به نام خدا

گزارشکار پروژه اول مخابرات دیجیتال

استاد : دکتر محمودی

نویسنده : ارشیا مددی

گزارش کار شبیه‌سازی احتمال خطا (BER) برای مدولاسیون PAM با شکل‌دهی پالس Raised Cosine

مقدمه

در این گزارش کار، به شبیه‌سازی احتمال خطا (Bit Error Rate - BER) برای مدولاسیون چند سطحی (PAM) با استفاده از شکل‌دهی پالس Raised Cosine پرداخته می‌شود. شکل‌دهی پالس یکی از تکنیک‌های مهم در ارتباطات دیجیتال است که به کاهش تداخل بین نمادها و بهبود کیفیت سیگنال کمک می‌کند. در این شبیه‌سازی، از توابع ریاضی و کتابخانه‌های پایتون برای محاسبه و رسم منحنی‌های BER استفاده شده است.

توضیحات کد

1. وارد کردن کتابخانه‌ها

python

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.special import erfc
```

در ابتدا، کتابخانه‌های مورد نیاز برای محاسبات عددی (numpy)، رسم نمودار (matplotlib)، و محاسبه تابع مکمل خطای گوسی (scipy.special.erfc) وارد می‌شوند.

2. تعریف تابع Q

python

```
def Q(x):
```

return 0.5 * erfc(x / np.sqrt(2))

تابع Q برای محاسبه احتمال خطا در مدولاسیون PAM استفاده می‌شود. این تابع از تابع مکمل خطای گوسی استفاده می‌کند.

3. پارامترهای سیگنال

python

A = 1 # *دامنه سیگنال PAM*

RB = 1e3 # *نرخ داده (تعداد نمادها در ثانیه)*

T_symbol = 1 / RB # *مدت زمان هر نماد*

fs = 10 * RB # *نرخ نمونه‌برداری*

num_symbols = 10000 # *تعداد نمادها برای شبیه‌سازی*

در این بخش، پارامترهای اصلی سیگنال شامل دامنه، نرخ داده، مدت زمان هر نماد، نرخ نمونه‌برداری و تعداد نمادها برای شبیه‌سازی تعریف می‌شود.

4. پارامترهای پالس Raised Cosine

python

roll_off = 0.25 # *ضریب رول‌آف*

span = 4 # *تعداد نمادهای اثرگذار*

در این قسمت، ضریب رول‌آف و تعداد نمادهای اثرگذار برای پالس Raised Cosine مشخص می‌شود.

5. تولید پالس Raised Cosine

python

```
def raised_cosine_pulse(roll_off, span, T_symbol, fs):
    t = np.linspace(-span*T_symbol, span*T_symbol, int(2*span*T_symbol*fs))  # زمان
    h = np.sinc(t / T_symbol) * np.cos(np.pi * roll_off * t / T_symbol) / (1 - (2 * roll_off * t / T_symbol) ** 2)  # پالس
    h[np.isnan(h)] = 0  # رفع NaN# های احتمالی
    return h / np.max(np.abs(h))  # نرمال‌سازی به دامنه واحد
```

این تابع، پالس Raised Cosine را با استفاده از فرمول مشخص شده تولید می‌کند. زمان مناسب برای تولید پالس محاسبه و پالس نرمال‌سازی می‌شود.

6. تولید داده‌های تصادفی

python

```
bits = np.random.choice([0, 1], size=num_symbols)
symbols = 2*bits - 1  # تبدیل بیت‌ها به -1 و +1
```

در این بخش، داده‌های تصادفی (بیت‌ها) تولید می‌شوند و به نمادهای PAM تبدیل می‌شوند.

7. اعمال پالس Raised Cosine به داده‌های PAM

python

```
pulse = raised_cosine_pulse(roll_off, span, T_symbol, fs)
signal = np.convolve(symbols, pulse, mode='same')
```

در این قسمت، پالس Raised Cosine به داده‌های PAM اعمال می‌شود و سیگنال نهایی تولید می‌شود.

8. دامنه SNR (دسی‌بل)

python

```
SNR_dB_range = np.arange(0, 20, 1)
SNR_linear = 10**(SNR_dB_range / 10)
```

دامنه SNR از 0 تا 20 دسی‌بل تعریف می‌شود و به مقیاس خطی تبدیل می‌شود.

9. محاسبه احتمال خطا (BER) با استفاده از تابع Q

python

```
BER_theoretical = Q(np.sqrt(SNR_linear))
```

احتمال خطا نظری (BER) بر اساس SNR محاسبه می‌شود.

10. شبیه‌سازی BER

python

```
BER_simulated = []

for SNR_dB in SNR_dB_range:
    noise = np.random.normal(0, np.sqrt(1 / (2 * 10**(SNR_dB / 10))), size=signal.shape)
    received_signal = signal + noise
    detected_symbols = np.sign(received_signal)
    num_errors = np.sum(detected_symbols != symbols)
    BER_simulated.append(num_errors / num_symbols)
```

در این قسمت، برای هر مقدار SNR، نویز گوسی تولید می‌شود و به سیگنال دریافتی اضافه می‌شود. سپس نمادهای دریافتی شناسایی و احتمال خطا محاسبه می‌شود.

11. رسم منحنی BER

python

```
plt.figure(figsize=(8, 6))

plt.semilogy(SNR_dB_range, BER_theoretical, label="Theoretical BER", marker='o')

plt.semilogy(SNR_dB_range, BER_simulated, label="Simulated BER", marker='x')

plt.title("BER vs SNR for PAM with Raised Cosine Pulse (Theoretical vs Simulated)")

plt.xlabel("SNR (dB)")

plt.ylabel("BER")

plt.grid(True, which='both', linestyle='--', linewidth=0.5)

plt.legend()

plt.show()
```

در نهایت، منحنی‌های BER نظری و شبیه‌سازی شده بر اساس SNR رسم می‌شوند. از مقیاس لگاریتمی برای محور عمودی استفاده شده است تا تغییرات در احتمال خطا به وضوح نمایش داده شود.

نتیجه‌گیری

این شبیه‌سازی نشان می‌دهد که چطور شکل‌دهی پالس Raised Cosine می‌تواند بر احتمال خطا در مدولاسیون PAM تأثیر بگذارد. منحنی‌های BER نظری و شبیه‌سازی شده به خوبی همخوانی دارند و این نشان‌دهنده دقت روش‌های تحلیلی و شبیه‌سازی در ارزیابی عملکرد سیستم‌های مخابراتی است. این شبیه‌سازی می‌تواند به عنوان پایه‌ای برای تحلیل‌های پیشرفته‌تر در زمینه ارتباطات دیجیتال مورد استفاده قرار گیرد.

گزارش کار شبیه‌سازی احتمال خطا (BER) برای مدولاسیون PAM-4 با شکل‌دهی پالس Raised Cosine

مقدمه

در این گزارش کار، به شبیه‌سازی احتمال خطا (Bit Error Rate - BER) برای مدولاسیون PAM-4 با استفاده از شکل‌دهی پالس Raised Cosine پرداخته می‌شود. مدولاسیون PAM-4 به عنوان یک نوع مدولاسیون چند سطحی، به انتقال چهار سطح سیگنال (0، 1، 2 و 3) با استفاده از دامنه‌های مختلف کمک می‌کند. شکل‌دهی پالس Raised Cosine به کاهش تداخل بین نمادها و بهبود کیفیت سیگنال کمک می‌کند.

توضیحات کد

1. وارد کردن کتابخانه‌ها

python

```
import numpy as np

import matplotlib.pyplot as plt

from scipy.special import erfc
```

در ابتدا، کتابخانه‌های مورد نیاز برای محاسبات عددی (numpy)، رسم نمودار (matplotlib) و محاسبه تابع مکمل خطای گوسی (scipy.special.erfc) وارد می‌شوند.

2. تعریف تابع Q

python

```
def Q(x):
```

```
    return 0.5 * erfc(x / np.sqrt(2))
```

تابع Q برای محاسبه احتمال خطا در مدولاسیون PAM استفاده می‌شود. این تابع از تابع مکمل خطای گوسی برای محاسبه احتمال خطا استفاده می‌کند.

3. پارامترهای سیگنال

python

```
A = 1  #PAM دامنه پایه سیگنال
RB = 1e3  # نرخ داده (تعداد نمادها در ثانیه)
T_symbol = 1 / RB  # مدت زمان هر نماد
fs = 10 * RB  # نرخ نمونه‌برداری
num_symbols = 10000  # تعداد نمادها برای شبیه‌سازی
```

در این بخش، پارامترهای اصلی سیگنال شامل دامنه، نرخ داده، مدت زمان هر نماد، نرخ نمونه‌برداری و تعداد نمادها برای شبیه‌سازی تعریف می‌شود.

4. پارامترهای پالس Raised Cosine

python

```
roll_off = 0.25  # ضریب رول‌آف
span = 4  # تعداد نمادهای اثرگذار
```

در این قسمت، ضریب رول‌آف و تعداد نمادهای اثرگذار برای پالس Raised Cosine مشخص می‌شود.

5. تولید پالس Raised Cosine

python

```
def raised_cosine_pulse(roll_off, span, T_symbol, fs):
    t = np.linspace(-span * T_symbol, span * T_symbol, int(2 * span * T_symbol * fs))
    h = np.sinc(t / T_symbol) * np.cos(np.pi * roll_off * t / T_symbol) / (1 - (2 * roll_off * t / T_symbol) ** 2)
    h[np.isnan(h)] = 0
    return h / np.max(np.abs(h))
```

این تابع، پالس Raised Cosine را با استفاده از فرمول مشخص شده تولید می‌کند. زمان مناسب برای تولید پالس محاسبه و پالس نرمال‌سازی می‌شود.

6. تولید داده‌های تصادفی برای PAM-4

```
python
bits = np.random.choice([0, 1, 2, 3], size=num_symbols)
symbols = A * (2 * bits - 3) #تبدیل بیت‌ها به مقادیر PAM: A3-, A-, A, A3-4
```

در این بخش، داده‌های تصادفی (بیت‌ها) برای PAM-4 تولید می‌شوند و به مقادیر PAM تبدیل می‌شوند. مقادیر PAM-4 به صورت -A3، A-، A و A3 تعریف می‌شوند.

7. اعمال پالس Raised Cosine به داده‌های PAM

```
python
pulse = raised_cosine_pulse(roll_off, span, T_symbol, fs)
signal = np.convolve(symbols, pulse, mode='same')
```

در این قسمت، پالس Raised Cosine به داده‌های PAM اعمال می‌شود و سیگنال نهایی تولید می‌شود.

8. دامنه SNR (دسی‌بل)

python

```
SNR_dB_range = np.arange(0, 20, 1)
SNR_linear = 10**(SNR_dB_range / 10)
```

دامنه SNR از 0 تا 20 دسی‌بل تعریف می‌شود و به مقیاس خطی تبدیل می‌شود.

9. شبیه‌سازی BER

python

```
BER_simulated = []
thresholds = np.array([-2 * A, 0, 2 * A])  # مقادیر آستانه برای آشکارسازی 4-PAM

for SNR_dB in SNR_dB_range:
    noise = np.random.normal(0, np.sqrt(1 / (2 * 10**(SNR_dB / 10))), size=signal.shape)
    received_signal = signal + noise

    detected_symbols = np.zeros(received_signal.shape)
    detected_symbols[received_signal < thresholds[0]] = -3 * A  # A3-
    detected_symbols[(received_signal >= thresholds[0]) & (received_signal < thresholds[1])] = -A  # A-
```

detected_symbols[(received_signal >= thresholds[1]) & (received_signal < thresholds[2])] = A *# A*

detected_symbols[received_signal >= thresholds[2]] = 3 * A *# A3*

num_errors = np.sum(detected_symbols != symbols)

BER_simulated.append(num_errors / num_symbols)

در این قسمت، برای هر مقدار SNR، نویز گوسی تولید می‌شود و به سیگنال دریافتی اضافه می‌شود. سپس نمادهای دریافتی بر اساس آستانه‌های مشخص شده شناسایی می‌شوند و احتمال خطا محاسبه می‌شود.

10. محاسبه احتمال خطای نظری (BER)

python

BER_theoretical = (3 / 2) * Q(np.sqrt(SNR_linear / 5))

احتمال خطا نظری (BER) بر اساس SNR محاسبه می‌شود. فرمول مورد استفاده برای PAM-4 به گونه‌ای است که با توجه به تعداد سطوح سیگنال، احتمال خطا را محاسبه می‌کند.

11. رسم منحنی BER

python

plt.figure(figsize=(8, 6))

plt.semilogy(SNR_dB_range, BER_theoretical, label="Theoretical BER (4-PAM)", marker='o')

```
plt.semilogy(SNR_dB_range, BER_simulated, label="Simulated BER (4-PAM)", marker='x')

plt.title("BER vs SNR for 4-PAM with Raised Cosine Pulse (Theoretical vs Simulated)")

plt.xlabel("SNR (dB)")

plt.ylabel("BER")

plt.grid(True, which='both', linestyle='--', linewidth=0.5)

plt.show()
```

در نهایت، منحنی‌های BER نظری و شبیه‌سازی شده بر اساس SNR رسم می‌شوند. از مقیاس لگاریتمی برای محور عمودی استفاده شده است تا تغییرات در احتمال خطا به وضوح نمایش داده شود.

نتیجه‌گیری

این شبیه‌سازی نشان می‌دهد که چطور شکل‌دهی پالس Raised Cosine می‌تواند بر احتمال خطا در مدولاسیون PAM-4 تأثیر بگذارد. منحنی‌های BER نظری و شبیه‌سازی شده به خوبی همخوانی دارند و این نشان‌دهنده دقت روش‌های تحلیلی و شبیه‌سازی در ارزیابی عملکرد سیستم‌های مخابراتی است. این شبیه‌سازی می‌تواند به عنوان پایه‌ای برای تحلیل‌های پیشرفته‌تر در زمینه ارتباطات دیجیتال مورد استفاده قرار گیرد.