中山大学计算机学院

信号与系统

实验三 周期信号的傅里叶级数

（2024 学年春季学期）

课程名称：Signal and System

教学班级	202320353	专业（方向）	计算机科学与技术
学号	22320107	姓名	饶鉴晟
班级	计算机 3 班	日期	2024/4/12

一、 实验题目

A. 给定下图的周期三角信号$x(t)$，用 $Python$ 分别计算并绘制其复指数形式和三角函数形式的傅里叶级数系数，并用有限项级数$x_N(t)$ $(N = 1, \dots, 10)$ 逼近$x(t)$。

B. 对 2.2 中的周期方波和上面 $3-A$ 中的周期三角波，分别绘图演示其有限项级数$x_N(t)$当项数$N = 10, 100, 1000$时对$x(t)$的逼近效果。对比分析两者逼近过程中是否出现吉布斯（$Gibbs$）现象并解释原因。

C. 已知微分方程

$$\frac{dy(t)}{dt} + 2y(t) = x(t)$$

满足初始松弛条件。计算该方程所对应系统的频率响应。将 3-A 中的周期三角波输入该系统，用 Python 求解并绘制其输出响应。

二、 实验内容

1. 实验原理

实验三涉及周期信号的傅里叶级数分解。实验三的核心在于理解和应用周期信号的傅里叶级数（$Fourier\ series$）表示，以及分析其在线性时不变（$Linear\ Time-Invariant, LTI$）系统中的响应。

（1）实验原理

周期信号可以分解为一组以基波频率为整数倍的正弦波和余弦波的叠加，这就是傅里叶级数。傅里叶级数提供了一种方式，能够将任何周期信号表达为不同频率的简单正弦波的无限和。

周期信号可以看作是不同频率的正弦波和余弦波的无限叠加。每个正弦波和余弦波的频率都是基波频率的整数倍，这些波的叠加能够精确重构原始的周期信号。这种分解不仅适用于连续时间信号，也适用于离散时间信号。

（2）核心公式

周期信号 $x(t)$ 的傅里叶级数分解可以表示为两种形式：复指数形式和三角函数形式。

复指数形式的傅里叶级数分解可以用以下公式表示：

$$x(t) = \sum_{k=-\infty}^{+\infty} a_k e^{jk\omega_0 t}$$

其中，a_k是傅里叶级数，a_k可以用以下公式表示：

$$a_k = \frac{1}{T}\int_{-\frac{T}{2}}^{\frac{T}{2}} x(t)e^{-jk\omega_0 t}dt$$

三角函数形式的傅里叶级数分解可以用以下公式表示：

$$x(t) = \frac{a_0}{2} + \sum_{k=1}^{+\infty}[a_k\cos(\mathrm{k}\omega_0\mathrm{t}) + [b_k\sin(\mathrm{k}\omega_0\mathrm{t})]$$

其中，a_k和b_k是傅里叶级数，可以用以下公式表示：

$$\begin{cases} a_k = \dfrac{2}{T}\displaystyle\int_{-\frac{T}{2}}^{\frac{T}{2}} x(t)\cos(\mathrm{k}\omega_0\mathrm{t})dt \\ b_k = \dfrac{2}{T}\displaystyle\int_{-\frac{T}{2}}^{\frac{T}{2}} x(t)\sin(\mathrm{k}\omega_0\mathrm{t})dt \end{cases}$$

在实验三中，为了进行周期信号的傅里叶级数分解以及分析线性时不变系统对这些信号的响应，可能会用到一些主要*Python*库，包括：

Numpy: 用于处理数值计算，提供高效的数组操作功能，非常适合进行复杂数学计算，如傅里叶变换系数的计算和数组操作。

Matplotlib: 主要用于数据可视化，可以用来绘制信号图形和频谱，帮助直观展示信号及其傅里叶级数的效果。

Scipy: 包括 $scipy.signal$ 和 $scipy.integrate$，用于信号处理和数值积分。$scipy.signal$ 提供信号处理工具，如频率响应计算；$scipy.integrate$ 用于执行信号的积分运算，这在计算傅里叶系数时非常有用

这些库共同支持从基本的数学和信号处理运算到复杂的系统分析和图形可视化，为实验三提供了强大的技术支持。

2. 伪代码

任务$A-1$：

Algorithm 1: Fourier Series Analysis of a Periodic Rectangular Signal

Input: Amplitude A, pulse width τ, period T, number of harmonics N

Output: Fourier series coefficients X_n

Define a rectangular function $Rect(A, \tau, T, t)$ that returns A if $|tT| \leq \tau/2$ or $|tT| \geq T - \tau/2$, and 0 otherwise;
Set $T = 4$, $\tau = 2$, $\omega = 2\pi/T$, $A = 1$, $N = 10$;
Initialize X_n as an array of size $2N + 1$ to store the Fourier series coefficients;
Initialize nn as an array of indices from $-N$ to N;
Define an integrand function $integrand(t, n, \omega, A, T, \tau)$ that returns $Rect(A, \tau, T, t) \cdot e^{-jn\omega t}$;
for $n \leftarrow -N$ **to** N **do**
    $func \leftarrow \lambda t : integrand(t, n, \omega, A, T, \tau)$;
    $X_n[n + N] \leftarrow quad(func, 0, T)[0]/T$;
end
Create a figure with appropriate size;
Plot the magnitude spectrum of the Fourier series coefficients using stem plot;
Set the title to "Magnitude Spectrum of Periodic Rectangular Signal";
Set the x-axis label to "Harmonic Index n";
Set the y-axis label to "Magnitude $|X_n|$";
Adjust the font settings to display Chinese characters properly;
Add a grid;
Show the plot;

图 1

任务$A-2$：

Algorithm 1: Fourier Series Approximation of a Periodic Rectangular Signal

Input: Amplitude A, pulse width τ, period T, number of harmonics N

Output: Fourier series coefficients a_n and b_n, approximated signal $f_N(t)$

Define a function $Rect(A, \tau, T, t)$ that returns $\left(\frac{4A}{T}|t| - A\right)$ if $|t| \leq \frac{\tau}{2}$, and $\left(-\frac{4A}{T}|t| + 3A\right)$ otherwise;

Set $T = 4$, $\tau = 2$, $\omega = \frac{2\pi}{T}$, $A = 2$;

Initialize a_n and b_n as arrays of size $N + 1$ to store the Fourier series coefficients;

$a_n[0] \leftarrow \int_{-T/2}^{T/2} Rect(A, \tau, T, t) \cdot \frac{2}{T}\, dt$;

for $n \leftarrow 1$ **to** N **do**

  $a_n[n] \leftarrow \int_{-T/2}^{T/2} Rect(A, \tau, T, t) \cdot \cos(n\omega t) \cdot \frac{2}{T}\, dt$;

  $b_n[n] \leftarrow \int_{-T/2}^{T/2} Rect(A, \tau, T, t) \cdot \sin(n\omega t) \cdot \frac{2}{T}\, dt$;

end

Create a figure with appropriate size;

Define a function $f_N(t)$ that calculates the N-term Fourier series approximation of the rectangular signal;

Create a new figure with appropriate size;

Show the plots;

图 2

任务B

Algorithm 1: Filtering a Triangular Wave Signal

Input: Triangular wave amplitude A, period T, time array t

Output: Output signal $y(t)$

Define a function $triangular\_wave(A, T, t)$ that returns a triangular wave signal with amplitude A and period T;

Define a function $frequency\_response(\omega)$ that returns the frequency response $H(j\omega) = 1/(j\omega + 2)$;

Set $T = 10$, $A = 1$;

Initialize t as an array of time points from $-T$ to T with 1000 samples;

$x(t) \leftarrow triangular\_wave(A, T, t)$;

Initialize ω as an array of angular frequencies from -20 to 20 with 1000 samples;

$H(j\omega) \leftarrow frequency\_response(\omega)$;

$X(f) \leftarrow fft(x(t))$;

$Y(f) \leftarrow X(f) \times H(2\pi fftfreq(len(t), t[1] - t[0]))$;

$y(t) \leftarrow ifft(Y(f))$;

Create a figure with appropriate size;

Show the plot;

图 3

任务 C：

Algorithm 1: Filtering a Triangular Wave Signal

Input: Triangular wave amplitude A, period T, time array t

Output: Output signal $y(t)$

Define a function $triangular\_wave(A, T, t)$ that returns a triangular wave signal with amplitude A and period T;

Define a function $frequency\_response(\omega)$ that returns the frequency response $H(j\omega) = 1/(j\omega + 2)$;

Set $T = 10$, $A = 1$;

Initialize t as an array of time points from $-T$ to T with 1000 samples;

$x(t) \leftarrow triangular\_wave(A, T, t)$;

Initialize ω as an array of angular frequencies from -20 to 20 with 1000 samples;

$H(j\omega) \leftarrow frequency\_response(\omega)$;

$X(f) \leftarrow fft(x(t))$;

$Y(f) \leftarrow X(f) \times H(2\pi fftfreq(len(t), t[1] - t[0]))$;

$y(t) \leftarrow ifft(Y(f))$;

Create a figure with appropriate size;

Show the plot;

图 4

3. 关键代码展示（带注释）

任务$A-1$：

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import quad

# 定义新的矩形波函数
def Rect(A, tao, T, t):
    temp_t = np.abs(t) % T
    return np.where((temp_t <= tao/2) | (temp_t >= T-tao/2), A,
0)
# 参数设定
T = 4  # 基波周期
tao = 2  # 有效宽度
omega = 2 * np.pi / T  # 基波频率
A = 1  # 幅度
N = 10  # 谐波数

# 计算傅里叶级数系数
Xn = np.zeros(2*N+1)  # 傅里叶级数指数形式的幅值
nn = np.arange(-N, N+1)  # 傅里叶级数项数-N~N

# 积分函数
def integrand(t, n, omega, A, T, tao):
    return Rect(A, tao, T, t) * np.exp(-1j * n * omega * t)

# 对每个n计算傅里叶系数
for n in range(-N, N+1):
    func = lambda t: integrand(t, n, omega, A, T, tao)
    # 积分区间为一个完整周期
    Xn[n + N] = quad(func, 0, T)[0] / T

# 绘制频谱图
plt.figure(figsize=(10, 5))
plt.stem(nn, np.abs(Xn), use_line_collection=True)  # 绘制幅值频谱图
plt.title("周期矩形信号的傅里叶级数幅度频谱")
plt.xlabel("谐波数 n")
plt.ylabel("幅值 |Xn|")
plt.rcParams['font.sans-serif'] = ['SimHei']  # 用来正常显示中文标签
plt.rcParams['axes.unicode_minus'] = False  # 用来正常显示负号
plt.grid(True)
plt.show()
```

图 5

任务$A-2$（省略了无关代码）：

```python
def Rect(A, Tao, T, t):
    # 周期三角函数
    temp_t = abs(t)
    while(temp_t >= T):  # 将时间变量t转移到[0,T]区间上
        temp_t -= T
    return ((4*A/T)*temp_t-A) if temp_t <= T/2 else ((-4*A/T)*temp_t+3*A)  # 返回区间[0,T]内的函数值


def aNt(N, A, omega, T, tao):
    # 周期矩形傅里叶级数有限项级数系数
    an = np.zeros(N+1)  # 系数an，三角形式
    bn = np.zeros(N+1)  # 系数bn，三角形式
    FuncA0 = lambda t : Rect(A, tao, T, t)*2/T # 用于计算a0
    an[0] = quad(FuncA0, -T/2, T/2)[0]  # 计算a0
    for n in range(1, N+1):
        def FuncA(t): return Rect(A, tao, T, t)*np.cos(n*omega*t)*2/T
        def FuncB(t): return Rect(A, tao, T, t)*np.sin(n*omega*t)*2/T
        an[n] = quad(FuncA, -T/2, T/2)[0]
        bn[n] = quad(FuncB, -T/2, T/2)[0]
    return an, bn  # 返回傅里叶系数，从0~N


def plotSquare(A, T, tao):
    # 绘制周期矩形信号
    tt = np.arange(-3*T, 3*T, 0.1)  # 采样序列，用于绘制波形
    rect = np.array([Rect(A, tao, T, t) for t in tt])  # 产生周期矩形信号
    plt.figure(figsize=(20, 5))  # 通过figsize调整图大小
    plt.rcParams['font.sans-serif'] = ['SimHei']  # 用来正常显示中文标签
    plt.rcParams['axes.unicode_minus'] = False  # 用来正常显示负号
    plt.plot(tt, rect)  # 绘制周期矩形信号
    # 横坐标间隔，使用变量"_"是防止matplotlib打印输出无用信息
    _ = plt.xticks(np.arange(-3*T, 4*T, step=1.0))
    plt.title('周期矩形信号', {'size': 18})  # 显示title
    _ = plt.xlabel("Time: 基波周期T=4,宽度τ=2", {'size': 18})
    plt.grid(True)  # 显示网格


def xNt(an, bn, omega, T, N, tao):
    # 周期矩形傅里叶级数有限项级数
    A0 = an[0]
    t = np.arange(-3*T, 3*T, 0.1)  # 时间采样序列
    fnt = A0/2  # 直流项，即a0/2
    for n in range(1, N+1):
        fnt = fnt + an[n]*np.cos(n*omega*t) + bn[n]*np.sin(n*omega*t)  # 傅里叶级数N项逼近值
    return fnt


def plot_xNt(an, bn, omega, T, N, tao):
    tt = np.arange(-3*T, 3*T, 0.1)  # 时间采样序列
    plt.figure(figsize=(20, 20))  # 通过figsize调整图大小
    plt.rcParams['font.sans-serif'] = ['SimHei']  # 用来正常显示中文标签
    plt.rcParams['axes.unicode_minus'] = False  # 用来正常显示负号
    plt.subplots_adjust(wspace=0.2, hspace=1.0)  # 子图间距调整
    plt.subplot((N+2)//2, 2, 1)  # 绘制周期矩形信号的子图
    plt.plot(tt, np.array([Rect(A, tao, T, t) for t in tt]))  # 绘制周期矩形信号
    _ = plt.xticks(np.arange(-3*T, 4*T, step=T))
    plt.title("周期三角信号", {'size': 14})
    plt.grid(True)  # 显示网格
    plt.subplot((N+2)//2, 2, 2)  # 绘制周期矩形信号直流分量的子图
    plt.plot(tt, np.ones_like(tt)*an[0]/2)  # 绘制周期矩形信号直流分量
    _ = plt.xticks(np.arange(-3*T, 4*T, step=T))
    plt.title("周期三角信号直流分量", {'size': 14})
    plt.grid(True)  # 显示网格
    for i in range(1, N+1):
        # 显示N=1~10项傅里叶级数逼近
        plt.subplot((N+2)//2, 2, i+2)  # 绘制周期矩形信号及其N项傅里叶级数逼近的子图
        plt.plot(tt, np.array([Rect(A, tao, T, t) for t in tt]))  # 绘制周期矩形信号
        plt.plot(tt, xNt(an, bn, omega, T, i, tao))  # N=i项傅里叶级数逼近
        _ = plt.xticks(np.arange(-3*T, 4*T, step=T))  # 横坐标间隔
        plt.title(f"周期三角信号及其N={i}项傅里叶级数逼近", {'size': 14})
        plt.grid(True)  # 显示网格
    plt.show()  # 显示图像
```

图 6

任务 B：

```python
def Rect(A, Tao, T, t):
    # 使用向量化操作处理输入数组t
    temp_t = np.abs(t) % T
    return np.where(temp_t <= T/2, (4*A/T)*temp_t - A, (-4*A/T)*temp_t + 3*A)

def aNt(N, A, omega, T, tao):
    an = np.zeros(N+1)
    bn = np.zeros(N+1)
    # 根据N值动态设置高斯积分点数
    n_gauss = max(50, N * 5)  # 基本点数设置为N的5倍，至少为50

    FuncA0 = lambda t : Rect(A, tao, T, t) * 2 / T
    an[0] = fixed_quad(FuncA0, -T/2, T/2, n=n_gauss)[0]

    for n in range(1, N+1):
        FuncA = lambda t: Rect(A, tao, T, t) * np.cos(n * omega * t) * 2 / T
        FuncB = lambda t: Rect(A, tao, T, t) * np.sin(n * omega * t) * 2 / T
        an[n] = fixed_quad(FuncA, -T/2, T/2, n=n_gauss)[0]
        bn[n] = fixed_quad(FuncB, -T/2, T/2, n=n_gauss)[0]

    return an, bn

def plot_xNt(an, bn, omega, T, N, tao):
    tt = np.arange(-3*T, 3*T, 0.01)
    plt.figure(figsize=(10, 6))
    original_signal = np.array([Rect(A, tao, T, t) for t in tt])
    plt.plot(tt, original_signal, label="Original Signal", linestyle='dashed')
    fnt = an[0] / 2
    for n in range(1, N + 1):
        fnt += an[n] * np.cos(n * omega * tt) + bn[n] * np.sin(n * omega * tt)
    plt.plot(tt, fnt, label=f"Approximation N={N}")
    plt.title(f"Approximation with N={N} terms")
    plt.xlabel("Time")
    plt.ylabel("Signal")
    plt.legend()
    plt.grid(True)
    plt.show()

T = 4
tao = 2
omega = 2 * np.pi / T
A = 2
N_list = [10, 100, 1000]

for N in N_list:
    an, bn = aNt(N, A, omega, T, tao)
    plot_xNt(an, bn, omega, T, N, tao)
```

图 7

任务 C：

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import freqz
from scipy.fft import fft, ifft

# 定义三角波生成函数
def triangular_wave(A, T, t):
    temp_t = np.abs(t) % T
    return np.where(temp_t <= T/2, (4*A/T)*temp_t - A, (-4*A/T)*temp_t + 3*A)

# 定义系统的频率响应
def frequency_response(omega):
    return 1 / (1j * omega + 2)

# 时间轴参数
T = 10  # 三角波周期
A = 1   # 三角波振幅
t = np.linspace(-T, T, 1000)  # 时间数组

# 生成三角波
x_t = triangular_wave(A, T, t)

# 计算频率响应 H(jω) 在不同频率ω上的值
omega = np.linspace(-20, 20, 1000)  # 频率范围
H_jw = frequency_response(omega)

# 计算x(t)的FFT
X_f = fft(x_t)

# 使用频率响应 H(jω) 计算 Y(f)
Y_f = X_f * frequency_response(2 * np.pi * np.fft.fftfreq(len(t), d=t[1]-
t[0]))
# 逆FFT得到时域中的输出 y(t)
y_t = ifft(Y_f)

# 绘制输入x(t)和输出y(t)
plt.figure(figsize=(14, 6))
plt.subplot(1, 2, 1)
plt.plot(t, x_t, label='Input x(t)')
plt.title('Input Triangular Wave')
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.grid(True)

plt.subplot(1, 2, 2)
plt.plot(t, y_t.real, label='Output y(t)')  # 只绘制实部
plt.title('Output Response')
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.grid(True)
plt.tight_layout()
plt.show()
```

图 8

三、 实验结果及分析

1. 实验结果展示示例

（1）任务 A - 1:

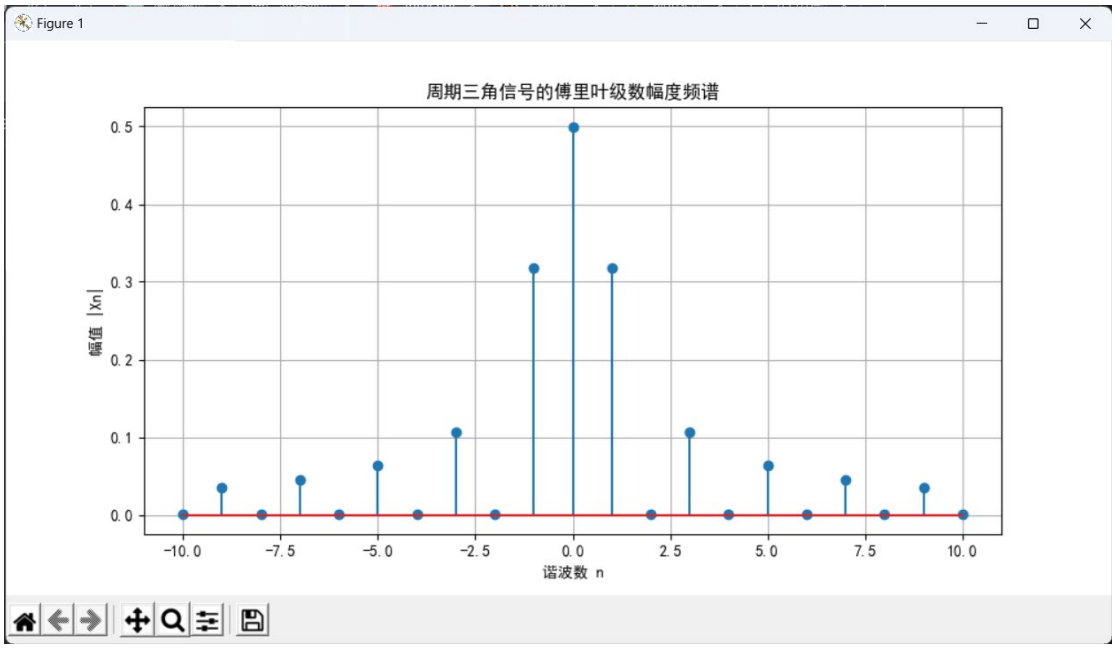

图 9

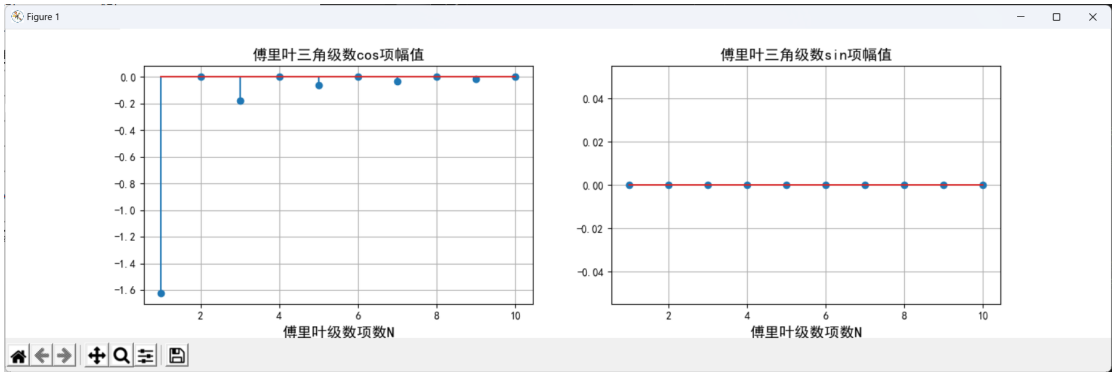

图 10

如图所示，图 9 和图 10 显示的频谱图显示了教材给定周期三角函数的复指数形式和三角函数形式的傅里叶级数系数。

（2）任务 A - 2:

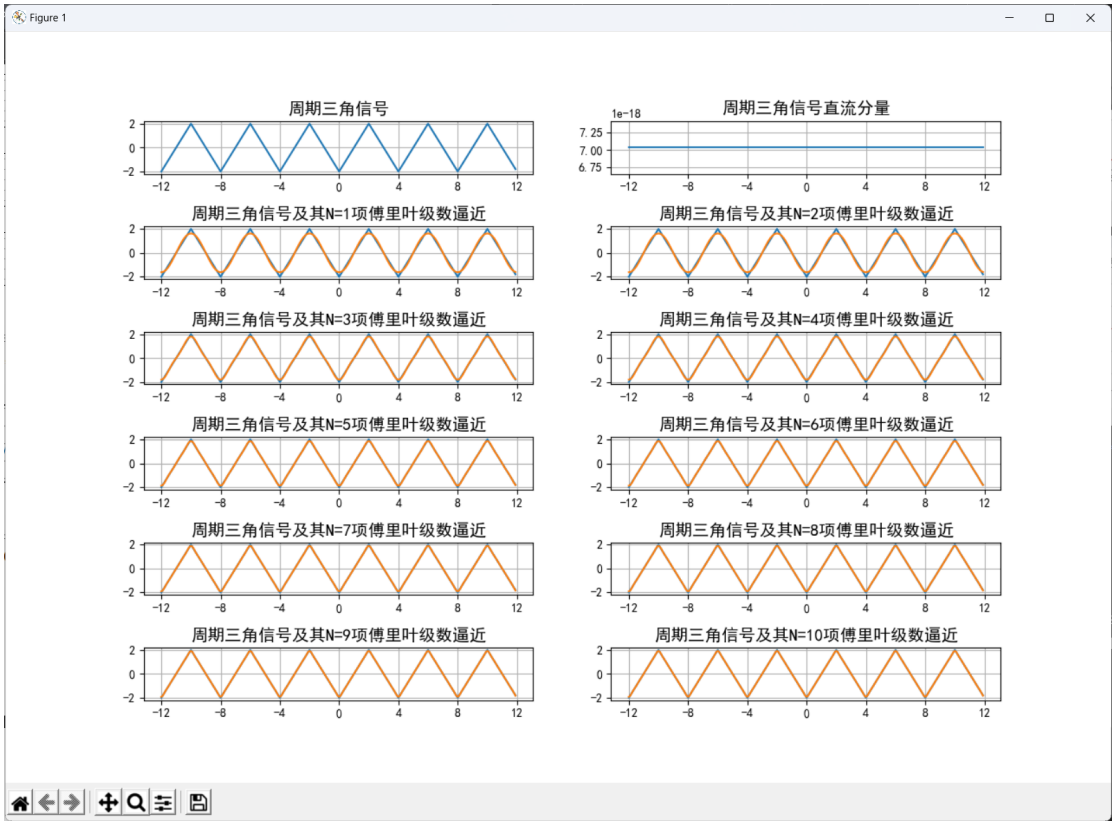

图 11

如图所示，上面的图象显示了教材给定周期三角函数的用有限项级数逼近的结果。

（3）关于任务 B 的讨论

我在第一次尝试完成任务 B 的时候，是沿用了样例当中的方法进行积分，但是效果并不好，在 $N = 1000$ 的时候积分结果和逼近结果出现了较大的误差，而在 $N = 100$ 和 $N = 10$ 的时候误差不大。

原始结果如下图所示。

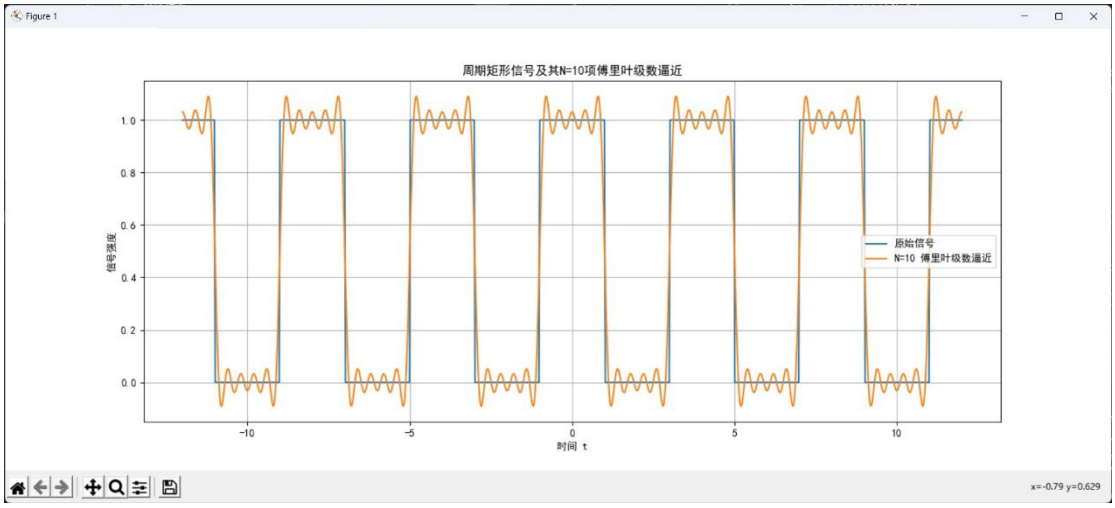

图 12

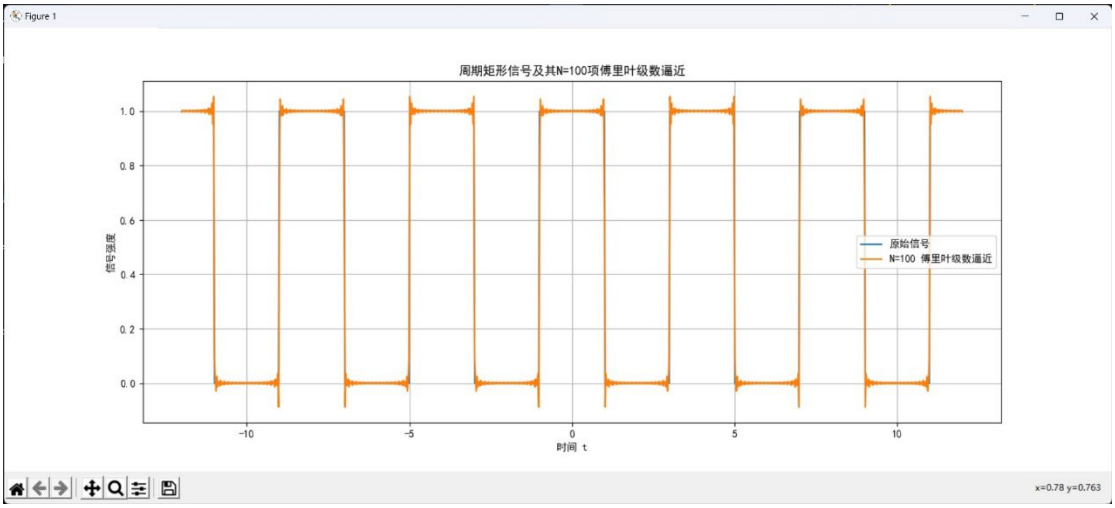

图 13

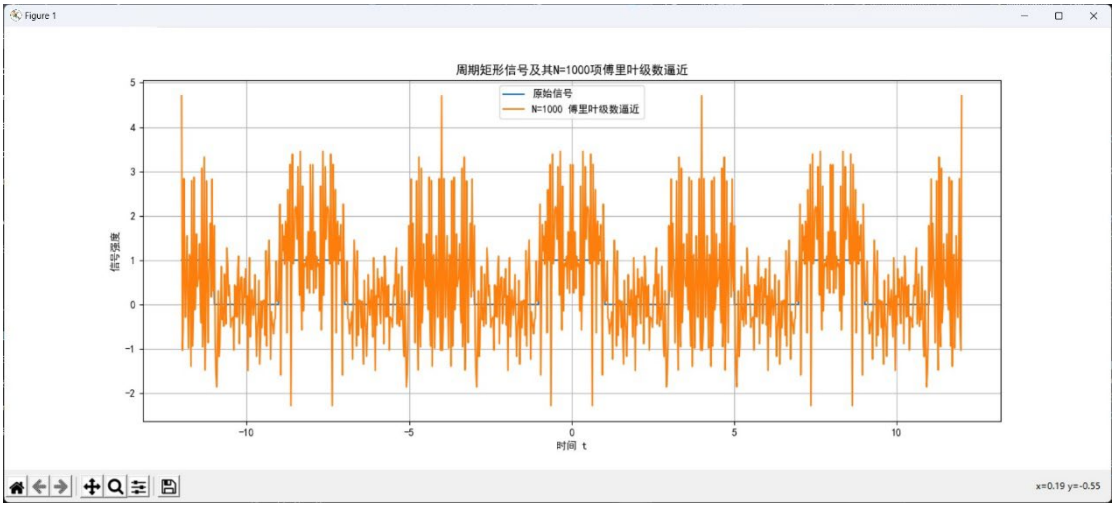

图 14

经过我和同班同学的讨论以及查阅公开资料，出现这种情况的原因很可能是因为积分的精度不足。尤其是在高频成分的计算中，积分的精度不足会导致较大的误差。

为了解决原有方法在积分精度不足上所导致的误差过大问题，我主要采用了以下几个优化措施：

1. 使用动态设置的高斯点数。在计算傅里叶系数时，代码动态地根据项数 N 设置高斯积分点的数量。这意味着随着 N 的增加，用于积分的高斯点也随之增加。这样可以更精确地捕捉函数的变化，尤其是在处理高频项时，这对于减少积分误差非常关键。关键代码如下所示。

$$n\_gauss = max(50, 10 * N)$$

2. 使用改进的数值积分方法。通过使用 $fixed\_quad$ 方法而不是简单的数值积分方法（如使用固定步长的数值方法），可以实现更高的积分精度。$fixed\_quad$ 基于高斯积分，这是一种高效且准确的积分方法，特别适合周期性和振荡性函数的积分。关键代码如下所示。

$$an[n] = fixed\_quad(FuncA, -T/2, T/2, n = n\_gauss)[0]$$

3. 进行向量化的信号处理。通过向量化的 $Rect$ 函数处理信号，允许函数直接处理时间数组，这样可以避免循环迭代每个时间点，提高计算效率和减少可能的数值误差。关键代码如下所示。

```
def Rect(A, tao, T, t):
    temp_t = np.abs(t)% T
    return np.where((temp_t <= tao/2) | (temp_t >= T - tao/2), A, 0)
```

经过以上优化措施，$N = 1000$ 时的误差大大降低，优化后的可视化图形如下图所示。

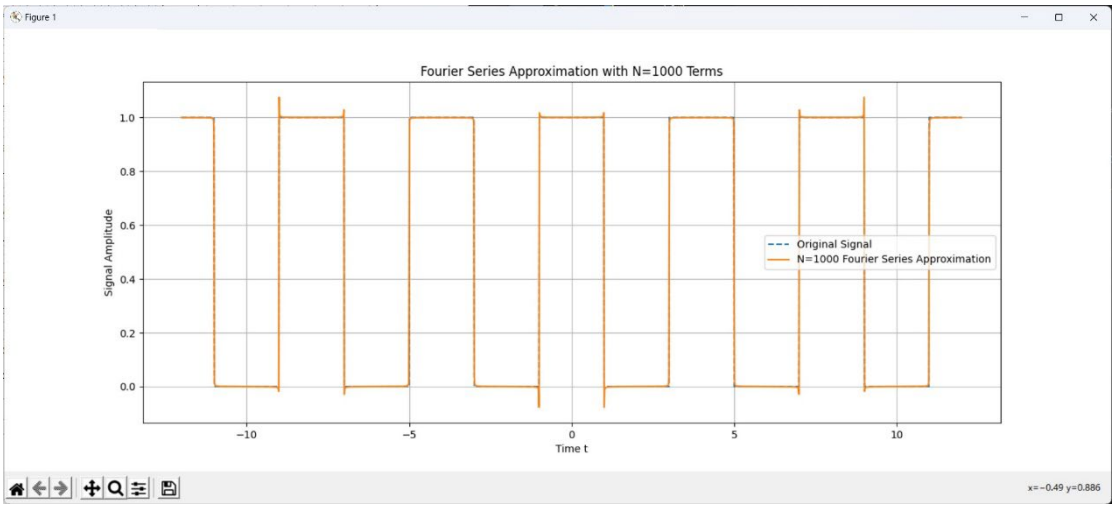
Figure 1
Fourier Series Approximation with N=1000 Terms
Signal Amplitude
Time t
Original Signal
N=1000 Fourier Series Approximation
x=-0.49 y=0.886

图 15

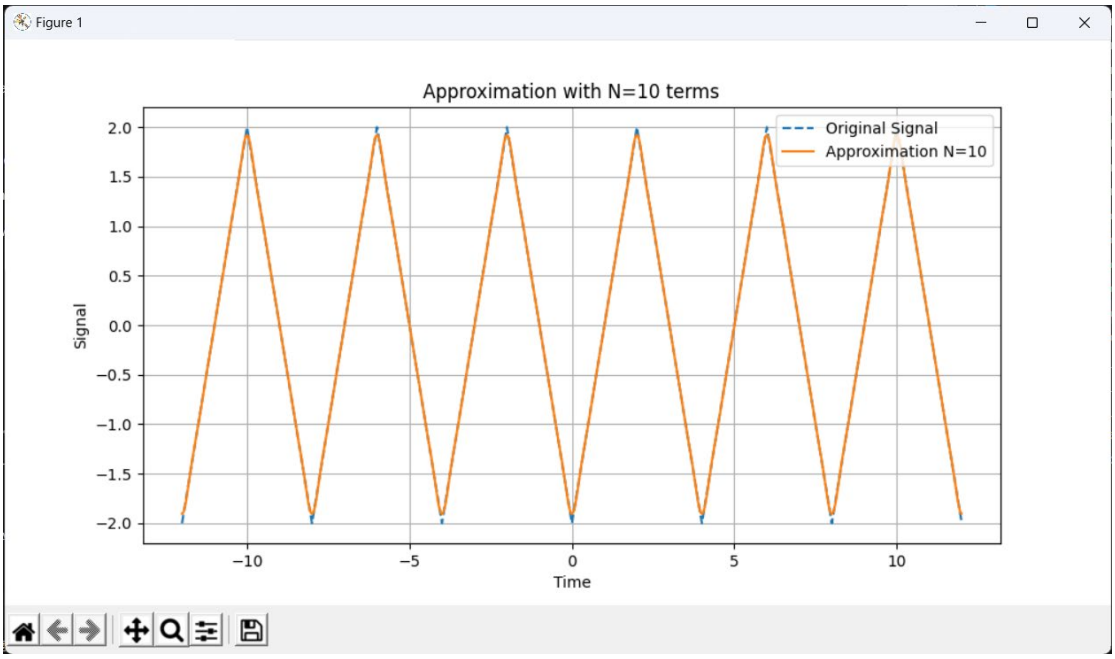
Figure 1
Approximation with N=10 terms
Signal
Time
Original Signal
Approximation N=10

图 16

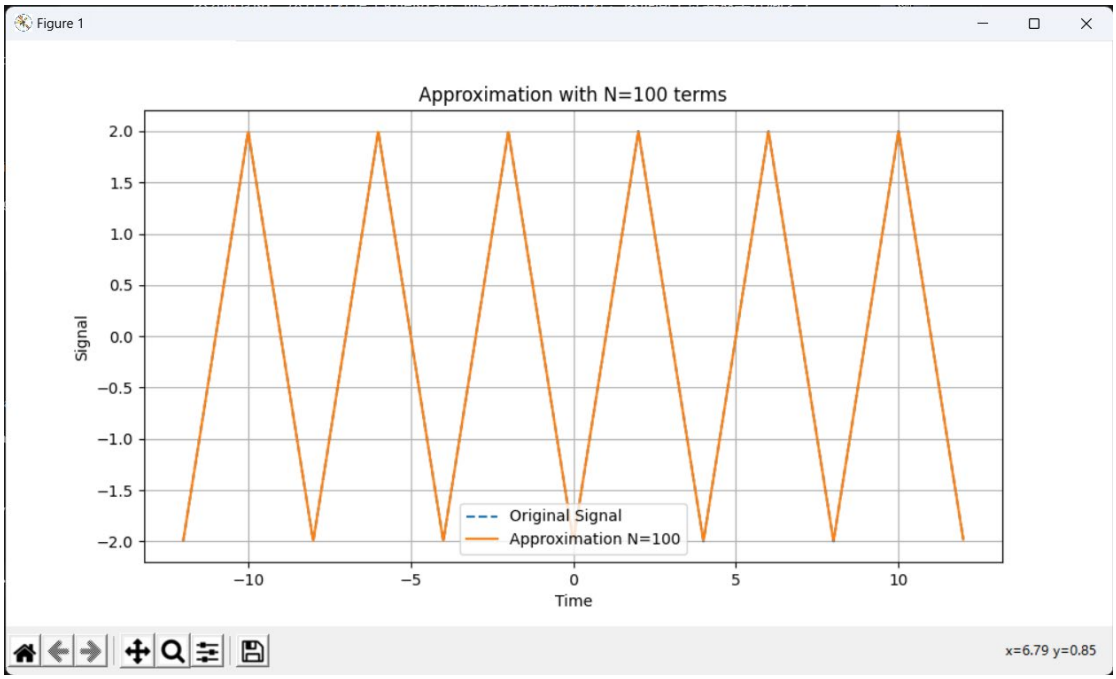

图 17

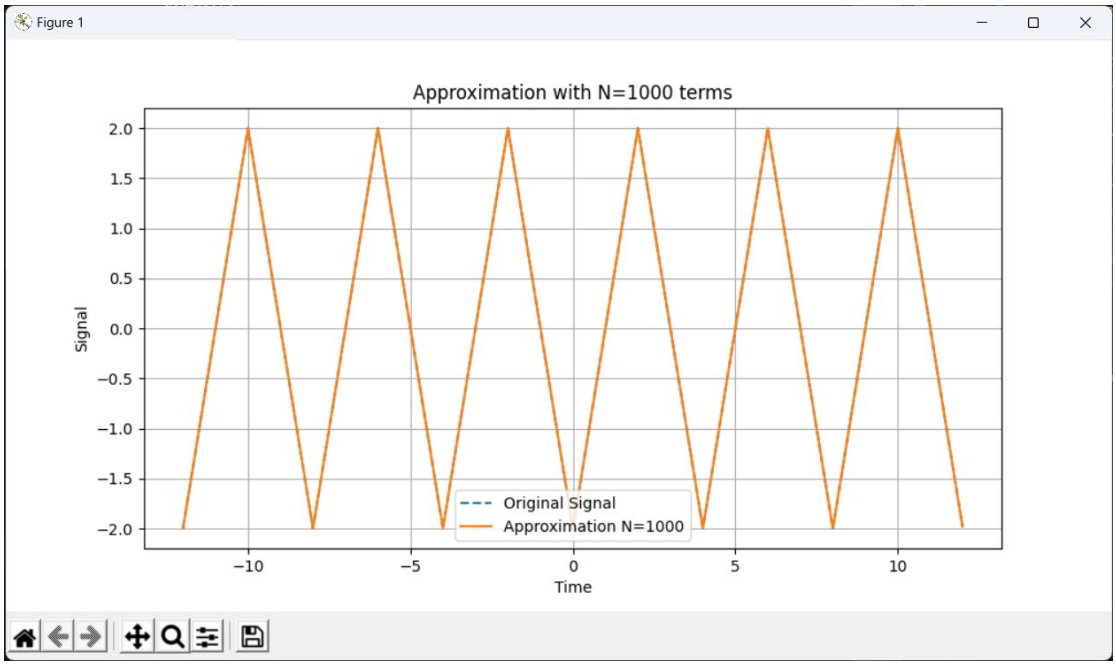

图 18

三角信号与矩形信号在吉布斯现象上的表现差异较大，矩形信号的吉布斯现象较为明显，而三角信号的吉布斯现象几乎无法观察到，这主要是由于两个信号的不连续点具有差异。周期矩形信号有锐利的跳变点（从 0 到最大值或从最大值到 0）。在这些跳变点，信号的导数不存在，或可以说是无穷大。这种类型的不连续性在傅里叶级数逼近中导致了明显的

吉布斯现象，即在跳变点附近的逼近信号会出现超调和振荡。而周期三角信号具有连续的线性变化部分，即使在周期的峰值处也只是导数的变化，而不是函数值的突变。三角信号的连续性质和渐进的变化使其在傅里叶级数逼近时不易出现明显的吉布斯现象。

任务 C：

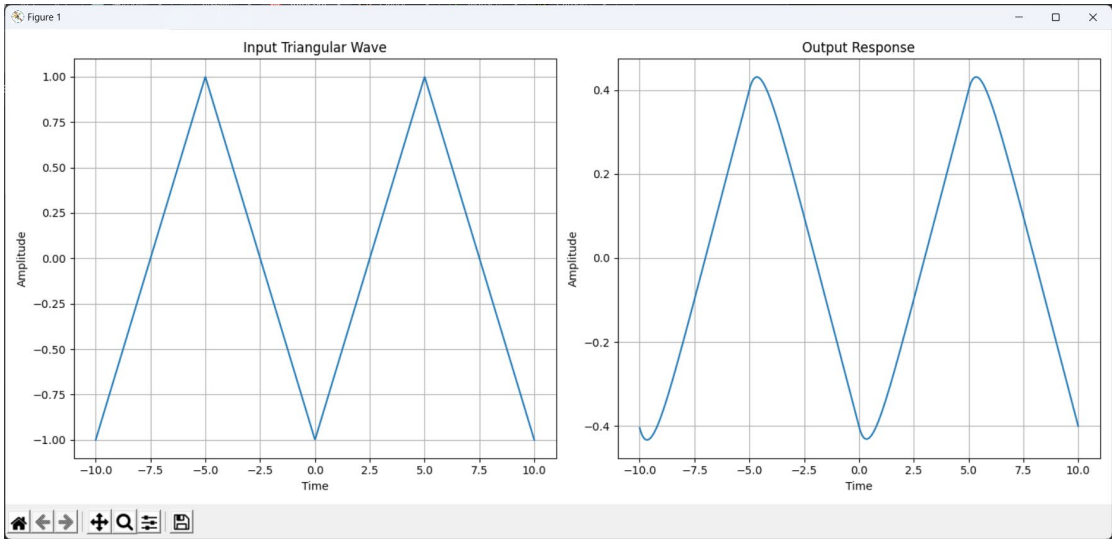

图 19

如图所示，输入$3-A$中的周期三角波后，可视化界面正确显示了微分方程

$$\frac{dy(t)}{dt}+2y(t)=x(t)$$

所对应的系统的频率响应。

四、 参考资料

[1] NumPy 开发团队. (n.d.). NumPy 官方参考文献. 取自 https://www.numpy.org.cn/reference/. Accessed April 10, 2024.
[2] SciPy 开发团队. (n.d.). SciPy 官方参考文档. 取自 https://docs.scipy.org/doc/scipy/index.html. Accessed April 10, 2024.
[3] Matplotlib 开发团队. (n.d.). Matplotlib 官方参考文档. 取自 https://www.matplotlib.net/stable/index.html. Accessed April 10, 2024.