Redes de Computadores:
Internetworking, Roteamento e Transmissão

UNIVERSIDADE DO VALE DO RIO DOS SINOS

Sistema completo de transmissão digital

Apresentado por:

Vítor Pires

Roteiro:

- Objetivo do Trabalho
- Ferramentas utilizidas e demais definições
- Criação da mensagem em ASCII e conversão em dados binários
- Codificação de canal utilizando Manchester
- Implementação das técnicas de modução digital BPSK e QPSK.
- Adição de ruído AWGN
- Demodulação e Diagrama de Constelação
- Testes
- Conclusão

Objetivo do Trabalho

Aplicar na prática os conceitos de Redes de Computadores, desenvolvendo uma simulação completa de um enlace digital. O projeto foca na implementação da codificação de linha Manchester e das modulações BPSK e QPSK, submetendo o sinal a um canal com ruído AWGN. A meta final é gerar e analisar gráficos de desempenho (BER vs SNR).

Ambiente, ferramentas e definições

Ambiente de desenvolvimento: Google Colab

Linguagem de programação: Python

Codificador de canal: Manchester

Modulação Digital: BPSK e QPSK
Ruído: AWGN

Mensagem em ASCII e conversão em binários

```python
import numpy as np

def texto_em_bits(texto):
    bits = []
    for char in texto:
        ascii_val = ord(char)  # valor ASCII do caractere
        bin_str = format(ascii_val, '08b')  # binário com 8 bits
        bits.extend([int(b) for b in bin_str])
    return np.array(bits)

def bits_em_texto(bits):
    chars = []
    for i in range(0, len(bits), 8):
        byte = bits[i:i+8]
        ascii_val = int("".join(str(b) for b in byte), 2)
        chars.append(chr(ascii_val))
    return "".join(chars)


# Exemplo de uso
mensagem = "Unisinos"
bits = texto_em_bits(mensagem)

print("Mensagem original:", mensagem)
print("Bits gerados:")
print(bits)
print("\nReconstrução (teste):", bits_em_texto(bits))
```

```
Mensagem original: Unisinos
Bits gerados:
[0 1 0 1 0 1 0 1 0 1 1 0 1 1 1 0 0 1 1 0 1 0 0 1 0 1 1 1 0 0 1 1 0 1 1 0 1
 0 0 1 0 1 1 0 1 1 1 0 0 1 1 0 1 1 1 1 0 1 1 1 0 0 1 1]

Reconstrução (teste): Unisinos
```

Codificar de canal utilizando Manchester

```python
def codificar_manchester(bits):
    manchester = []
    for b in bits:
        if b == 0:
            manchester.extend([0, 1]) # Borda de subida (Transição de Baixo para Alto)
        else:
            manchester.extend([1, 0]) # Borda de descida (Transição de Alto para Baixo)
    return np.array(manchester)

# Teste
bits_codificados = codificar_manchester(bits)

print(f"Bits originais ({len(bits)}): {bits[:10]}...")
print(f"Bits Manchester ({len(bits_codificados)}): {bits_codificados[:20]}...")
```

```
Bits originais (64): [0 1 0 1 0 1 0 1 0 1]...
Bits Manchester (128): [0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0]...
```

Técnicas de modução digital BPSK e QPSK.

astype(complex) → Preparar o array para interagir com o ruído.

Sinal é dividido em dois componentes ortogonais.

Componente em fase, controlada pelo primeiro bit.

Componente em quadratura, controlada pelo segundo bit.

A normalização da energia ocorre para que o símbolo QPSK não transmita com mais energia do que o símbolo BPSK.

```python
def modular_bpsk(bits):
    # 2*bit - 1 faz: (2*0 - 1 = -1) e (2*1 - 1 = 1)
    return (2 * bits - 1).astype(complex)

def modular_qpsk(bits):
    simbolos = []
    # Garante um número par de bits (adiciona padding se necessário)
    if len(bits) % 2 != 0:
        bits = np.append(bits, 0)

    for i in range(0, len(bits), 2):
        b1 = bits[i]
        b2 = bits[i+1]

        # Mapeamento Real (I) e Imaginário (Q)
        # 0 -> -1, 1 -> +1
        real = -1 if b1 == 0 else 1
        imag = -1 if b2 == 0 else 1

        simbolos.append(real + 1j * imag)

    return np.array(simbolos) / np.sqrt(2) # Normalização de energia

# Teste
# Modular bits previamente codificados em Manchester
sinal_bpsk = modular_bpsk(bits_codificados)
sinal_qpsk = modular_qpsk(bits_codificados)

print(f"Símbolos BPSK ({len(sinal_bpsk)}): {sinal_bpsk[:5]}")
print(f"Símbolos QPSK ({len(sinal_qpsk)}): {sinal_qpsk[:5]}")
```

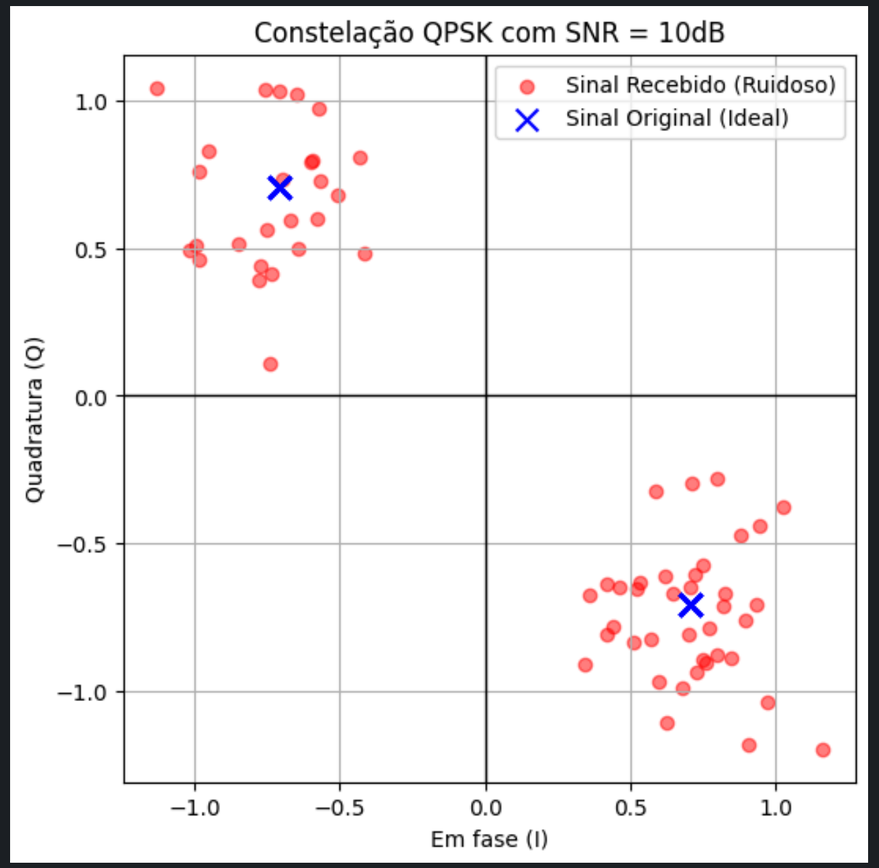
```
Símbolos BPSK (128): [-1.+0.j  1.+0.j  1.+0.j -1.+0.j -1.+0.j]
Símbolos QPSK (64): [-0.70710678+0.70710678j  0.70710678-0.70710678j -0.70710678+0.70710678j
  0.70710678-0.70710678j -0.70710678+0.70710678j]
```

Adição de ruído AWGN

```python
def adicionar_ruido(sinal, snr_db):
    # Converter dB para linear
    snr_linear = 10**(snr_db/10)

    # Calcular potência do sinal e do ruído necessário
    potencia_sinal = np.mean(np.abs(sinal)**2)
    potencia_ruido = potencia_sinal / snr_linear

    # Gerar ruído complexo (parte real + imaginária)
    # Dividimos por sqrt(2) porque a potência se divide entre real e imag
    ruido = np.sqrt(potencia_ruido/2) * (np.random.randn(len(sinal)) + 1j * np.random.randn(len(sinal)))

    return sinal + ruido
```

Demodulação e Plotagem do Diagrama

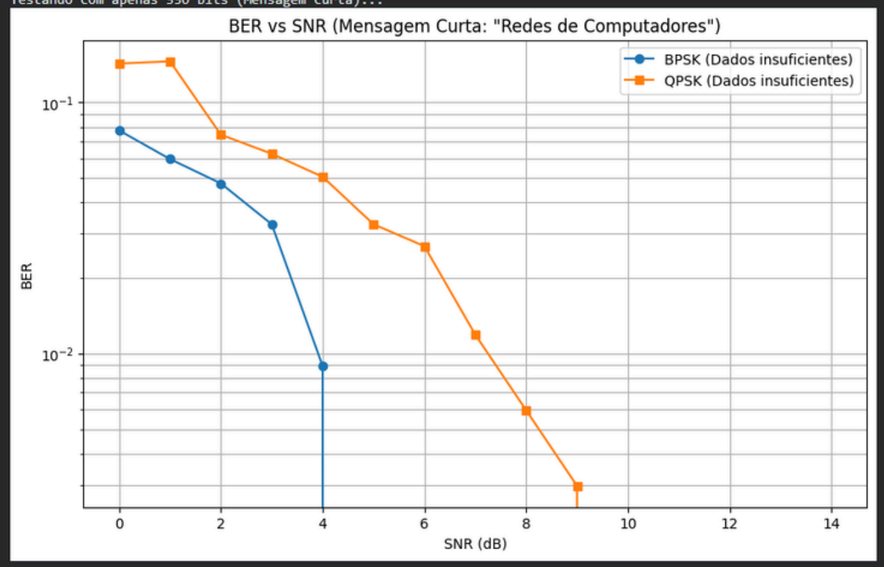
```python
def demodular_bpsk(sinal_recebido):
    return (sinal_recebido.real > 0).astype(int)

def demodular_qpsk(sinal_recebido):
    bits_recuperados = []
    # Desnormalizar
    sinal = sinal_recebido * np.sqrt(2)

    for simbolo in sinal:
        # Recupera 1º bit (Eixo Real)
        b1 = 1 if simbolo.real > 0 else 0
        # Recupera 2º bit (Eixo Imaginário)
        b2 = 1 if simbolo.imag > 0 else 0
        bits_recuperados.extend([b1, b2])

    return np.array(bits_recuperados)

# Uso de ruído de 10dB (moderado)
ruido_db = 10
sinal_recebido_qpsk = adicionar_ruido(sinal_qpsk, ruido_db)

# Plotar o Diagrama de Constelação (Scatter Plot)
import matplotlib.pyplot as plt
plt.figure(figsize=(6,6))
plt.scatter(sinal_recebido_qpsk.real, sinal_recebido_qpsk.imag, c='red', alpha=0.5, label='Sinal Recebido (Ruidoso)')
plt.scatter(sinal_qpsk.real, sinal_qpsk.imag, c='blue', marker='x', s=100, label='Sinal Original (Ideal)')
plt.title(f'Constelação QPSK com SNR = {ruido_db}dB')
plt.grid(True)
plt.legend()
plt.xlabel('Em fase (I)')
plt.ylabel('Quadratura (Q)')
plt.axhline(0, color='black', lw=1)
plt.axvline(0, color='black', lw=1)
plt.show()
```



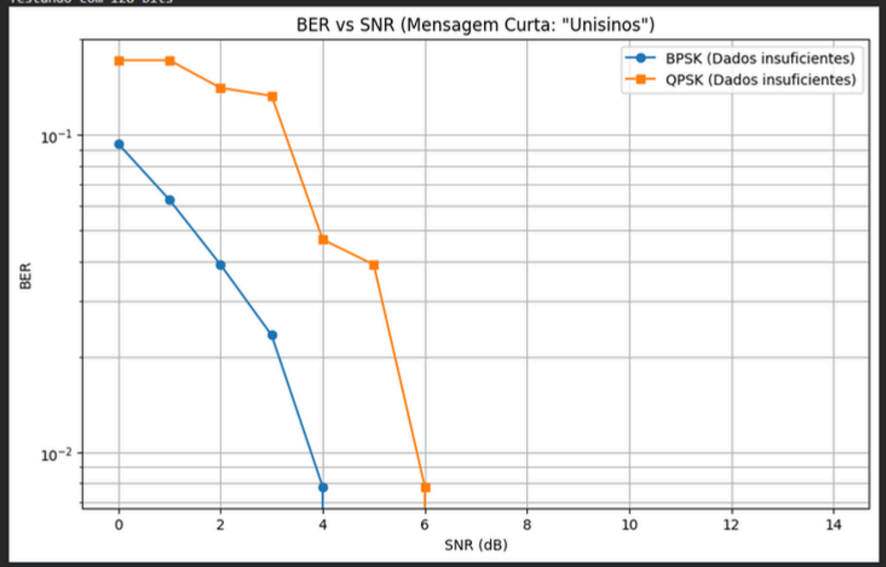
Testes com poucos bits

```python
print(f"Testando com {len(bits_codificados)} bits ")

snr_range = range(0, 15, 1) # Teste de 0 a 14 dB (de 1 em 1)
ber_bpsk = []
ber_qpsk = []

for snr in snr_range:
    # BPSK
    # Adição de ruído ao sinal
    rx_bpsk = adicionar_ruido(sinal_bpsk, snr)
    bits_rx_bpsk = demodular_bpsk(rx_bpsk)
    erros_bpsk = np.sum(bits_codificados != bits_rx_bpsk)
    ber_bpsk.append(erros_bpsk / len(bits_codificados))

    # QPSK
    rx_qpsk = adicionar_ruido(sinal_qpsk, snr)
    bits_rx_qpsk = demodular_qpsk(rx_qpsk)
    bits_rx_qpsk = bits_rx_qpsk[:len(bits_codificados)] # Garantia de tamanho
    erros_qpsk = np.sum(bits_codificados != bits_rx_qpsk)
    ber_qpsk.append(erros_qpsk / len(bits_codificados))

# Plotagem
plt.figure(figsize=(10, 6))
plt.semilogy(snr_range, ber_bpsk, 'o-', label='BPSK (Dados insuficientes)')
plt.semilogy(snr_range, ber_qpsk, 's-', label='QPSK (Dados insuficientes)')
plt.title('BER vs SNR (Mensagem Curta: "Unisinos")')
plt.xlabel('SNR (dB)')
plt.ylabel('BER')
plt.grid(True, which="both", ls="-")
plt.legend()
plt.show()
```

```
Testando com apenas 336 bits (Mensagem curta)...
```



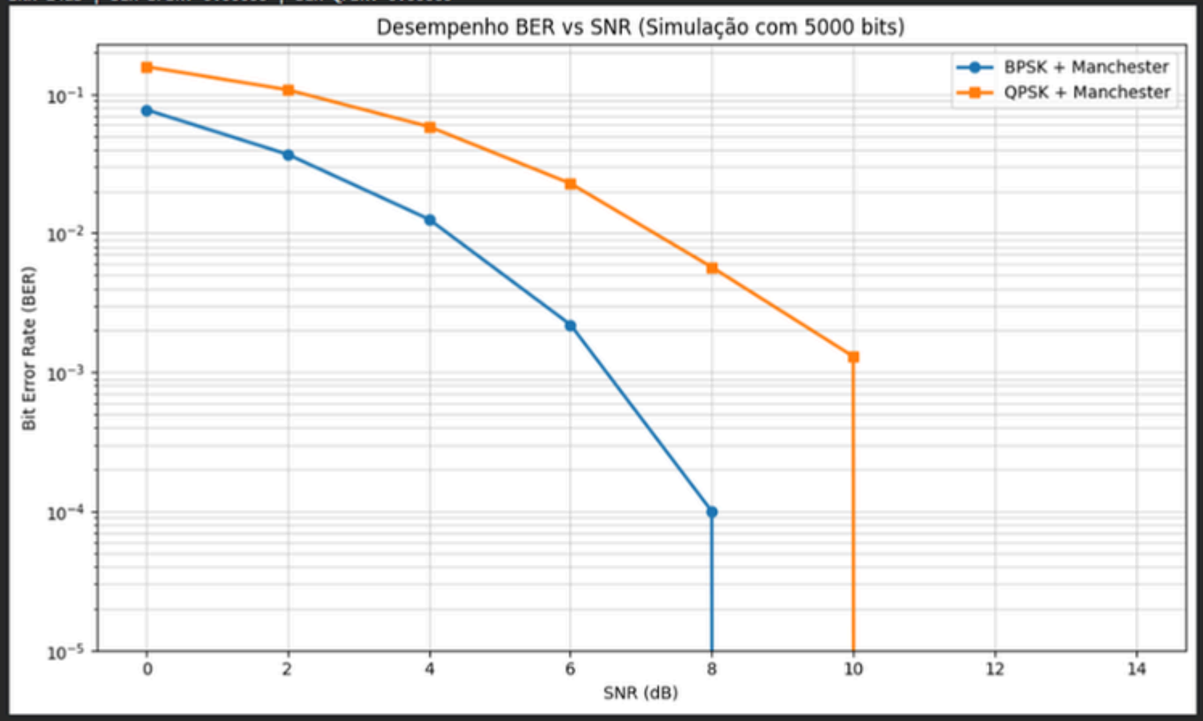

```
Testando com 128 bits
```



Testes com muitos bits (Aleatórios)

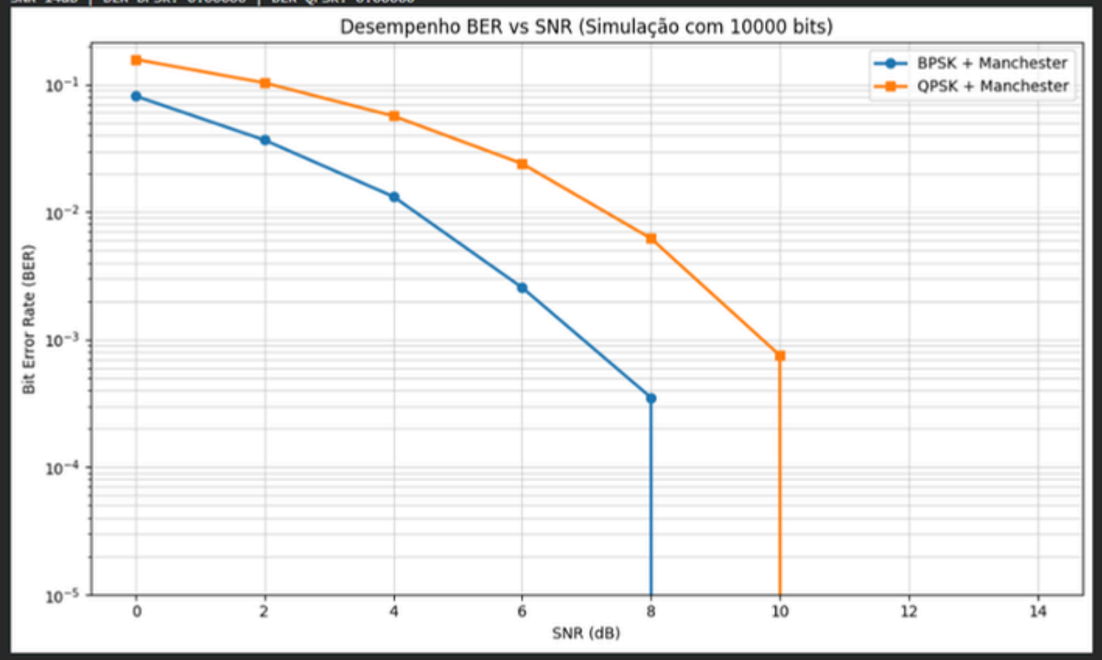
```python
N_BITS = 5000
print(f"Gerando {N_BITS} bits aleatórios para a simulação...")

# Geração dos Dados (Aleatórios)
bits_originais = np.random.randint(0, 2, N_BITS)

# Codificação de Canal (Manchester)
bits_cod = codificar_manchester(bits_originais)

# Modulação
tx_bpsk = modular_bpsk(bits_cod)
tx_qpsk = modular_qpsk(bits_cod)

# Loop de SNR
snr_range = range(0, 15, 2) # 0, 2, 4 ... 14 dB
ber_bpsk = []
ber_qpsk = []

print("Simulando... (Isso pode levar alguns segundos)")

for snr in snr_range:
    # BPSK
    rx_bpsk = adicionar_ruido(tx_bpsk, snr)
    bits_rx_bpsk = demodular_bpsk(rx_bpsk)
    erros_bpsk = np.sum(bits_cod != bits_rx_bpsk)
    ber_bpsk.append(erros_bpsk / len(bits_cod))

    # QPSK
    rx_qpsk = adicionar_ruido(tx_qpsk, snr)
    bits_rx_qpsk = demodular_qpsk(rx_qpsk)
    # Garante mesmo tamanho (caso padding tenha sido adicionado)
    bits_rx_qpsk = bits_rx_qpsk[:len(bits_cod)]
    erros_qpsk = np.sum(bits_cod != bits_rx_qpsk)
    ber_qpsk.append(erros_qpsk / len(bits_cod))

    print(f"SNR {snr:02d}dB | BER BPSK: {ber_bpsk[-1]:.5f} | BER QPSK: {ber_qpsk[-1]:.5f}")

# Plotagem do Gráfico
plt.figure(figsize=(10, 6))

# Uso de 'semilogy' para escala logarítmica no eixo Y
plt.semilogy(snr_range, ber_bpsk, 'o-', linewidth=2, label='BPSK + Manchester')
plt.semilogy(snr_range, ber_qpsk, 's-', linewidth=2, label='QPSK + Manchester')
```

```
Gerando 5000 bits aleatórios para a simulação...
Simulando... (Isso pode levar alguns segundos)
SNR 00dB | BER BPSK: 0.07730 | BER QPSK: 0.15760
SNR 02dB | BER BPSK: 0.03690 | BER QPSK: 0.10750
SNR 04dB | BER BPSK: 0.01260 | BER QPSK: 0.05840
SNR 06dB | BER BPSK: 0.00220 | BER QPSK: 0.02280
SNR 08dB | BER BPSK: 0.00010 | BER QPSK: 0.00570
SNR 10dB | BER BPSK: 0.00000 | BER QPSK: 0.00130
SNR 12dB | BER BPSK: 0.00000 | BER QPSK: 0.00000
SNR 14dB | BER BPSK: 0.00000 | BER QPSK: 0.00000
```

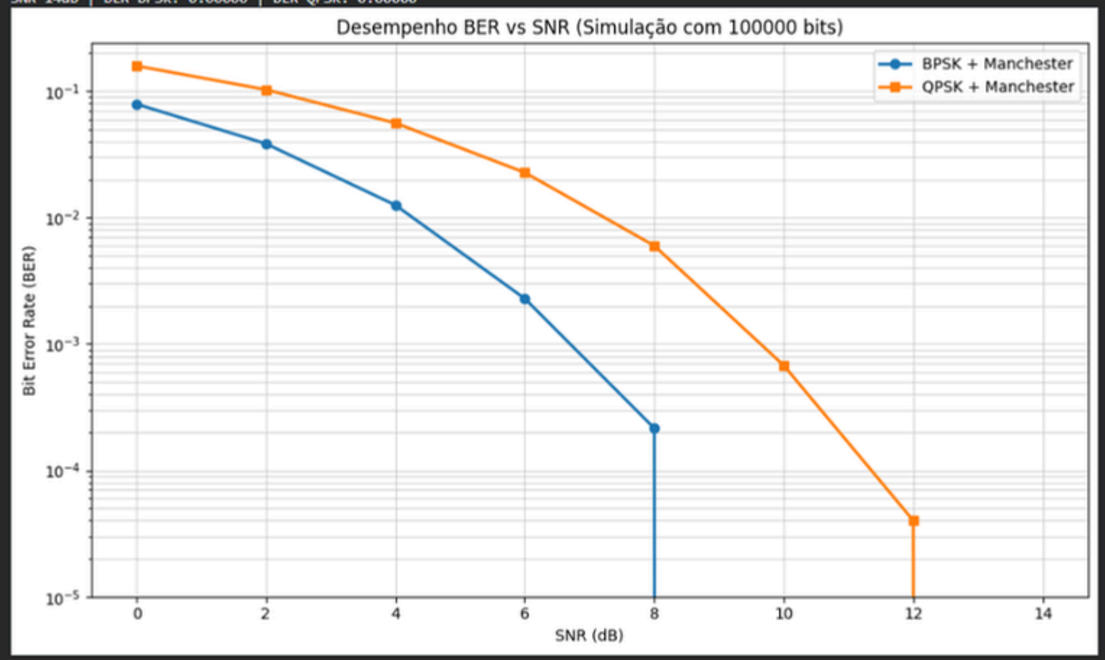


Testes com muitos bits (Aleatórios)

```
Gerando 10000 bits aleatórios para a simulação...
Simulando... (Isso pode levar alguns segundos)
SNR 00dB | BER BPSK: 0.08100 | BER QPSK: 0.15700
SNR 02dB | BER BPSK: 0.03660 | BER QPSK: 0.10290
SNR 04dB | BER BPSK: 0.01320 | BER QPSK: 0.05655
SNR 06dB | BER BPSK: 0.00255 | BER QPSK: 0.02390
SNR 08dB | BER BPSK: 0.00035 | BER QPSK: 0.00620
SNR 10dB | BER BPSK: 0.00000 | BER QPSK: 0.00075
SNR 12dB | BER BPSK: 0.00000 | BER QPSK: 0.00000
SNR 14dB | BER BPSK: 0.00000 | BER QPSK: 0.00000
```




```
Gerando 100000 bits aleatórios para a simulação final...
Simulando... (Isso pode levar alguns segundos)
SNR 00dB | BER BPSK: 0.07897 | BER QPSK: 0.15845
SNR 02dB | BER BPSK: 0.03839 | BER QPSK: 0.10281
SNR 04dB | BER BPSK: 0.01254 | BER QPSK: 0.05595
SNR 06dB | BER BPSK: 0.00228 | BER QPSK: 0.02268
SNR 08dB | BER BPSK: 0.00021 | BER QPSK: 0.00598
SNR 10dB | BER BPSK: 0.00000 | BER QPSK: 0.00068
SNR 12dB | BER BPSK: 0.00000 | BER QPSK: 0.00004
SNR 14dB | BER BPSK: 0.00000 | BER QPSK: 0.00000
```



Conclusão dos Testes

- Baixa quantidade de dados gerou curvas de BER "degrau", onde a taxa de erro caía abruptamente para zero em SNRs baixas (4dB a 6dB);

- aumentar a massa de dados para 100.000 bits aleatórios resultou em curvas suaves (Smooth), permitindo observar o comportamento do sistema até taxas de erro de 10^{-5}.

- robustez superior da modulação BPSK em comparação à QPSK

- Para atingir uma taxa de erro de 10^{-3} o BPSK necessitou de uma SNR de aproximadamente 6.5 dB;

- Para a mesma taxa de erro, o QPSK exigiu uma SNR de aproximadamente 9.5 dB;

- Como os símbolos do QPSK estão mais próximos no diagrama de constelação eles são mais suscetíveis à interferência do ruído AWGN;

- O uso da codificação Manchester garantiu sincronismo, mas reduziu a eficiência espectral pela metade;

- A modulação QPSK transmitiu o dobro de bits por símbolo que a BPSK, sendo espectralmente mais eficiente. No entanto, pagou o preço exigindo maior potência de sinal (SNR) para manter a integridade dos dados.