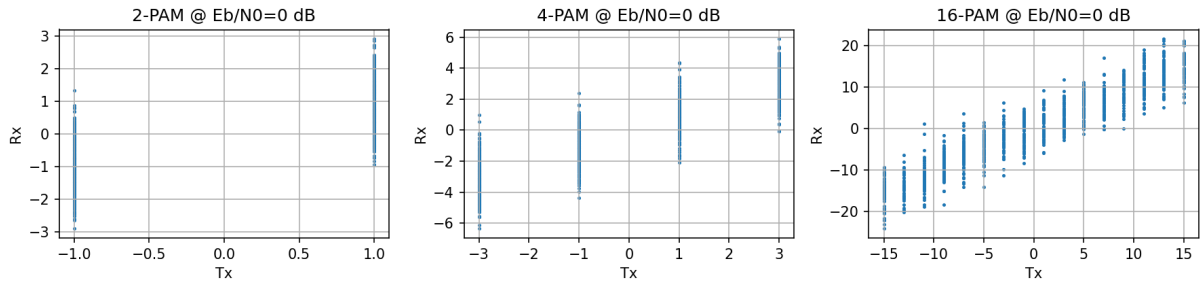


Figure 1: $E_b/N_0=0$ dB para M-PAM.

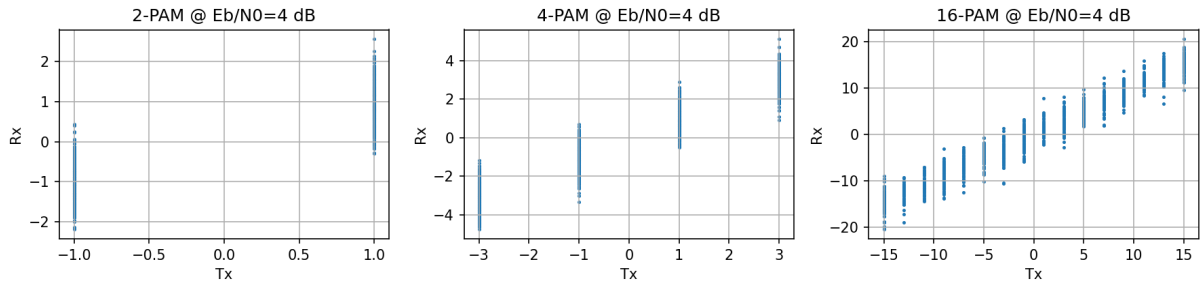


Figure 2: $E_b/N_0=4$ dB para M-PAM.

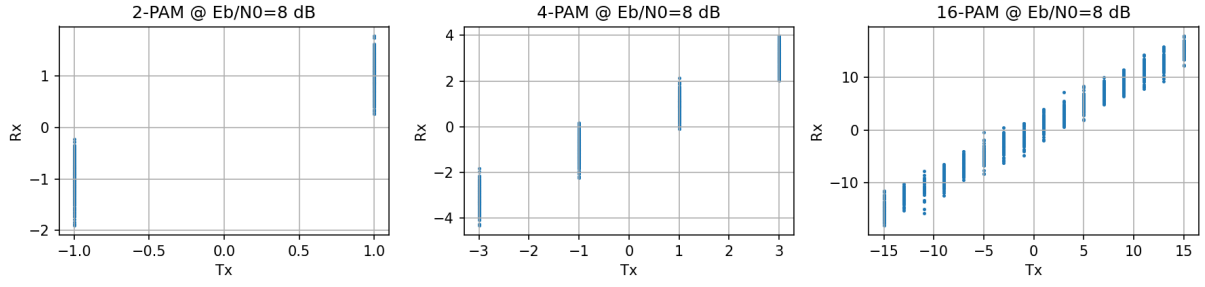


Figure 3: $E_b/N_0=8$ dB para M-PAM.

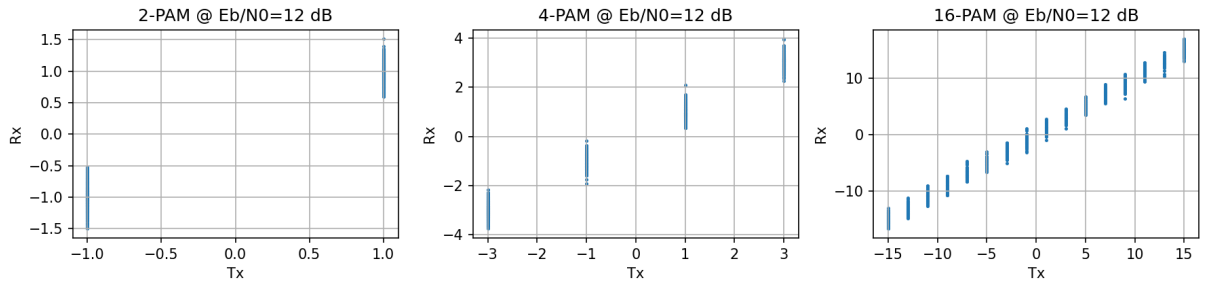


Figure 4: $E_b/N_0=12$ dB para M-PAM.

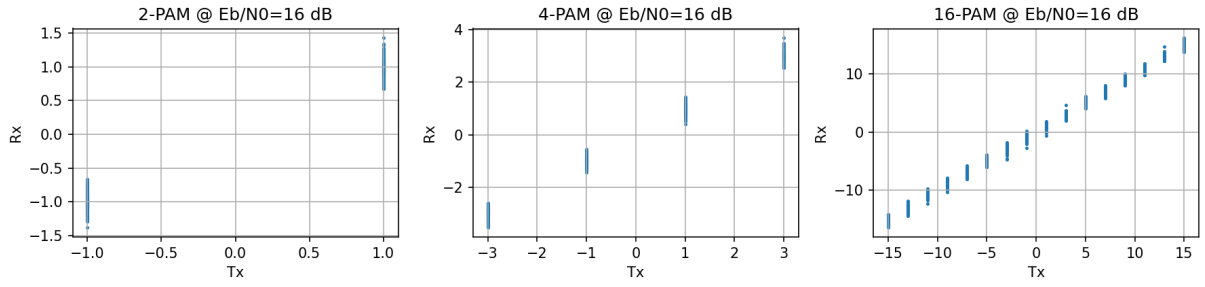


Figure 5: $E_b/N_0=16$ dB para M-PAM.

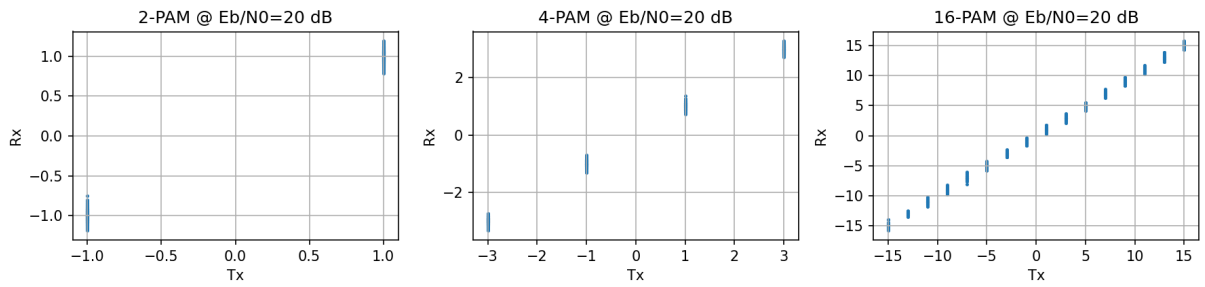


Figure 6: $E_b/N_0=20$ dB para M-PAM.

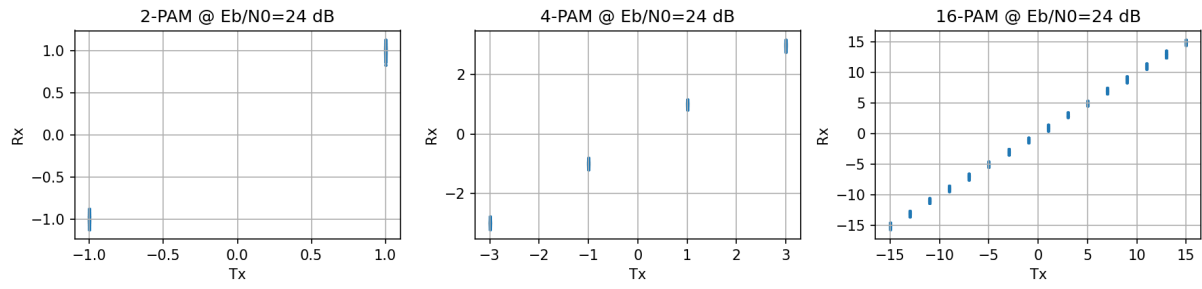


Figure 7: $E_b/N_0 = 24$ dB para M-PAM.

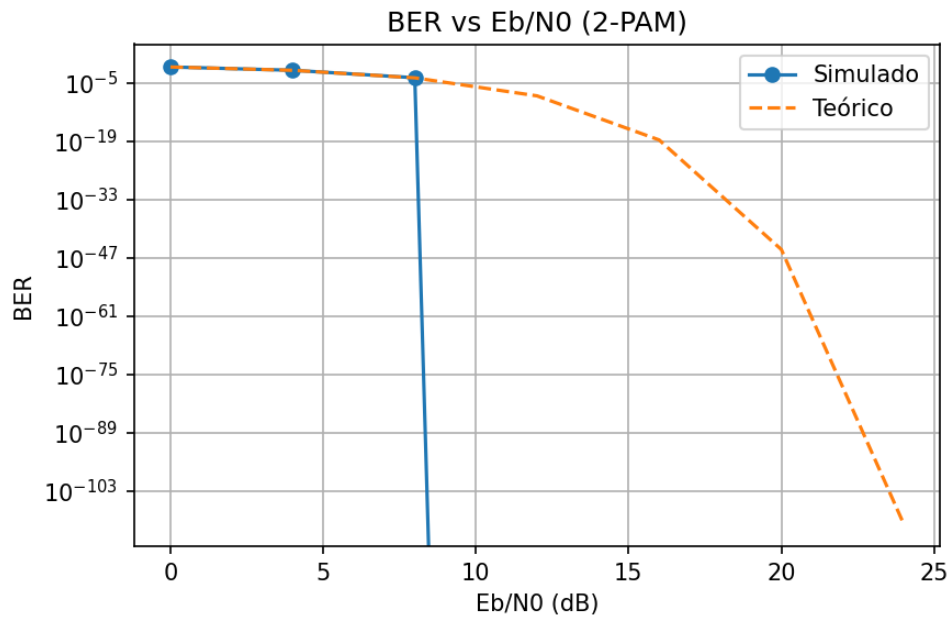


Figure 8: BER vs E_b/N_0 – 2-PAM.

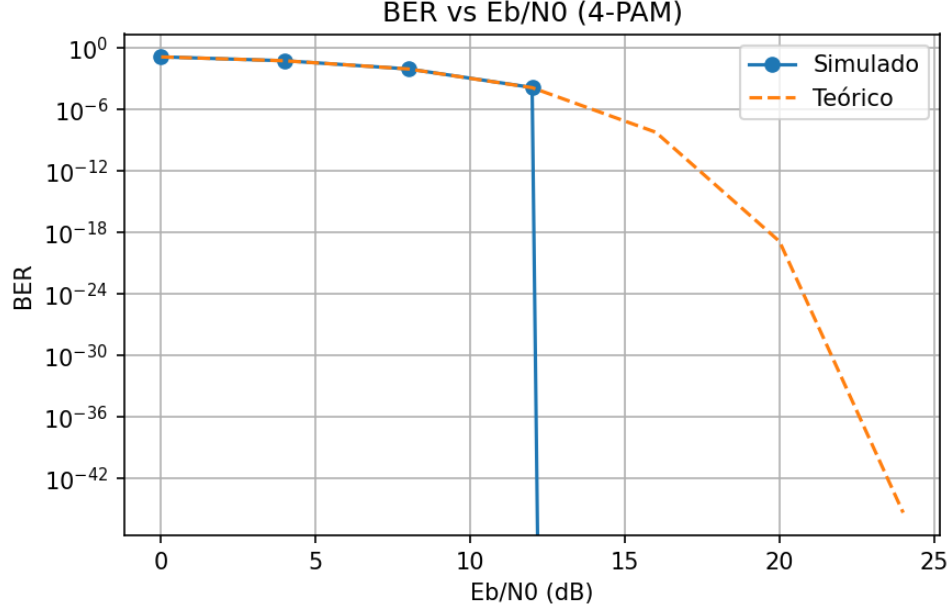


Figure 9: BER vs E_b/N_0 – 4-PAM.

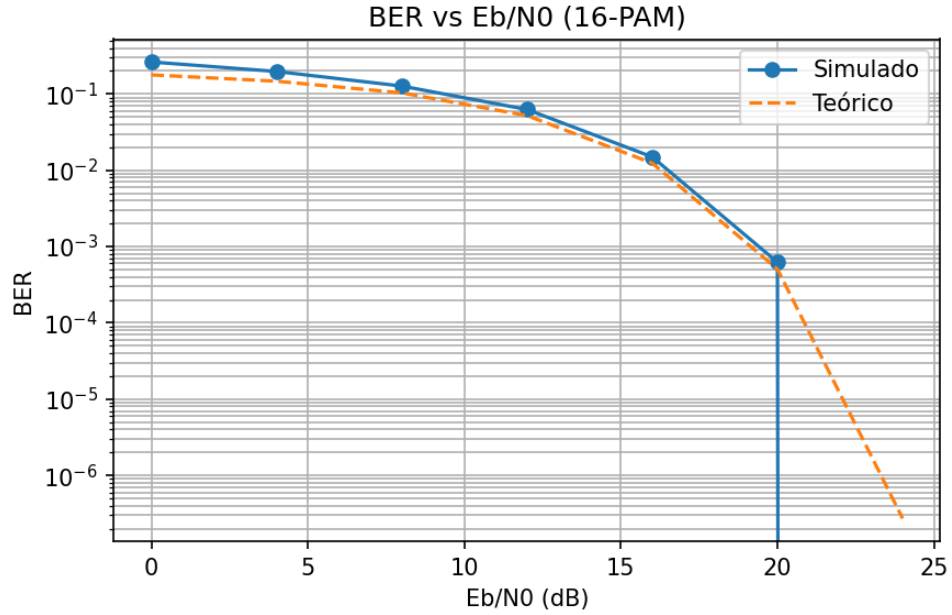


Figure 10: BER vs E_b/N_0 – 16-PAM.

A expressão teórica de BER utilizada baseia-se no cálculo da probabilidade de erro de símbolo para M-PAM em canal AWGN com detetor de máxima verossimilhança. A probabilidade de erro de símbolo P_e ou SER é dada por:

$$P_e = \frac{2(M-1)}{M} Q\left(\sqrt{\frac{6 \log_2(M)}{M^2-1} \frac{E_b}{N_0}}\right)$$

Para converter a SER (probabilidade de erro por símbolo) para a BER (probabilidade

de erro de bit), dividimos pelo número de bits por símbolo:

$$\text{BER} \approx \frac{P_e}{\log_2(M)}$$

Observa-se que pequenas discrepâncias entre os valores simulados e as previsões teóricas devem-se a limitações práticas de cálculo: quanto maior o **span** (extensão do filtro Raiz-Cosseno Levantado em unidades de símbolo), mais próximo o pulso fica do ideal (menor ISI), mas também cresce linearmente o custo computacional de cada convolução. Além disso, aumentar o número de bits para estimar o BER (**num_bits**) incrementaria exponencialmente o tempo de execução, sobrecarregando o processador e obrigando a interromper a simulação antes de atingir a resolução desejada. Caso fosse possível elevar significativamente ambos (**span** e **num_bits**), as curvas simulada e teórica tenderiam a coincidir.

Os *diagramas de constelação* evidenciam a dispersão dos símbolos provocada pelo ruído. Em razões de E_b/N_0 reduzidas (por exemplo, 0–4 dB), as nuvens de pontos apresentam ampla dispersão e sobreposição, resultando em elevadas taxas de erro. À medida que E_b/N_0 aumenta, essas nuvens contraem-se em torno dos níveis ideais (± 1 no caso de 2-PAM, $\pm 1, \pm 3$ na 4-PAM etc.), o que facilita significativamente a distinção entre os símbolos.

Nas *curvas de BER* versus E_b/N_0 , observa-se um decaimento quase exponencial: cada incremento de cerca de 4 dB reduz a BER em aproximadamente uma ordem de magnitude.

Ao comparar as curvas de BER para distintos ordens de modulação M-PAM, verifica-se que, para um mesmo valor de E_b/N_0 , a taxa de erro cresce com o aumento de M . Por exemplo, em $E_b/N_0 = 8$ dB, o BER aproximado é da ordem de 10^{-3} para 2-PAM, de 10^{-2} para 4-PAM e de 10^{-1} para 16-PAM. Isso ocorre porque, ao dobrar o número de níveis de amplitude, duplica-se também o número de limiares de decisão, tornando os símbolos mais próximos e, portanto, mais suscetíveis a confusões induzidas pelo ruído.

Além disso, embora todas as curvas decaiam quase exponencialmente, a inclinação efetiva (o “declive” da curva) torna-se menos acentuada para modulações de ordem mais alta. Em 2-PAM, a redução de BER entre 4 dB e 8 dB pode alcançar cerca de duas ordens de magnitude, enquanto em 16-PAM o mesmo incremento de 4 dB costuma representar apenas meia a uma ordem de magnitude. Isso reflete o fato de que modulações de ordem maior exigem ganhos de E_b/N_0 progressivamente maiores para atingir níveis de erro equivalentes aos observados em modulações mais simples.

Questão 2

M-QAM

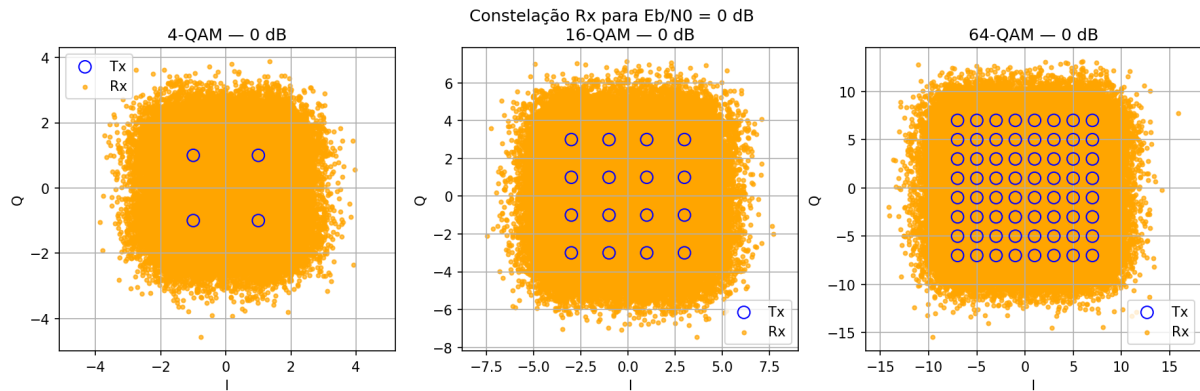


Figure 11: $E_b/N_0=0$ dB para M-QAM.

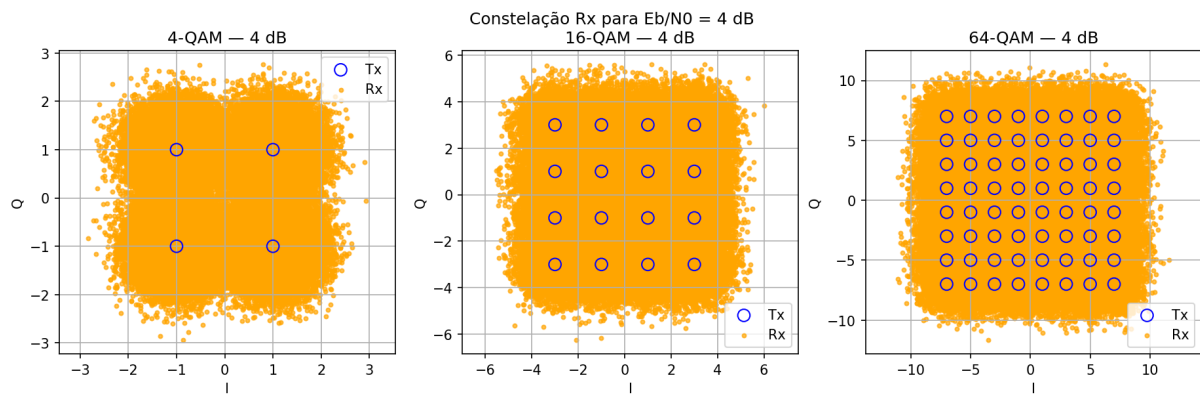


Figure 12: $E_b/N_0=4$ dB para M-QAM.

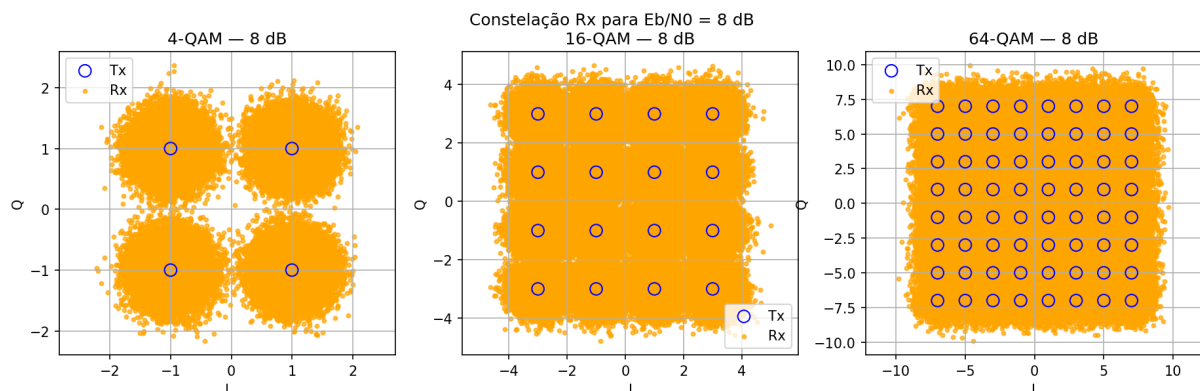


Figure 13: $E_b/N_0=8$ dB para M-QAM.

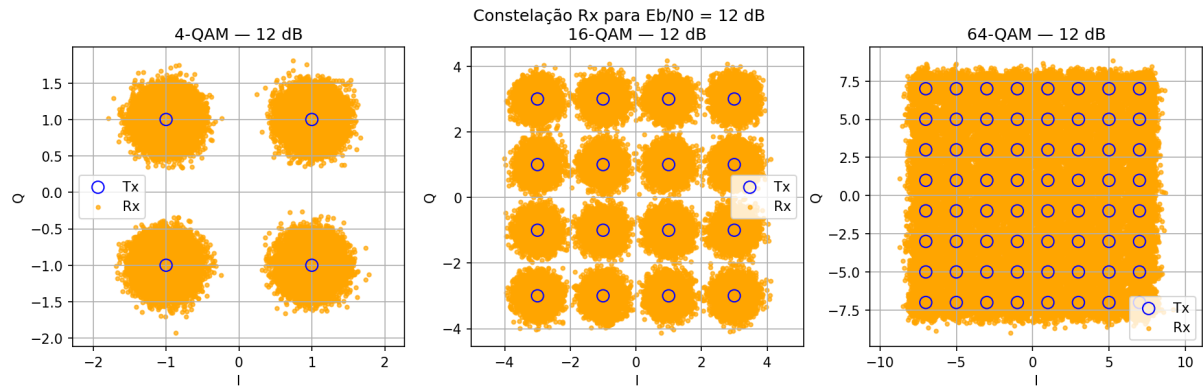


Figure 14: $E_b/N_0=12$ dB para M-QAM.

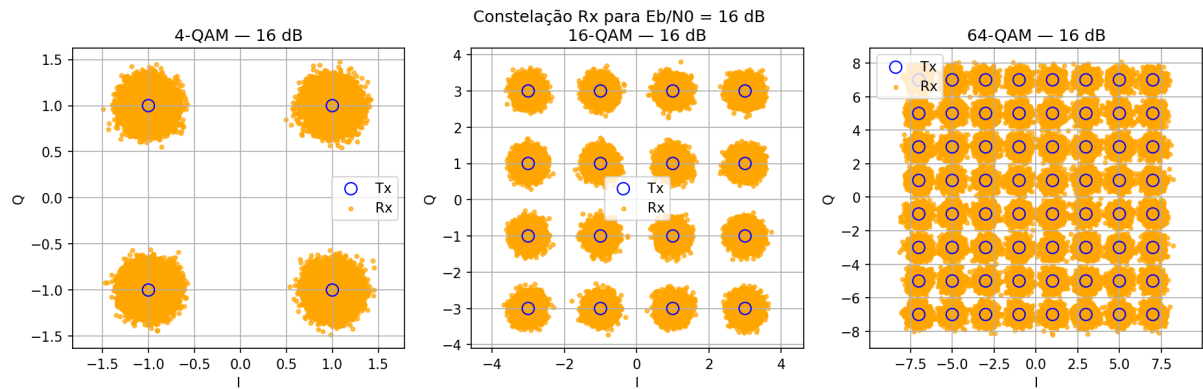


Figure 15: $E_b/N_0=16$ dB para M-QAM.

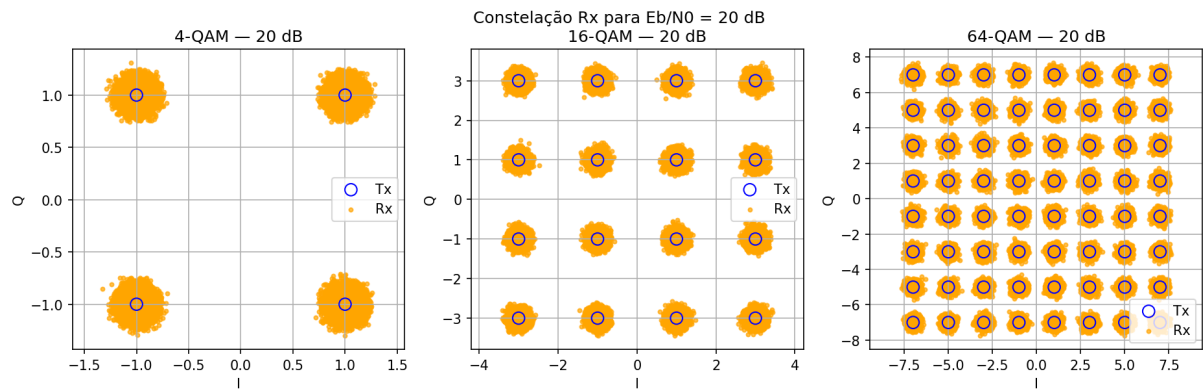


Figure 16: $E_b/N_0=20$ dB para M-QAM.

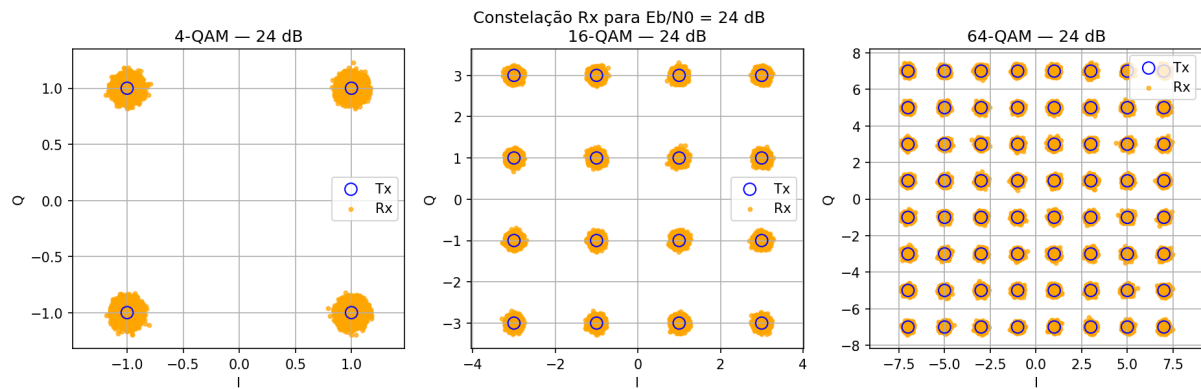


Figure 17: $E_b/N_0=24$ dB para M-QAM.

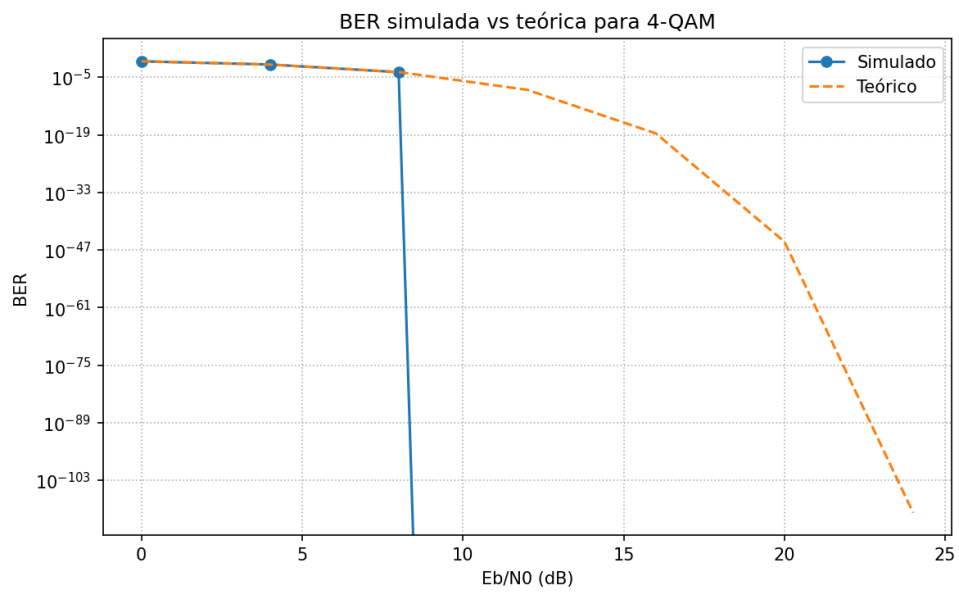


Figure 18: BER vs E_b/N_0 – 4-QAM.

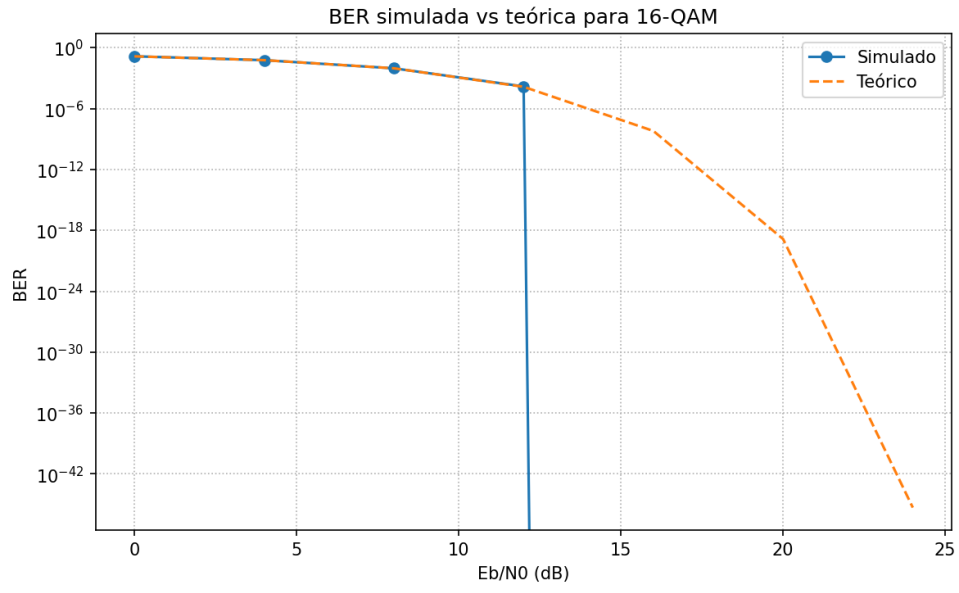


Figure 19: BER vs E_b/N_0 – 16-QAM.

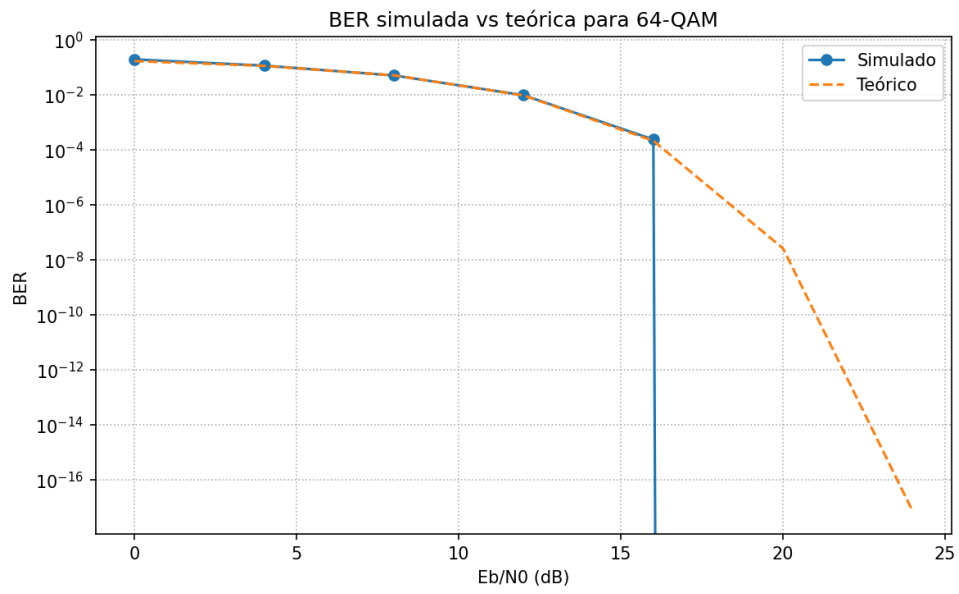


Figure 20: BER vs E_b/N_0 – 64-QAM.

M-PSK

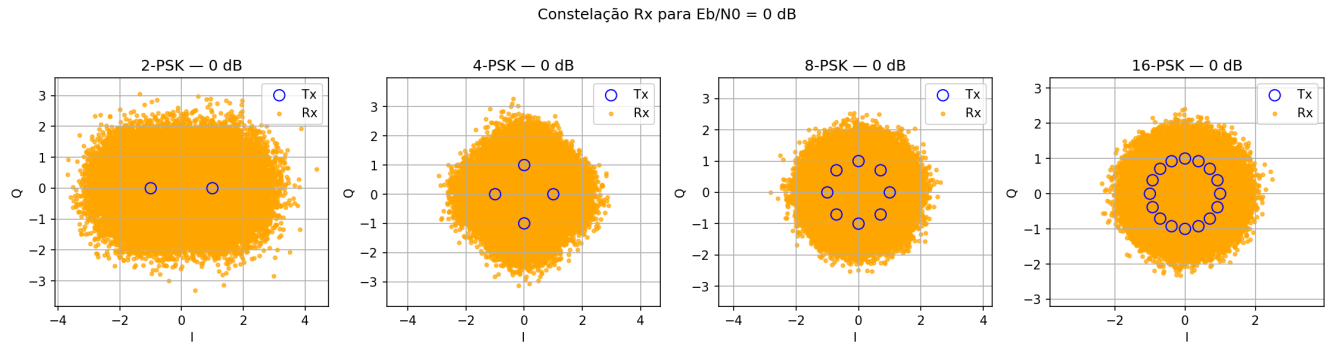


Figure 21: $E_b/N_0=0$ dB para M-PSK.

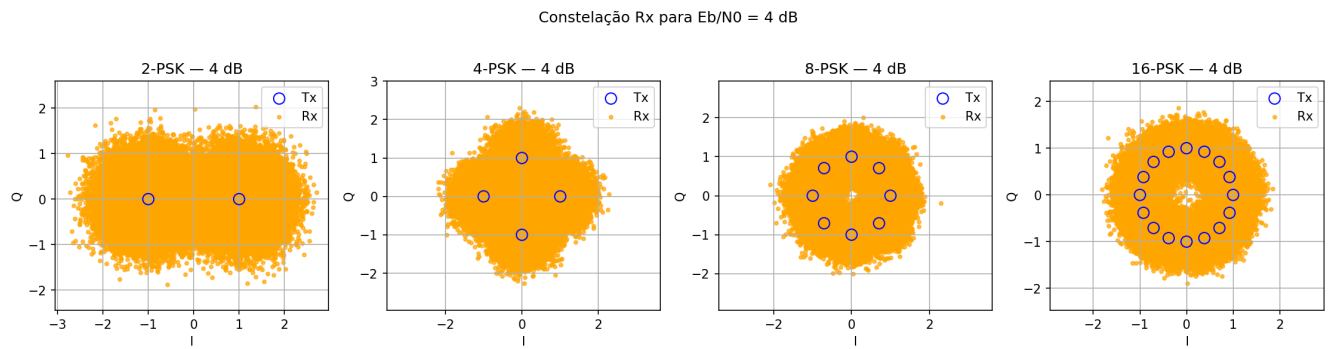


Figure 22: $E_b/N_0=4$ dB para M-PSK.

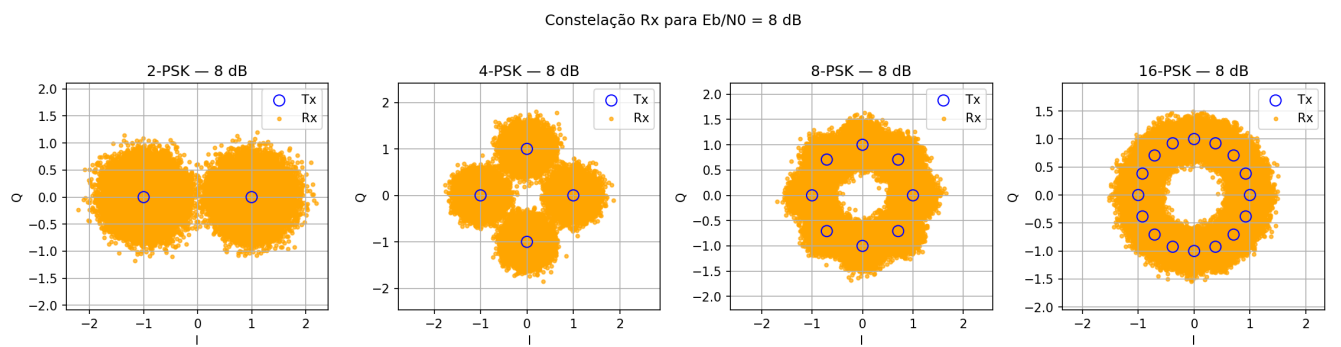


Figure 23: $E_b/N_0=8$ dB para M-PSK.

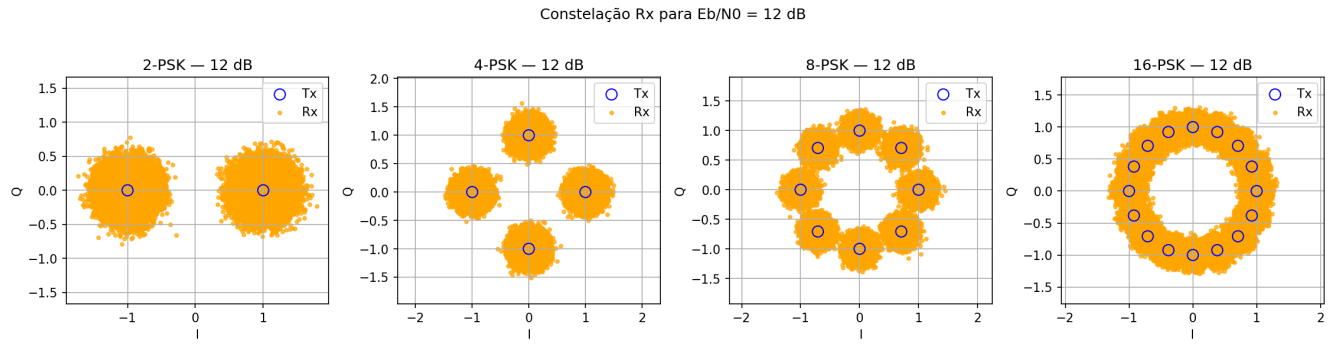


Figure 24: $E_b/N_0=12$ dB para M-PSK.

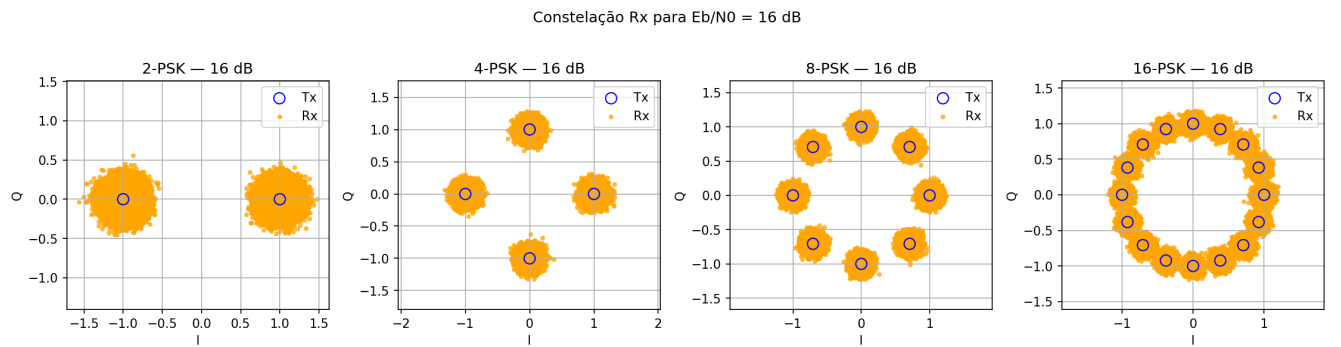


Figure 25: $E_b/N_0=16$ dB para M-PSK.

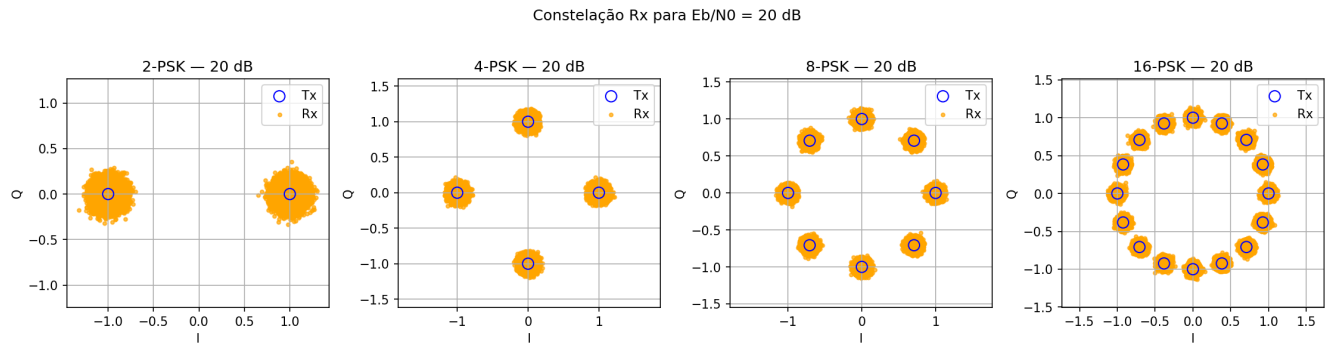


Figure 26: $E_b/N_0=20$ dB para M-PSK.

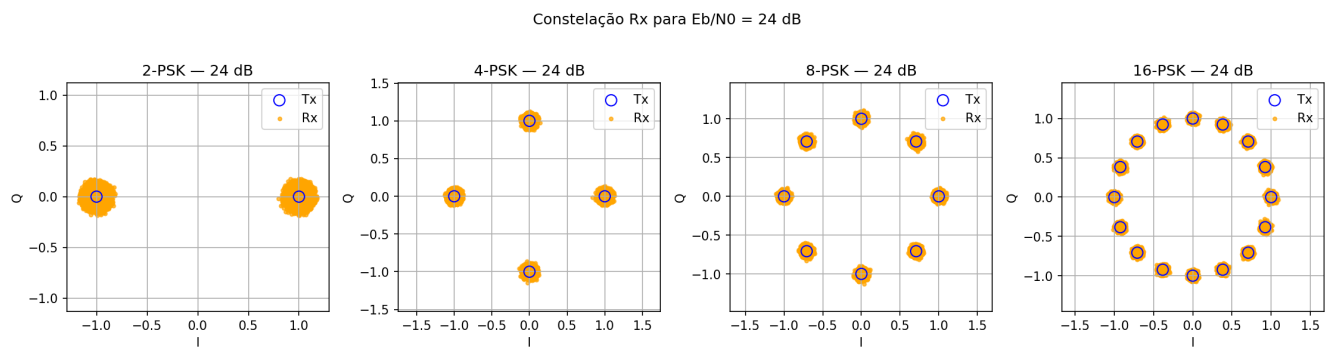


Figure 27: $E_b/N_0=24$ dB para M-PSK.

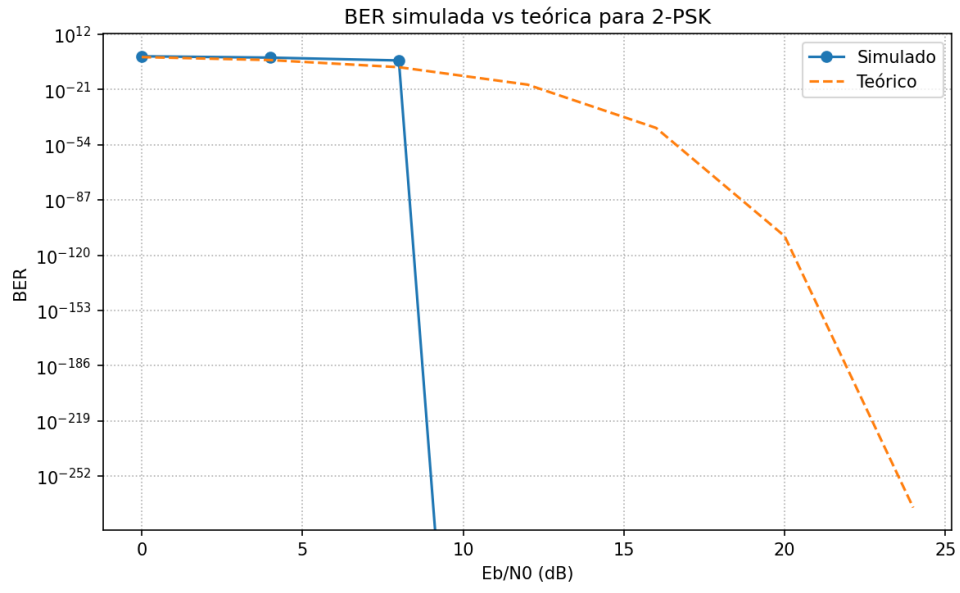


Figure 28: BER vs E_b/N_0 – 2-PSK.

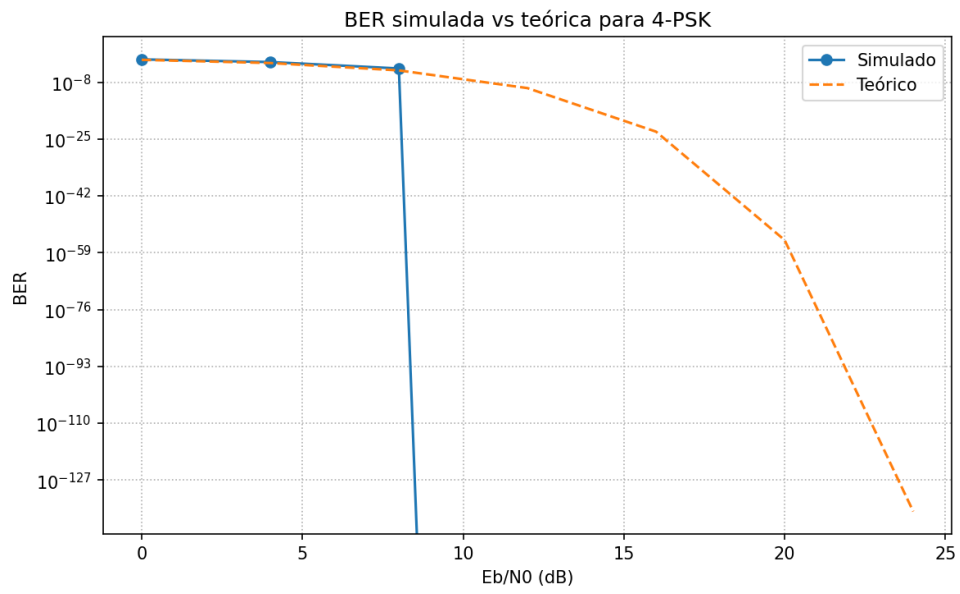


Figure 29: BER vs E_b/N_0 – 4-PSK.

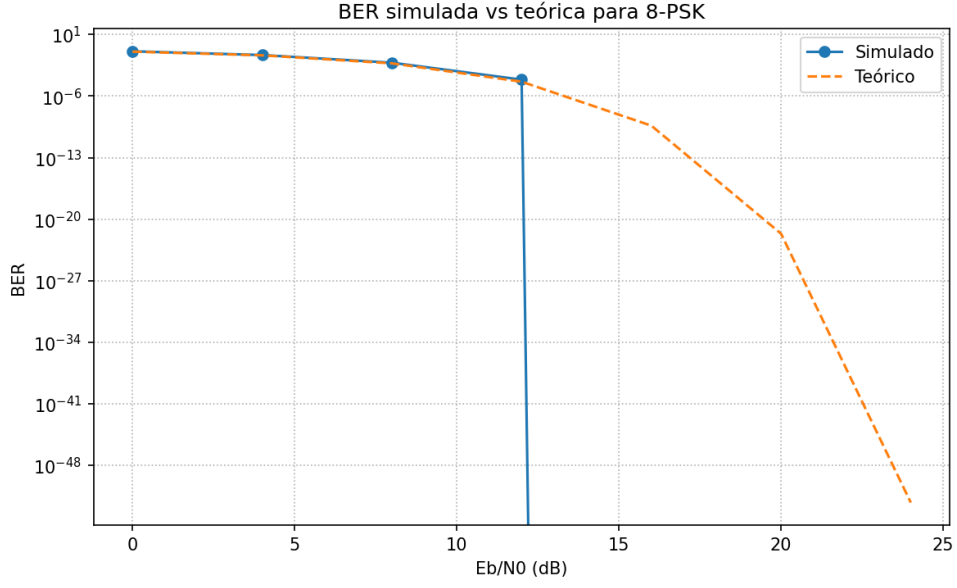


Figure 30: BER vs E_b/N_0 – 8-PSK.

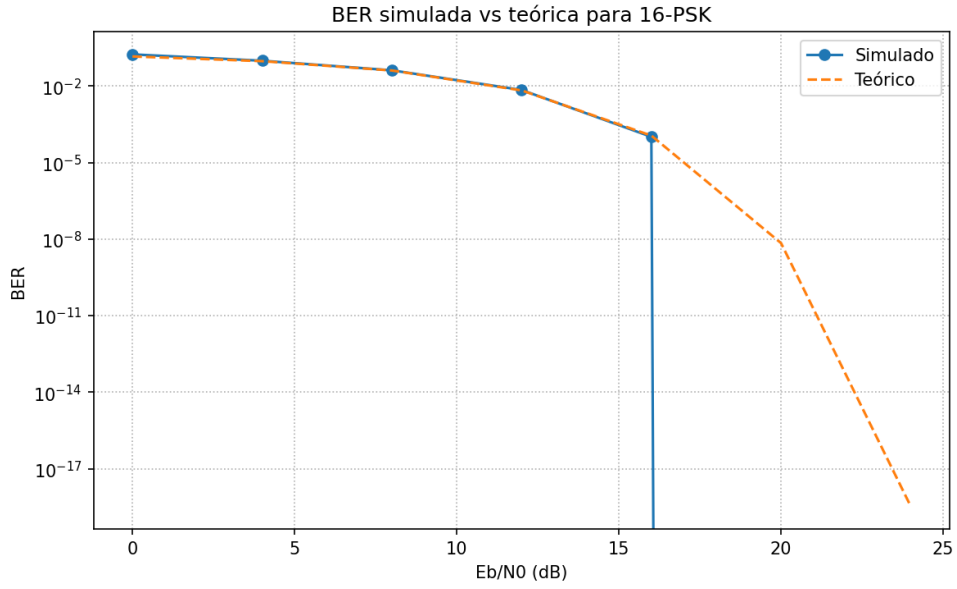


Figure 31: BER vs E_b/N_0 – 16-PSK.

Para M-QAM em canal AWGN com detector de máxima verossimilhança, a probabilidade de erro de símbolo P_e (SER) é dada por:

$$P_e = 4 \left(1 - \frac{1}{\sqrt{M}} \right) Q \left(\sqrt{\frac{3 \log_2 M}{M-1} \frac{E_b}{N_0}} \right)$$

Para converter de SER para BER (erro de bit), dividimos por $\log_2 M$:

$$\text{BER} \approx \frac{P_e}{\log_2 M} = \frac{4}{\log_2 M} \left(1 - \frac{1}{\sqrt{M}} \right) Q \left(\sqrt{\frac{3 \log_2 M}{M-1} \frac{E_b}{N_0}} \right).$$

Para M-PSK coerente em canal AWGN, usando a aproximação válida para $E_b/N_0 \gg 1$ e $M > 2$, a probabilidade de erro de símbolo é

$$P_e \approx 2Q\left(\sqrt{\frac{2\pi^2 \log_2 M}{M^2} \frac{E_b}{N_0}}\right)$$

e a BER resultante é:

$$\text{BER} \approx \frac{P_e}{\log_2 M} = \frac{2}{\log_2 M} Q\left(\sqrt{\frac{2\pi^2 \log_2 M}{M^2} \frac{E_b}{N_0}}\right).$$

Observou-se que, assim como em M -PAM, as constelações em M -QAM e nos esquemas M -PSK exibem grande dispersão em níveis baixos de E_b/N_0 , passando por um afinamento progressivo dos “grupos” de símbolos à medida que o E_b/N_0 aumenta.

Quando a ordem de modulação M aumenta e mantemos a mesma relação E_b/N_0 , os pontos da constelação ficam mais próximos uns dos outros. Consequentemente, qualquer ruído ou distorção faz com que os símbolos recebidos se misturem com maior facilidade, aumentando a probabilidade de erro na decisão.

As curvas de BER obtidas na Questão 2 seguem exatamente as mesmas tendências e conclusões da Questão 1: o BER simulado aproxima-se do valor teórico para E_b/N_0 e para ordens de modulação maiores o erro cresce mais rapidamente.

Códigos Utilizados

A seguir, apresentam-se os códigos em Python utilizados no relatório.

Questão 1

```
import numpy as np
import matplotlib
matplotlib.use('Agg')          # Usa backend 'Agg' para plotagem sem
                                janelas
import matplotlib.pyplot as plt
from scipy.special import erfc

# 1) Parametros de simulacao
b_vals      = [1, 2, 4]          # Bits por simbolo (1->2-
                                PAM, 2->4-PAM, 4->16-PAM)
alpha       = 0.15              # Roll-off do filtro RRC
sps         = 8                 # Amostras por simbolo
span        = 40                # Suporte do filtro em
                                simbolos (+-40 simbolos)
EbN0_dB     = np.arange(0, 25, 4) # Eb/NO em dB: [0, 4, 8,
                                ..., 24]
num_bits    = 1_000_000         # Total de bits para
                                estimar o BER

# 2) Gera os coeficientes do filtro RRC (p) e do matched filter (
                                q = p invertido)
```

```

def design_rrc(alpha, span, sps):
    N = span * sps + 1                                # Numero de taps: suporte
                                                         total x amostras + 1
    t = np.linspace(-span/2, span/2, N)                # Eixo do tempo de -span
                                                         /2 a +span/2
    h = np.zeros_like(t)
    for i, ti in enumerate(t):
        if np.isclose(ti, 0):
            # formula do RRC no tempo zero
            h[i] = 1 + alpha*(4/np.pi - 1)
        elif np.isclose(abs(ti), 1/(4*alpha)):
            # formula em ti = +-1/(4alpha)
            h[i] = (alpha/np.sqrt(2))*(
                (1 + 2/np.pi)*np.sin(np.pi/(4*alpha)) +
                (1 - 2/np.pi)*np.cos(np.pi/(4*alpha))
            )
        else:
            # expressao geral do pulso RRC
            num = np.sin(np.pi*ti*(1-alpha)) + 4*alpha*ti*np.cos(
                np.pi*ti*(1+alpha))
            den = np.pi*ti*(1 - (4*alpha*ti)**2)
            h[i] = num/den
    h /= np.linalg.norm(h)                                # Normaliza a energia do
                                                         filtro para 1
    return h, h[::-1]                                    # Retorna p e q (espelhado
                                                         )

p, q = design_rrc(alpha, span, sps)
delay = len(p) - 1                                    # Atraso total do par de
                                                         filtros

# 3) Mapeia bits -> simbolos Gray-codificados para M-PAM
def bits_to_symbols(bits, b):
    syms = bits.reshape(-1, b)                        # Agrupa
                                                         bits de b em b
    ints = syms.dot(1 << np.arange(b)[::-1])          # Converte
                                                         grupo em inteiro
    gray = ints ^ (ints >> 1)                          # Converte
                                                         para codigo Gray
    M = 2**b
    return 2*gray - (M - 1)                            # Mapeia
                                                         para {-M+1, ..., +M-1}

# 4) Transmissao: upsampling + convolucao com p(t)
def transmit(a, p, sps):
    ups = np.zeros(len(a)*sps)                        # Vetor de zeros numa taxa
                                                         maior
    ups[::sps] = a                                     # Insere cada simbolo a em um
                                                         slot
    return np.convolve(ups, p, mode='full')            # Pulse-shaping

```

```

# 5) Canal AWGN: adiciona ruido gaussiano
def awgn(x, Eb, EbN0_dB):
    NO = Eb / (10**(EbN0_dB/10))          # Densidade espectral
        de ruido
    sigma = np.sqrt(NO/2)                  # Desvio-padrão por
        amostra
    return x + sigma*np.random.randn(len(x))

# 6) Recepção: matched filter q(t) + downsampling
def receive(y, q, sps, delay):
    z = np.convolve(y, q, mode='full')      # Filtra com matched
        filter
    return z[delay::sps]                   # Compensa atraso e
        amostra por simbolo

# 7) Decisão e demapeamento Gray->bits
def detect(y_samp, M, b):
    consts = np.arange(-M+1, M, 2)          # Constelação teórica
    idxs = np.argmin(np.abs(y_samp[:,None] - consts[None,:]),
        axis=1)
    decisions = consts[idxs]                # Símbolos detectados
    gray_hat = (decisions + (M-1))/2        # Converte de volta
        pra Gray code
    binary = gray_hat.copy()
    shift = gray_hat >> 1
    # Reverte Gray->binário
    while np.any(shift):
        binary ^= shift
        shift >>= 1
    return ((binary[:,None] >> np.arange(b)[::-1]) & 1).flatten()

# 8) BER teórico para Gray-M-PAM
def ber_theoretical(EbN0_dB, M, b):
    arg = np.sqrt(3*b/(M**2-1) * 10**(EbN0_dB/10))
    return (2*(M-1)/(M*b)) * 0.5 * erfc(arg)

# 9a) Gera e salva diagramas de constelação para cada Eb/N0
for EbN0 in EbN0_dB:
    fig, axes = plt.subplots(1, len(b_vals), figsize=(12, 3))
    for ax, b in zip(axes, b_vals):
        M = 2**b
        bits = np.random.randint(0, 2, num_bits)
        a = bits_to_symbols(bits, b)
        x = transmit(a, p, sps)
        Eb = np.mean(a**2)/b
        y = awgn(x, Eb, EbN0)
        y_s = receive(y, q, sps, delay)[:1000] # so plotar 1000
            pontos
        ax.scatter(a[:1000], y_s, s=2)
        ax.set_title(f'{M}-PAM @ Eb/N0={EbN0} dB')
        ax.set_xlabel('Tx'); ax.set_ylabel('Rx'); ax.grid(True)

```



```

fig.tight_layout()
fig.savefig(f'constellation_{EbN0}dB.png', dpi=150,
            bbox_inches='tight')
plt.close(fig)

# 9b) Gera e salva curvas de BER vs Eb/NO para cada M-PAM
for b in b_vals:
    M = 2**b
    ber_sim = []
    ber_theo = []
    for EbN0 in EbN0_dB:
        bits = np.random.randint(0, 2, num_bits)
        a = bits_to_symbols(bits, b)
        x = transmit(a, p, sps)
        Eb = np.mean(a**2)/b
        y = awgn(x, Eb, EbN0)
        y_s = receive(y, q, sps, delay)[:len(a)]
        ber_sim.append(np.mean(bits != detect(y_s, M, b)))
        ber_theo.append(ber_theoretical(EbN0, M, b))
fig, ax = plt.subplots(figsize=(6, 4))
ax.semilogy(EbN0_dB, ber_sim, 'o-', label='Simulado')
ax.semilogy(EbN0_dB, ber_theo, '--', label='Teorico')
ax.set_title(f'BER vs Eb/NO ({M}-PAM)')
ax.set_xlabel('Eb/NO (dB)')
ax.set_ylabel('BER')
ax.grid(True, which='both')
ax.legend()
fig.tight_layout()
fig.savefig(f'ber_{M}PAM.png', dpi=150, bbox_inches='tight')
plt.close(fig)

```

Questão 2

M-QAM

```

import numpy as np
import matplotlib.pyplot as plt
from commpy.modulation import QAMModem
from scipy.special import erfc

# -- Parametros gerais -----
alpha = 0.15
T = 1e-3
span = 40
fc = 100e6
Fs = 4 * fc
upsample_rate = 16
num_symbols = 1000_00
EbN0_dB_range = np.arange(0, 25, 4)

```

```

# -- Gera vetor de tempo e pulso RRC
-----
# 2) Gera os coeficientes do filtro RRC (p) e do matched filter (
    q = p invertido)
def design_rrc(alpha, span, sps):
    N = span * sps + 1                # Numero de taps: suporte
        total x amostras + 1
    t = np.linspace(-span/2, span/2, N) # Eixo do tempo de -span
        /2 a +span/2
    h = np.zeros_like(t)
    for i, ti in enumerate(t):
        if np.isclose(ti, 0):
            # formula do RRC no tempo zero
            h[i] = 1 + alpha*(4/np.pi - 1)
        elif np.isclose(abs(ti), 1/(4*alpha)):
            # formula em ti = +-1/(4*alpha)
            h[i] = (alpha/np.sqrt(2))*(
                (1 + 2/np.pi)*np.sin(np.pi/(4*alpha)) +
                (1 - 2/np.pi)*np.cos(np.pi/(4*alpha))
            )
        else:
            # expressao geral do pulso RRC
            num = np.sin(np.pi*ti*(1-alpha)) + 4*alpha*ti*np.cos(
                np.pi*ti*(1+alpha))
            den = np.pi*ti*(1 - (4*alpha*ti)**2)
            h[i] = num/den
    h /= np.linalg.norm(h)            # Normaliza a energia do
        filtro para 1
    return h, h[::-1]                # Retorna p e q (espelhado
        )

p, q = design_rrc(alpha, span, upsample_rate)
delay = len(p) - 1                  # Atraso total do par de
    filtros
N = len(p)
t = np.linspace(-span/2, span/2, N)

# -- Plota o pulso RRC -----
plt.figure(figsize=(8,3))
plt.plot(t, p, linewidth=1.5)
plt.title(f'Pulso RRC (alpha={alpha}, T={T*1e3:.1f} ms, span={
    span})')
plt.xlabel('Tempo (s)'); plt.ylabel('Amplitude')
plt.grid(True); plt.tight_layout()
plt.show()

# -- Funcoes de up-sampling -----
def upsample(x, L):
    y = np.zeros(len(x) * L, dtype=x.dtype)
    y[::L] = x
    return y

```

```

# -- Simulacao para cada b = 2,4,6
-----
b_values = [2, 4, 6]
received = {}
tx_constellation = {}
rx_constellation = {EbNO_dB: {} for EbNO_dB in EbNO_dB_range}

for b in b_values:
    M = 2**b

    #Modulacao M-QAM
    modem = QAMModem(M)

    ber_sim = []
    ber_th = []
    # guarda constelacao pura
    tx_constellation[b] = modem.constellation.copy()

    for EbNO_dB in EbNO_dB_range:
        #geracao da entrada com bits aleatorios
        bits = np.random.randint(0, 2, b * num_symbols)

        #modula essa entrada com M-QAM
        symbols_tx = modem.modulate(bits)
        I_seq, Q_seq = symbols_tx.real, symbols_tx.imag

        #upsampling
        I_up = upsample(I_seq, upsample_rate)
        Q_up = upsample(Q_seq, upsample_rate)

        #convolucao com p(t)
        I_t = np.convolve(I_up, p, mode='full')
        Q_t = np.convolve(Q_up, p, mode='full')

        #multiplicacao por cos e sen (portadora)
        t_sig = np.arange(len(I_t)) / Fs
        carrier_cos = np.sqrt(2)*np.cos(2*np.pi*fc*t_sig)
        carrier_sin = np.sqrt(2)*np.sin(2*np.pi*fc*t_sig)
        x_pass = I_t*carrier_cos - Q_t*carrier_sin

        # ruído AWGN adicionado ao sinal X
        Ex = np.mean(np.abs(symbols_tx)**2)
        Eb = Ex / b
        EbNO = 10**((EbNO_dB / 10))
        sigma = np.sqrt((Eb / EbNO) / 2)
        noise = sigma * np.random.randn(len(x_pass))
        x_noisy = x_pass + noise

        #multiplicacao por cos e sen (portadora) do sinal X +
        # ruído AWGN

```

```

I_rx = x_noisy * carrier_cos
Q_rx = x_noisy * (-carrier_sin)

#convolucao com filtro casado - p(T-t)
I_mf = np.convolve(I_rx, p[::-1], mode='full')
Q_mf = np.convolve(Q_rx, p[::-1], mode='full')

#downsampling e amostragem
overall_delay = delay
YI_k = I_mf[overall_delay::upsample_rate][:num_symbols]
YQ_k = Q_mf[overall_delay::upsample_rate][:num_symbols]

# sinal recebido RX - Y com componentes I e Q
received[b] = (YI_k, YQ_k)
symbols_rx = YI_k + 1j * YQ_k

rx_constellation[EbN0_dB][b] = symbols_rx.copy()

#demodula o sinal recebido
bits_rx = modem.demodulate(symbols_rx, 'hard')
#compara com bit de entrada e bits recebido para poder
    formar o BER simulado
ber_sim.append(np.mean(bits_rx != bits))

#formula do BER teorico
k = np.log2(M)
ber_th.append(((4/k)*(1 - 1/np.sqrt(M))*0.5*erfc(np.sqrt
    ((3*k*EbN0)/(M-1))/np.sqrt(2))))

# grafico da BER vs Eb/NO dB
fig = plt.figure(figsize=(8, 5))
plt.semilogy(EbN0_dB_range, ber_sim, 'o-', label='Simulado')
plt.semilogy(EbN0_dB_range, ber_th, '--', label='Teorico')
plt.xlabel('Eb/NO (dB)')
plt.ylabel('BER')
plt.title(f'BER simulada vs teorica para {M}-QAM')
plt.grid(which='both', linestyle=':')
plt.legend()
plt.tight_layout()
fig.savefig(f'ber_{M}_QAM.png', dpi=150, bbox_inches='tight')
plt.close(fig)

# 2. Plote para cada Eb/NO uma figura com as tres modulacoes lado
a lado - CONSTELACOES
for EbN0_dB in EbN0_dB_range:
    fig, axes = plt.subplots(1, len(b_values), figsize=(15, 4))
    for ax, b in zip(axes, b_values):
        M = 2**b
        tx = tx_constellation[b]
        rx = rx_constellation[EbN0_dB][b]

```

```

ax.scatter(tx.real, tx.imag, s=80, ec='blue', facecolors=
    'none', label='Tx', zorder=2)
ax.scatter(rx.real, rx.imag, s=8, color='orange', label='
    Rx', zorder=1, alpha=0.7)
ax.set_title(f'{M}-QAM - {EbNO_dB} dB')
ax.set_xlabel('I')
ax.set_ylabel('Q')
ax.axis('equal')
ax.grid(True)
ax.legend()
plt.suptitle(f'Constelacao Rx para Eb/NO = {EbNO_dB} dB')
fig.savefig(f'constellation_EbNO_{EbNO_dB}dB_QAM.png', dpi
    =150, bbox_inches='tight')
plt.close(fig)

```

M-PSK

```

import numpy as np
import matplotlib.pyplot as plt
from commpy.modulation import PSKModem
from scipy.special import erfc

# -- Parametros gerais -----
alpha = 0.15
T = 1e-3
span = 40
fc = 100e6
Fs = 4 * fc
upsample_rate = 16
num_symbols = 1000_00
EbNO_dB_range = np.arange(0, 25, 4)

# -- Gera vetor de tempo e pulso RRC
-----
# 2) Gera os coeficientes do filtro RRC (p) e do matched filter (
    q = p invertido)
def design_rrc(alpha, span, sps):
    N = span * sps + 1                # Numero de taps: suporte
        total x amostras + 1
    t = np.linspace(-span/2, span/2, N) # Eixo do tempo de -span
        /2 a +span/2
    h = np.zeros_like(t)
    for i, ti in enumerate(t):
        if np.isclose(ti, 0):
            # formula do RRC no tempo zero
            h[i] = 1 + alpha*(4/np.pi - 1)
        elif np.isclose(abs(ti), 1/(4*alpha)):
            # formula em ti = +-1/(4*alpha)
            h[i] = (alpha/np.sqrt(2))*(
                (1 + 2/np.pi)*np.sin(np.pi/(4*alpha)) +
                (1 - 2/np.pi)*np.cos(np.pi/(4*alpha))
            )

```

```

    )
    else:
        # expressao geral do pulso RRC
        num = np.sin(np.pi*ti*(1-alpha)) + 4*alpha*ti*np.cos(
            np.pi*ti*(1+alpha))
        den = np.pi*ti*(1 - (4*alpha*ti)**2)
        h[i] = num/den
    h /= np.linalg.norm(h) # Normaliza a energia do
        filtro para 1
    return h, h[::-1] # Retorna p e q (espelhado
        )

p, q = design_rrc(alpha, span, upsample_rate)
delay = len(p) - 1 # Atraso total do par de
    filtros
N = len(p)
t = np.linspace(-span/2, span/2, N)

# -- Plota o pulso RRC -----
plt.figure(figsize=(8,3))
plt.plot(t, p, linewidth=1.5)
plt.title(f'Pulso RRC (alpha={alpha}, T={T*1e3:.1f} ms, span={
    span})')
plt.xlabel('Tempo (s)'); plt.ylabel('Amplitude')
plt.grid(True); plt.tight_layout()
plt.show()

# -- Funcoes de up-sampling -----
def upsample(x, L):
    y = np.zeros(len(x) * L, dtype=x.dtype)
    y[::L] = x
    return y

# -- Simulacao para cada b = 1,2,3,4
    -----
b_values = [1, 2, 3, 4]
received = {}
tx_constellation = {}
rx_constellation = {EbNO_dB: {} for EbNO_dB in EbNO_dB_range}

for b in b_values:
    M = 2**b

    #Modulacao M-PSK
    modem = PSKModem(M)

    ber_sim = []
    ber_th = []
    # guarda constelacao pura
    tx_constellation[b] = modem.constellation.copy()

```

```

for EbNO_dB in EbNO_dB_range:
    #geracao da entrada com bits aleatorios
    bits = np.random.randint(0, 2, b * num_symbols)

    #modula essa entrada com M-PSK
    symbols_tx = modem.modulate(bits)
    I_seq, Q_seq = symbols_tx.real, symbols_tx.imag

    #upsampling
    I_up = upsample(I_seq, upsample_rate)
    Q_up = upsample(Q_seq, upsample_rate)

    #convolucao com p(t)
    I_t = np.convolve(I_up, p, mode='full')
    Q_t = np.convolve(Q_up, p, mode='full')

    #multiplicacao por cos e sen (portadora)
    t_sig = np.arange(len(I_t)) / Fs
    carrier_cos = np.sqrt(2)*np.cos(2*np.pi*fc*t_sig)
    carrier_sin = np.sqrt(2)*np.sin(2*np.pi*fc*t_sig)
    x_pass = I_t*carrier_cos - Q_t*carrier_sin

    # ruído AWGN adicionado ao sinal X
    Ex = np.mean(np.abs(symbols_tx)**2)
    Eb = Ex / b
    EbNO = 10**(EbNO_dB / 10)
    sigma = np.sqrt((Eb / EbNO) / 2)
    noise = sigma * np.random.randn(len(x_pass))
    x_noisy = x_pass + noise

    #multiplicacao por cos e sen (portadora) do sinal X +
    # ruído AWGN
    I_rx = x_noisy * carrier_cos
    Q_rx = x_noisy * (-carrier_sin)

    #convolucao com filtro casado - p(T-t)
    I_mf = np.convolve(I_rx, p[::-1], mode='full')
    Q_mf = np.convolve(Q_rx, p[::-1], mode='full')

    #downsampling e amostragem
    overall_delay = delay
    YI_k = I_mf[overall_delay::upsample_rate][:num_symbols]
    YQ_k = Q_mf[overall_delay::upsample_rate][:num_symbols]

    # sinal recebido RX - Y com componentes I e Q
    received[b] = (YI_k, YQ_k)
    symbols_rx = YI_k + 1j * YQ_k

    rx_constellation[EbNO_dB][b] = symbols_rx.copy()

    #demodula o sinal recebido

```

```

bits_rx = modem.demodulate(symbols_rx, 'hard')
#compara com bit de entrada e bits recebido para poder
    formar o BER simulado
ber_sim.append(np.mean(bits_rx != bits))

#formula do BER teorico
EbNO = 10**((EbNO_dB/10))
# argumento da Q-function aproximada para simbolo
arg = np.sqrt(2 * np.pi**2 * EbNO * np.log2(M) / M**2)
#  $P_s \sim 2Q(arg) = \text{erfc}(arg/\sqrt{2})$ 
Ps = erfc(arg/np.sqrt(2))
# BER de bit aproximado (Gray coding)
Pb = Ps / np.log2(M)
ber_th.append(Pb)

# grafico da BER vs Eb/NO dB
fig = plt.figure(figsize=(8, 5))
plt.semilogy(EbNO_dB_range, ber_sim, 'o-', label='Simulado')
plt.semilogy(EbNO_dB_range, ber_th, '--', label='Teorico')
plt.xlabel('Eb/NO (dB)')
plt.ylabel('BER')
plt.title(f'BER simulada vs teorica para {M}-PSK')
plt.grid(which='both', linestyle=':')
plt.legend()
plt.tight_layout()
fig.savefig(f'ber_{M}_PSK.png', dpi=150, bbox_inches='tight')
plt.close(fig)

# 2. Plote para cada Eb/NO uma figura com as tres modulacoes lado
a lado - CONSTELACOES
for EbNO_dB in EbNO_dB_range:
    fig, axes = plt.subplots(1, len(b_values), figsize=(15, 4))
    for ax, b in zip(axes, b_values):
        M = 2**b
        tx = tx_constellation[b]
        rx = rx_constellation[EbNO_dB][b]
        ax.scatter(tx.real, tx.imag, s=80, ec='blue', facecolors=
            'none', label='Tx', zorder=2)
        ax.scatter(rx.real, rx.imag, s=8, color='orange', label='
            Rx', zorder=1, alpha=0.7)
        ax.set_title(f'{M}-PSK - {EbNO_dB} dB')
        ax.set_xlabel('I')
        ax.set_ylabel('Q')
        ax.axis('equal')
        ax.grid(True)
        ax.legend()
    plt.suptitle(f'Constelacao Rx para Eb/NO = {EbNO_dB} dB')
    plt.tight_layout(rect=[0, 0, 1, 0.96])
    fig.savefig(f'constellation_EbNO_{EbNO_dB}dB_PSK.png', dpi
        =150, bbox_inches='tight')
    plt.close(fig)

```