lab1

一、实验目的

1. 掌握信号的表示及其可视化方法。
2. 掌握信号基本时域运算的实现方法。
3. 实现线性时不变LTI系统的全响应求解，并把基于仿真平台内置函数的仿真结果与理论计算结果进行比较。
4. 实现周期信号的傅里叶级数展开。

二、实验报告要求

1. 提交：实验报告一份，PDF格式，其他格式拒收。

实验报告中需要包括：

a）　若题目要求理论结果，报告中需要给出理论结果。

b）　结果图；图中需要有适当的标识、横坐标、纵坐标等。

c）　源代码。源代码中要有合适的注释。

d）　实验体会和感悟。

2. 提交实验报告规则：

e）　2022年10月28日12am之前将实验报告发到助教邮箱。

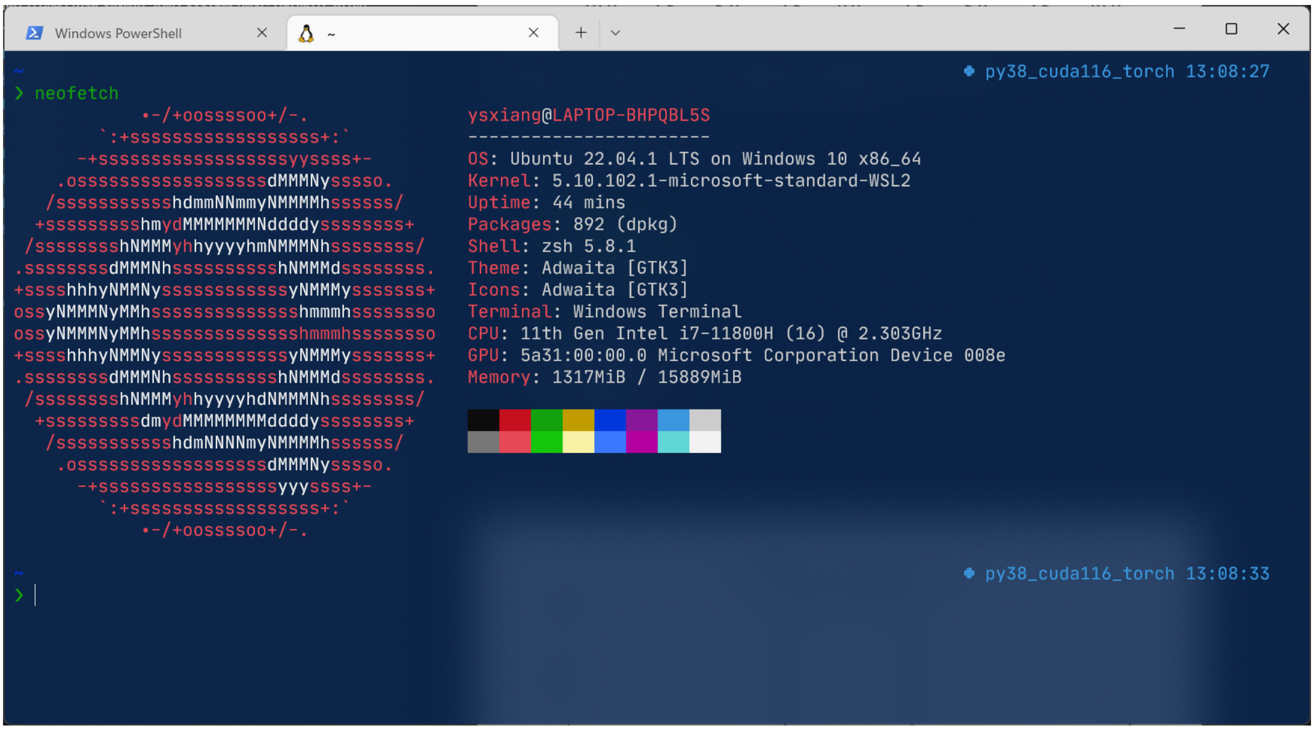
（第一课堂交给施锐，邮箱：296206140@qq.com;

第二课堂曹歌，邮箱：1765578099 @qq.com）

文件名命名规则：课堂号-学号-姓名-第几次实验。（比如第2课堂的学生，姓名：李三，学号为2019050，第2次实验，文件名命名为：2-2019050-李三-2）

三、实验环境

```
OS: Ubuntu 22.04.1 LTS on Windows 10 x86_64
Kernel: 5.10.102.1-microsoft-standard-WSL2
Shell: zsh 5.8.1
Terminal: Windows Terminal
CPU: 11th Gen Intel i7-11800H (16) @ 2.303GHz
GPU: 5a31:00:00.0 Microsoft Corporation Device 008e
Memory: 1319MiB / 15889MiB
```



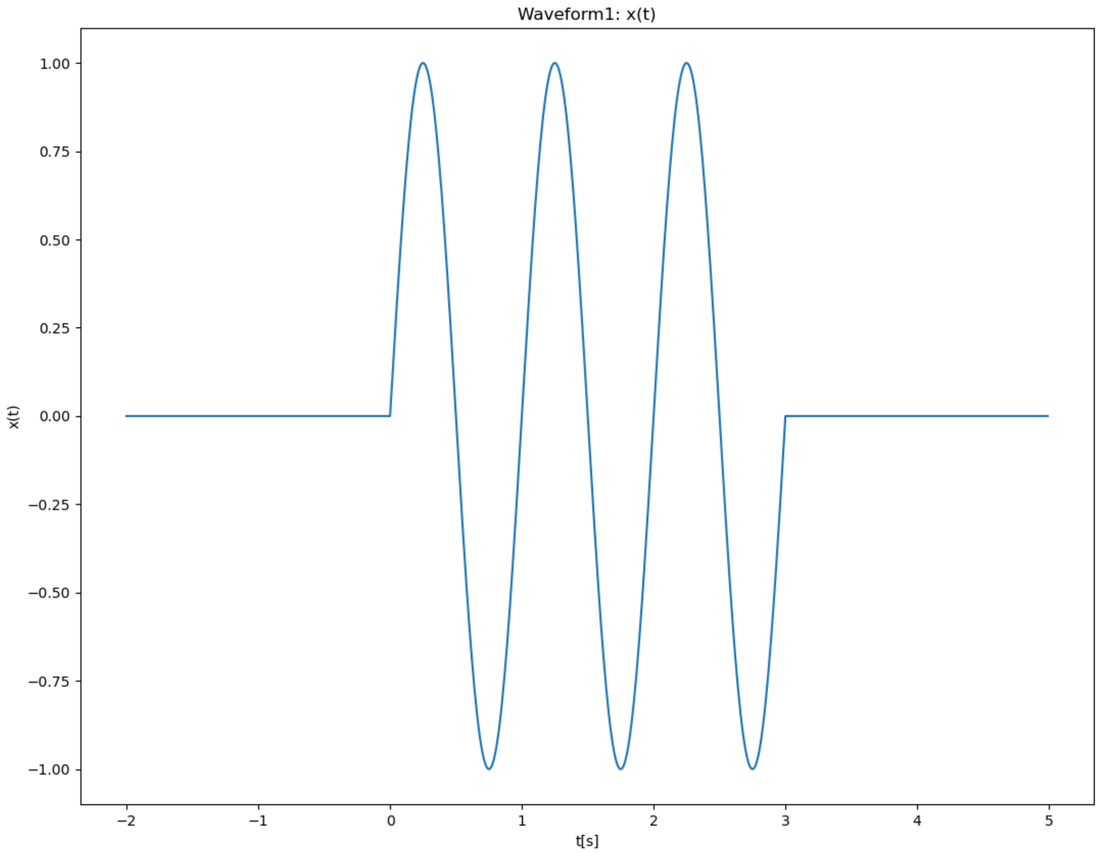
解释器：Anaconda(python=3.8)

四、实验内容与实验结果

1. 利用Python绘制下列连续时间信号的波形

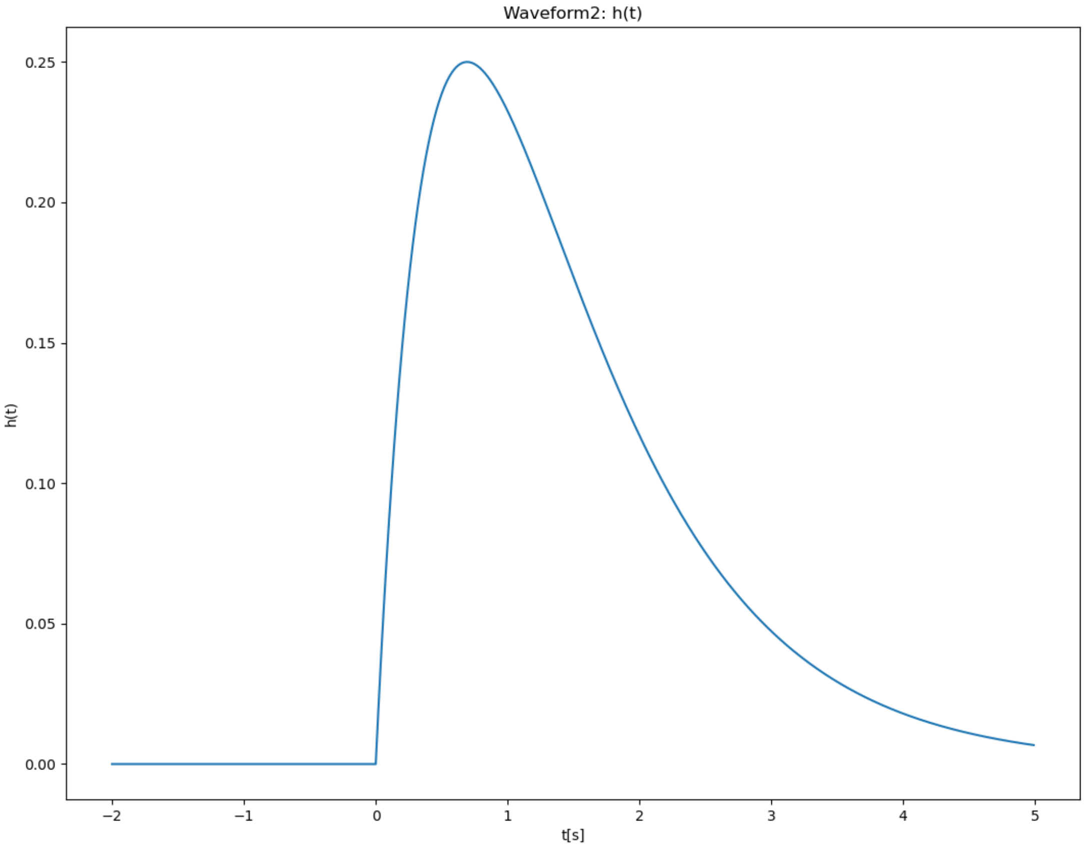
(1) $x(t)=\sin{(2\pi t)}[\varepsilon(t)-\varepsilon(t-3)]$，其中，$\varepsilon(t)$为阶跃函数

```
# 导包
import numpy as np
import matplotlib.pyplot as plt
import scipy.signal as sgn

# $x(t)=\sin (2 \pi t)[\varepsilon(t)-\varepsilon(t-3)]$
def draw_waveform1():
    # 规定t的范围为-2到5，梯度0.01递增
    t = np.arange(-2, 5, 0.01)
    # x(t) 函数，使用numpy的heaviside函数作为阶跃函数
    x = np.sin(2 * np.pi * t) * (np.heaviside(t, 1) - np.heaviside(t - 3, 1))

    #作图。横轴为t，纵轴为x
    plt.plot(t, x)
    #以下三句声明横轴和纵轴的标签以及图像
    plt.xlabel("t[s]")
    plt.ylabel("x(t)")
    plt.title("Waveform1: x(t)")
    #绘制图像
    plt.show()

draw_waveform1()
```



(2) $h(t) = e^{-t}\varepsilon(t) - e^{-2t}\varepsilon(t)$

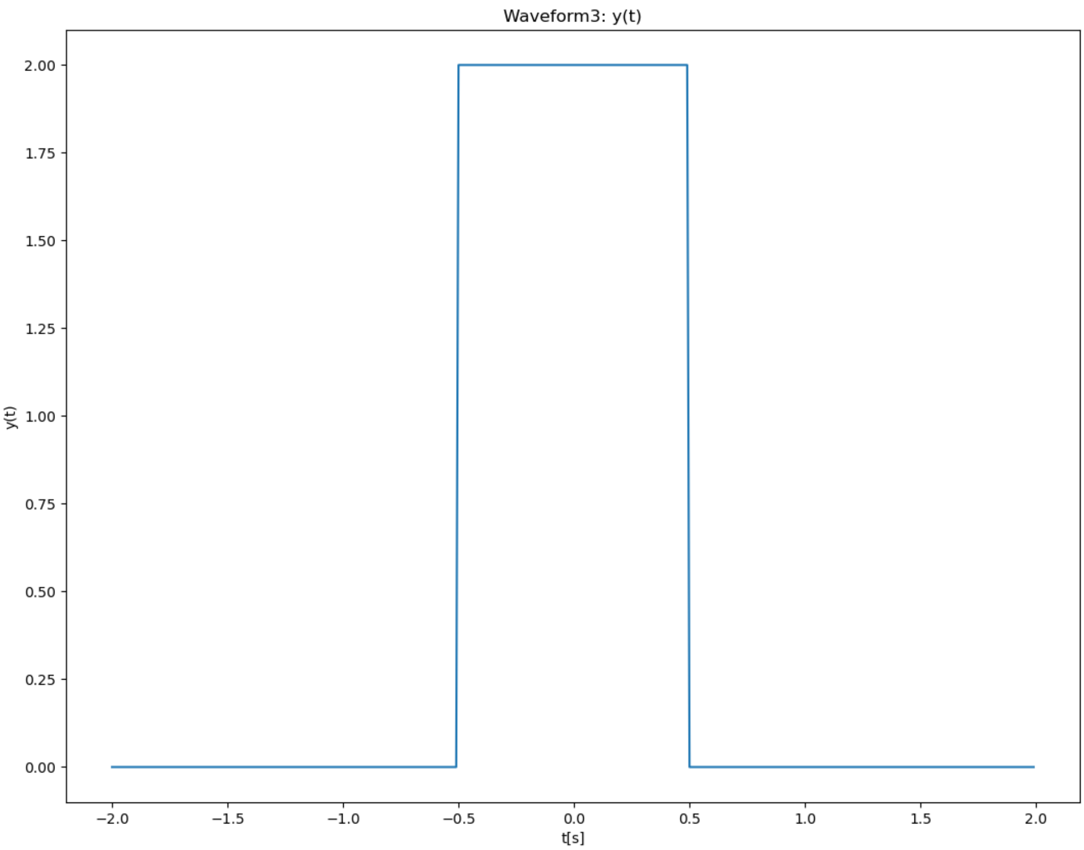
```
# $h(t)=e^{-t} \varepsilon(t)-e^{-2 t}
\varepsilon(t)$
def draw_waveform2():
    t = np.arange(-2, 5, 0.01)
    h = np.exp(-t) * np.heaviside(t, 1) - np.exp(-2 *
t) * np.heaviside(t, 1)

    plt.plot(t, h)
    plt.xlabel("t[s]")
    plt.ylabel("h(t)")
    plt.title("Waveform2: h(t)")
    plt.show()

draw_waveform2()
```



(3) $y(t)=2G_2(t)$

```python
# 门函数
def gate_function(variable: np.ndarray) -> np.ndarray:
    return np.heaviside(variable + 0.5, 1) - np.heaviside(variable - 0.5, 1)


# $y(t)=2 G_{2}(t)$
def draw_waveform3():
    t = np.arange(-2, 2, 0.01)
    y = 2 * gate_function(t)

    plt.plot(t, y)
    plt.xlabel("t[s]")
    plt.ylabel("y(t)")
    plt.title("Waveform3: y(t)")
    plt.show()
```

```python

draw_waveform3()
```

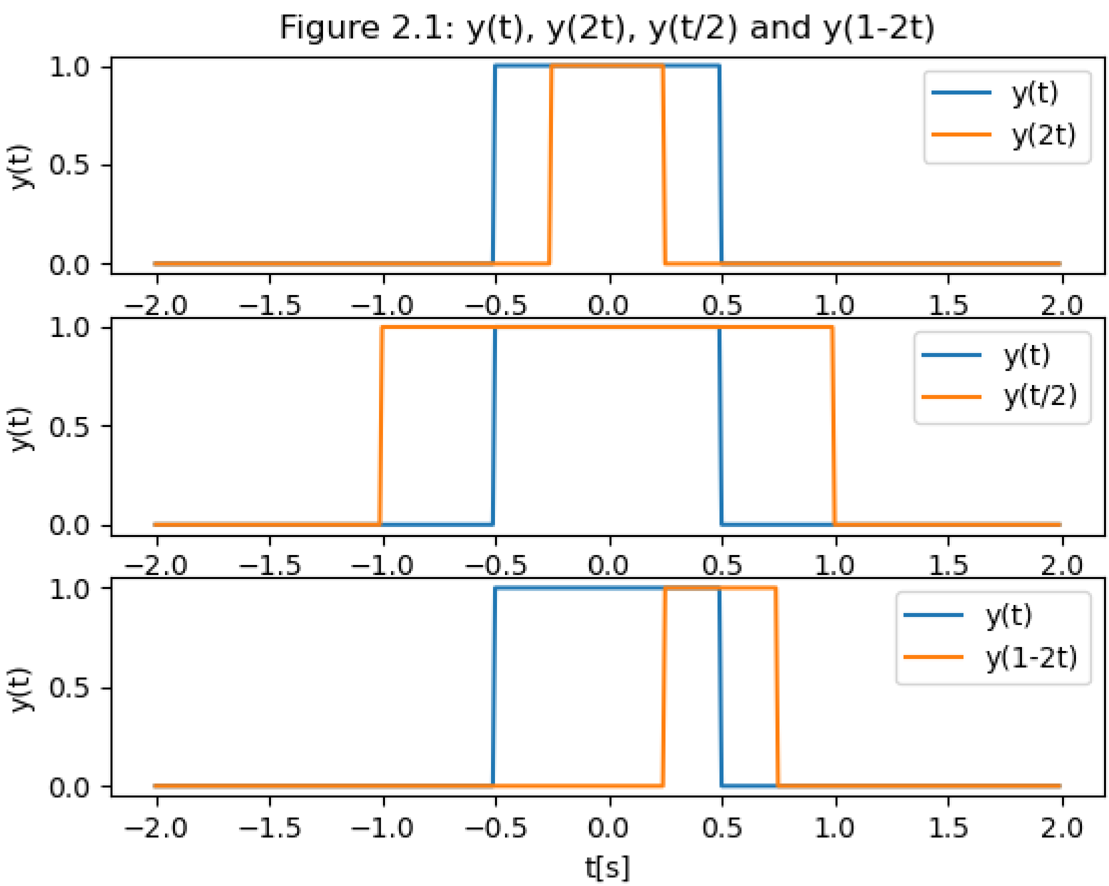


2. 利用Python验证信号的基本运算

(1) 以单位门函数$y(t) = G_1(t)$为例，画出$y(2t), y\left(\frac{t}{2}\right), y(2-2t)$。注意观察Python画出的结果是否和理论分析得出的结果一致。

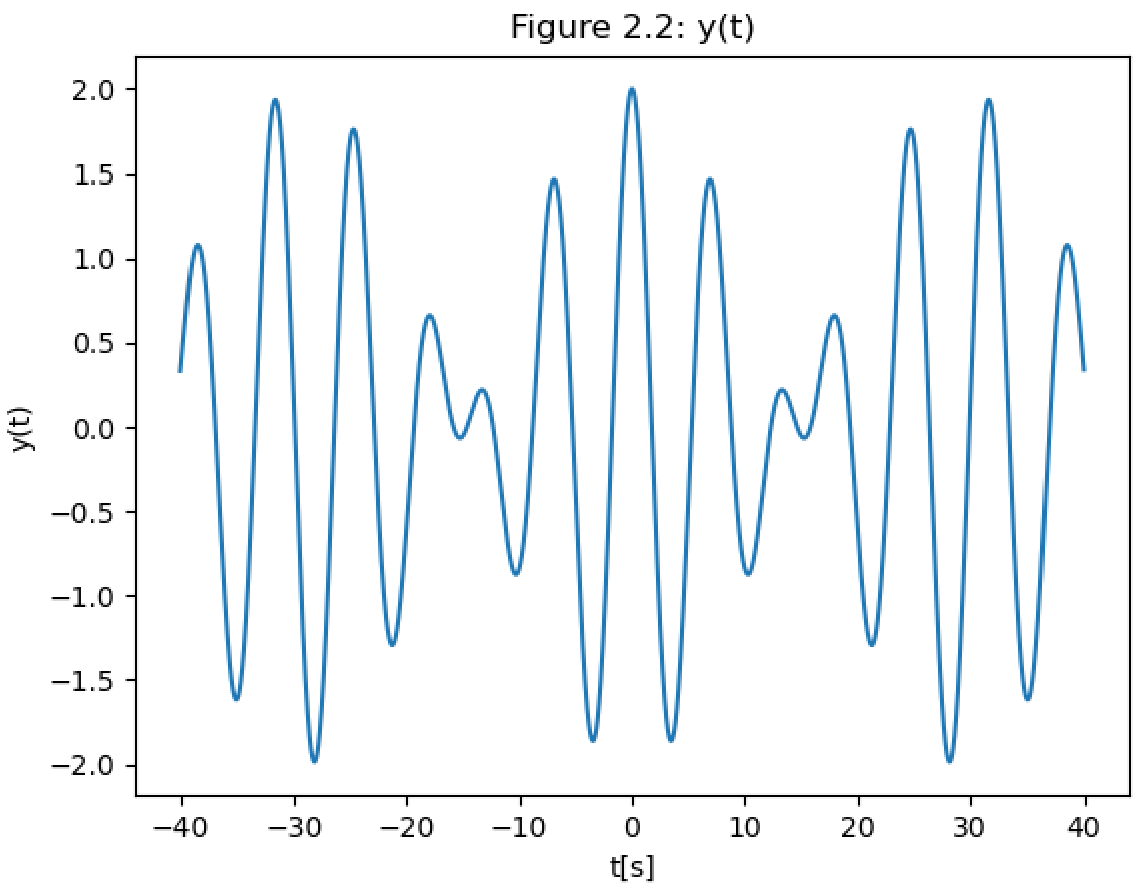
```python
# 验证信号的基本运算
# 导包
import numpy as np
import matplotlib.pyplot as plt

# 门函数
def gate_function(variable: np.ndarray) -> np.ndarray:
    return np.heaviside(variable + 0.5, 1) - np.heaviside(variable - 0.5, 1)

```

```
def verify1():
    t = np.arange(-2, 2, 0.01)
    # G(t)
    gate = gate_function(t)
    # y(2 t)
    y1 = gate_function(2 * t)
    # y(t / 2)
    y2 = gate_function(t / 2.0)
    # y(1 - 2t)
    y3 = gate_function(1 - 2 * t)
    # 绘制第一幅图像
    plt.subplot(3, 1, 1)
    plt.plot(t, gate, t, y1)
    plt.xlabel("t[s]")
    plt.ylabel("y(t)")
    plt.legend(['y(t)', 'y(2t)'])
    plt.title("Figure 2.1: y(t), y(2t), y(t/2) and y(1-2t)")
    # 第二幅图像
    plt.subplot(3, 1, 2)
    plt.plot(t, gate, t, y2)
    plt.xlabel("t[s]")
    plt.ylabel("y(t)")
    plt.legend(['y(t)', 'y(t/2)'])
    # 第三幅图像
    plt.subplot(3, 1, 3)
    plt.plot(t, gate, t, y3)
    plt.xlabel("t[s]")
    plt.ylabel("y(t)")
    plt.legend(['y(t)', 'y(1-2t)'])
    # 绘图
    plt.savefig("Figure 2.1.png")
    plt.show()

verify1()
```



(2) 画出$\cos(t)+\cos\left(\frac{\pi}{4}t\right)$，并观察其是否为周期函数，如果是，周期为多少?

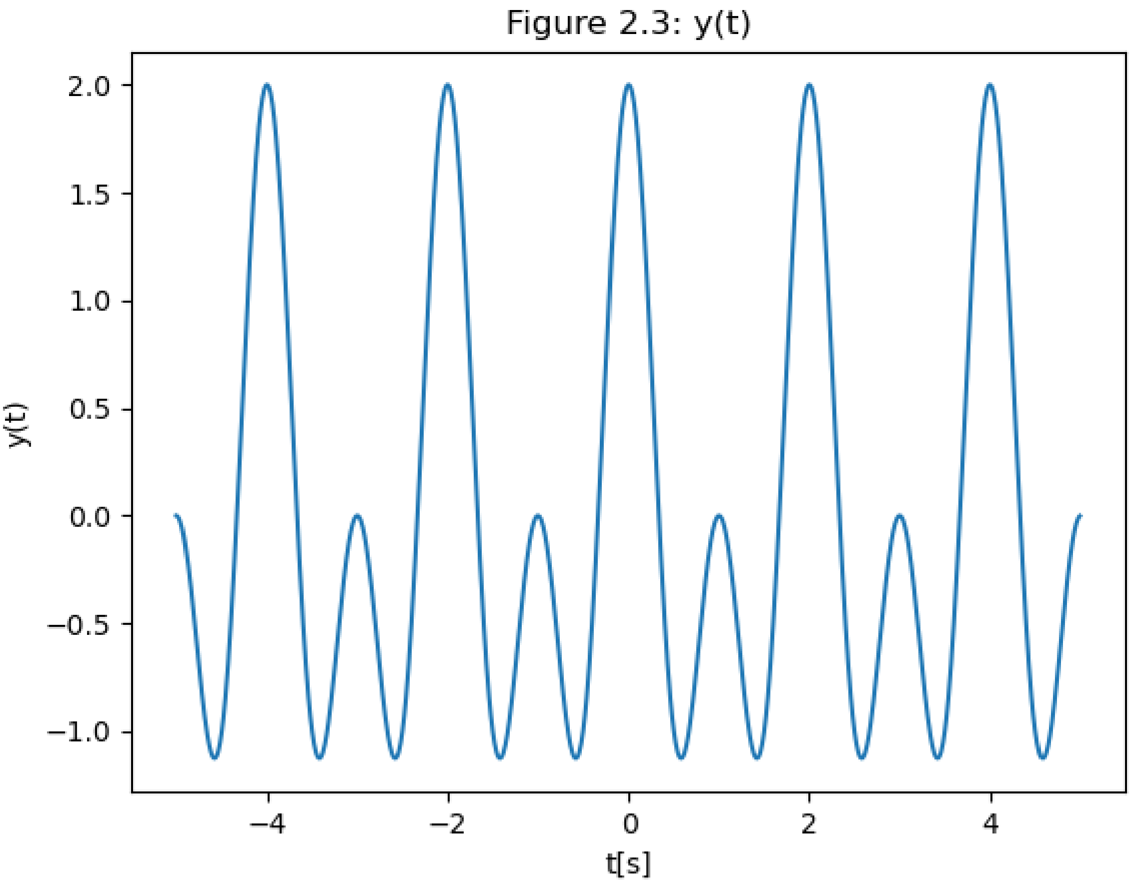
```python
# $\cos (t)+\cos \left(\frac{\pi}{4} t\right)$
def verify2():
    t = np.arange(-40, 40, 0.01)
    y = np.cos(t) + np.cos(np.pi * t / 4.0)

    plt.plot(t, y)
    plt.xlabel("t[s]")
    plt.ylabel("y(t)")
    plt.title("Figure 2.2: y(t)")
    plt.savefig("./lab1/img/Figure 2.2.png")
    plt.show()

verify2()
```



观察得$\cos(t)+\cos\left(\frac{\pi}{4}t\right)$不是周期函数

(3) 画出$\cos(\pi t)+\cos(2\pi t)$，并观察其是否为周期函数，如果是，周期为多少?

```
# $\cos (\pi t)+\cos \left(2 \pi t\right)$
def verify3():
    t = np.arange(-5, 5, 0.01)
    y = np.cos(np.pi * t) + np.cos(2 * np.pi * t)

    plt.plot(t, y)
    plt.xlabel("t[s]")
    plt.ylabel("y(t)")
    plt.title("Figure 2.3: y(t)")
    plt.savefig("./lab1/img/Figure 2.3.png")
    plt.show()

verify3()
```



如图，$\cos(\pi t) + \cos(2\pi t)$是周期函数，周期为2

3. 卷积运算

已知$x(t) = \left[e^{-2t}\varepsilon(t)\right] * \left[e^{-3t}\varepsilon(t)\right]$

(1) 根据卷积的定义，推导得到$x(t)$的理论值；

$$x(t) = \left[e^{-2t}\varepsilon(t)\right] * \left[e^{-3t}\varepsilon(t)\right]$$

$$= \int_{-\infty}^{+\infty} e^{-2\tau}\varepsilon(\tau)\cdot e^{-3(t-\tau)}\varepsilon(t-\tau)\,d\tau$$

$$= \int_{0}^{t} e^{\tau-3t}\varepsilon(\tau)\,\varepsilon(t-\tau)\,d\tau$$

$$= e^{\tau-3t}\Big|_{0}^{t}$$

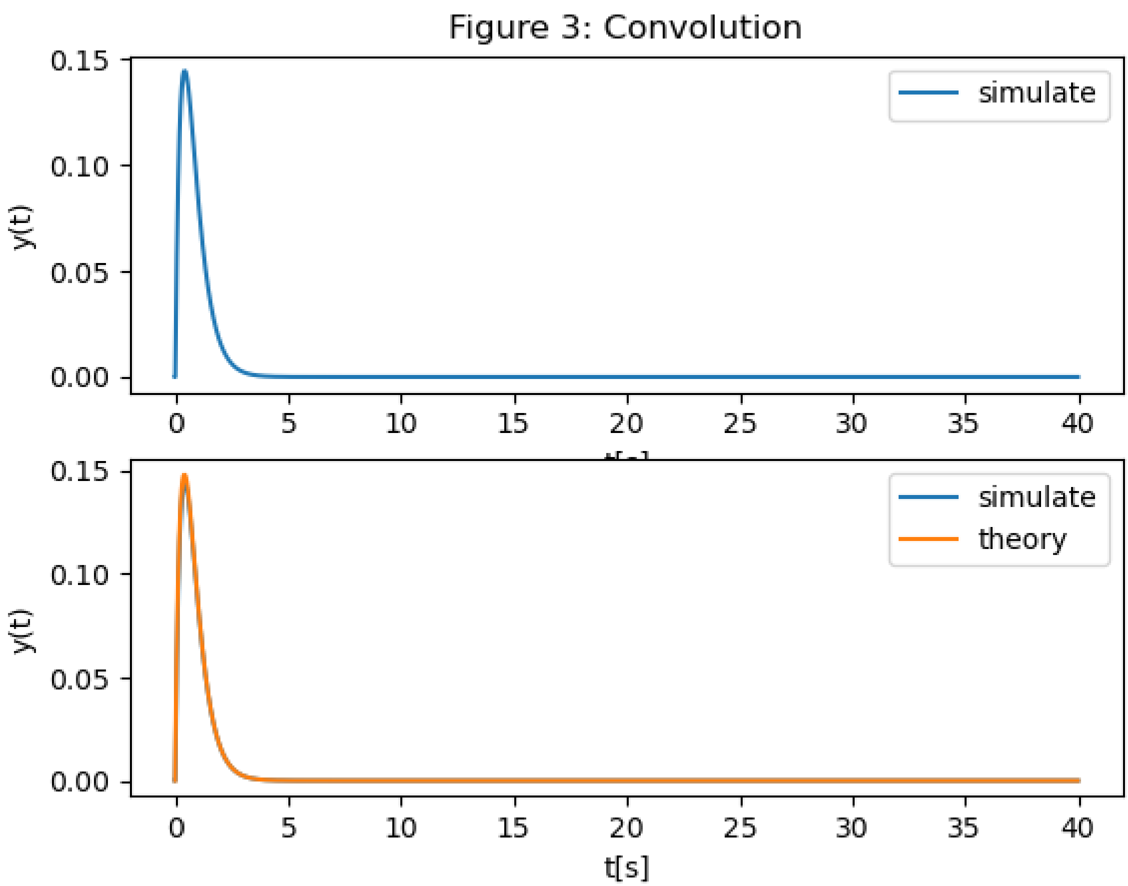
$$= e^{-2t} - e^{-3t},\ t>0$$

(2)利用MATLAB的conv函数获得$x(t)$的数值

```
# 卷积运算
# $x(t)=\left[e^{-2 t} \varepsilon(t)\right]*\left[e^{-3 t} \varepsilon(t)\right]$
# 导包
import numpy as np
import matplotlib.pyplot as plt
import scipy.signal as sgn

STEP_SIZE = 0.01


def simulate_theory():
    # 绘制模拟数值图像
    t = np.arange(0, 20, STEP_SIZE)
    y1 = np.exp(-2 * t) * np.heaviside(t, 0)
    y2 = np.exp(-3 * t) * np.heaviside(t, 0)
    # 将y1和y2进行卷积。
    y = sgn.convolve(y1, y2) * STEP_SIZE # 由于计算是离散的点，卷积后需要乘以步长。
    plt.subplot(2, 1, 1)
```

```
    plt.plot(np.arange(0, 39.99, STEP_SIZE), y,
label='simulate')
    plt.xlabel("t[s]")
    plt.ylabel("y(t)")
    plt.legend()
    plt.title("Figure 3: Convolution")
    # 绘制理论值图像
    t_theory = np.arange(0, 39.99, STEP_SIZE)
    y_theory = (np.exp(-2 * t_theory) - np.exp(-3 *
t_theory)) * np.heaviside(t_theory, 1)
    plt.subplot(2, 1, 2)
    plt.plot(t_theory, y, t_theory, y_theory,
label='theory')
    plt.xlabel("t[s]")
    plt.ylabel("y(t)")
    plt.legend(['simulate', 'theory'])
    plt.savefig("./lab1/img/Figure 3.png")
    plt.show()

simulate_theory()
```

(3)把此数值画出来，并且与理论值相比对，查看其有无差异。



如图可知，python仿真值与理论值一致。

4. 求解系统的零状态响应

设有一个线性时不变系统，其微分方程为
$r''(t)+3r'(t)+2r(t)=e(t)$，其中$e(t)$为输入信号，$r(t)$为系统输出，$e(t)=e^{-2t}\varepsilon(t)$。

（1） 根据理论推导获得系统的零状态响应$r_{\mathrm{zs}}(t)$。

对于微分方程 $\ddot{r}(t)+3\dot{r}(t)+2r(t)=e(t)$

其转移算子， $H(p)=\dfrac{1}{p^2+3p+2}=\dfrac{1}{p+1}-\dfrac{1}{p+2}$

得到冲激函数 $h(t)=H(p)\cdot\delta(t)=(e^{-t}-e^{-2t})\varepsilon(t)$

得到零状态响应：

$$
\begin{aligned}
r_{zs} &= e(t) * h(t)\\
&= e^{-2t}\varepsilon(t) * (e^{-t}-e^{-2t})\varepsilon(t)\\
&= \int_{-\infty}^{+\infty}(e^{-\tau}-e^{-2\tau})\varepsilon(\tau)\cdot e^{-2(t-\tau)}\varepsilon(t-\tau)\,d\tau\\
&= \int_{-\infty}^{+\infty}e^{\tau-2t}\varepsilon(\tau)\varepsilon(t-\tau)\,d\tau-\int_{-\infty}^{+\infty}e^{-2t}\varepsilon(\tau)\varepsilon(t-\tau)\,d\tau\\
&= e^{\tau-2t}\Big|_0^t - te^{-t}\\
&= (e^{-t}-e^{-2t})-te^{-2t}\\
&= e^{-t}-(1+t)e^{-2t} \quad ,t>0
\end{aligned}
$$

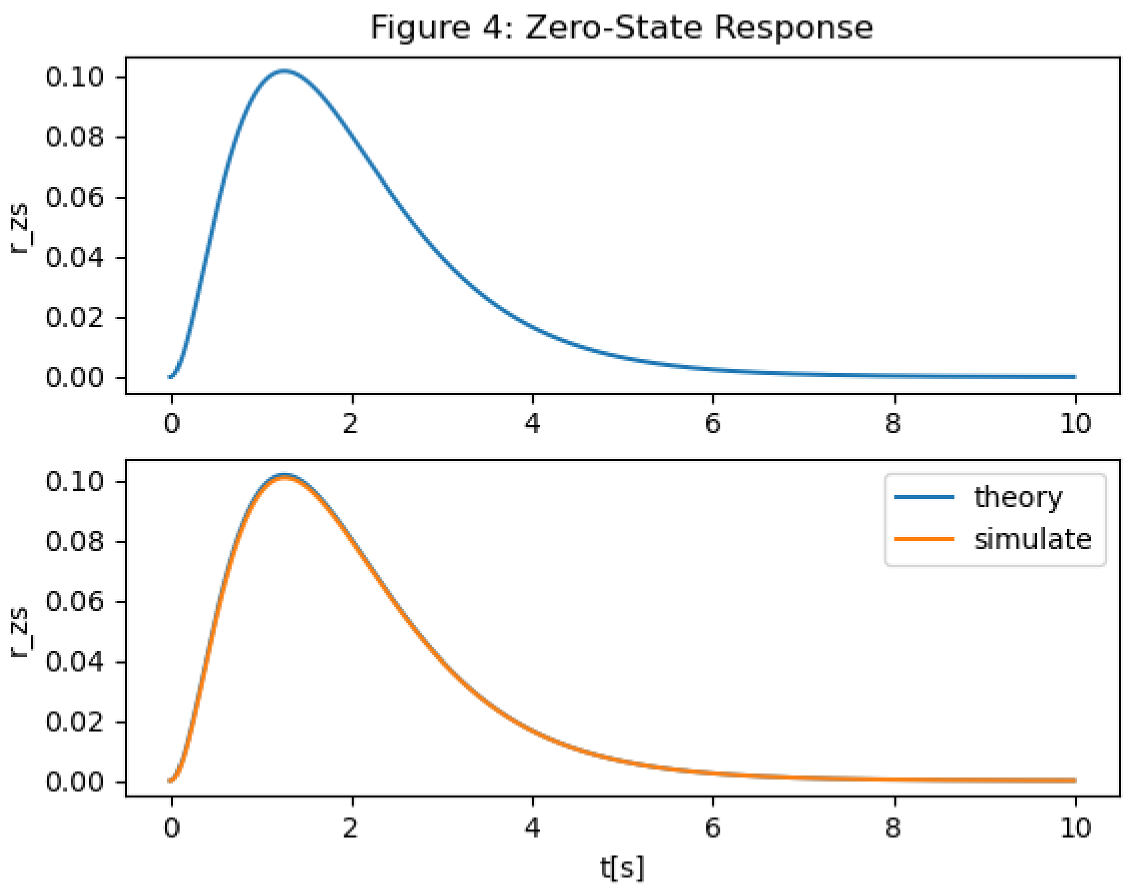
10 / 10

（2）利用MATLAB内置的函数lsim得到零状态响应。

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.signal as sgn


def draw_zs():
    # 线性时不变系统
    # $$
    # r^{\prime \prime}(t)+3 r^{\prime}(t)+2 r(t)=e(t)
    # $$
    system = sgn.lti([1], [1, 3, 2])
    # 自变量取值
    t = np.arange(0, 10, 0.01)
    # f(t)
    f = np.exp(-2 * t) * np.heaviside(t, 0)
    # 理论值
    y_theory = np.exp(-t) - (1 + t) * np.exp(-2 * t)
    plt.subplot(2, 1, 1)  # 指定图像位置
```

```
    plt.plot(t, y_theory)  # 作图理论值
    plt.ylabel("r_zs")
    plt.title('Figure 4: Zero-State Response')
    # 用lsim函数获取系统零状态响应
    tout, yout, xout = sgn.lsim(system, f, t)
    plt.subplot(2, 1, 2)
    plt.plot(t, y_theory, label='theory')
    plt.plot(tout, yout, label='simulate')  # 仿真结果图
    plt.xlabel("t[s]")
    plt.ylabel("r_zs")
    plt.legend()
    plt.savefig("./img/Figure 4.png")
    plt.show()


draw_zs()
```

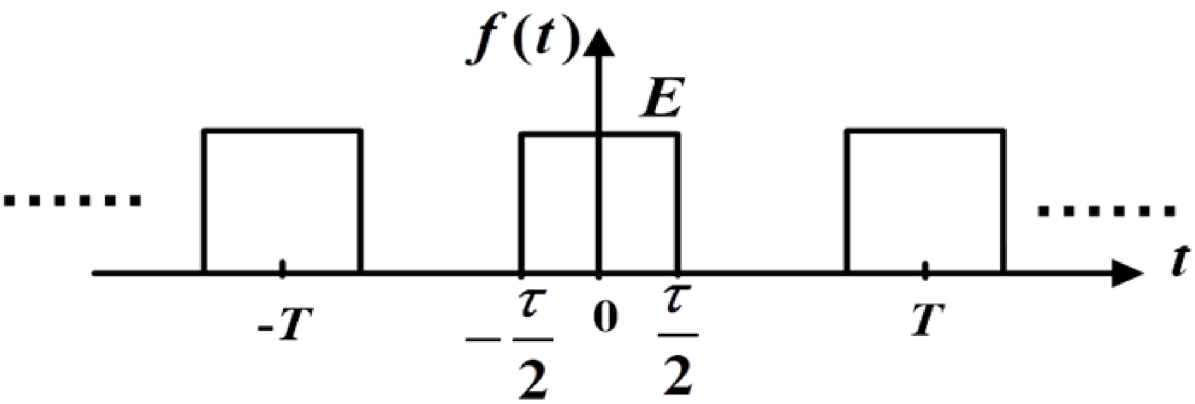
（3）比较第一步得到的理论值与第二步得到的数值解是否一致。



由图像可知，python仿真值与理论值一致。

5. 周期信号的傅里叶级数展开

定义一个周期信号$f(t)$为矩形脉冲序列，如下图所示，设定
$E = 2, \tau = 1, T = 2$



其三角函数/正余弦傅里叶级数展开式为：

$$f(t) = a_0 + \sum^{\infty} (a_n \cos n\Omega t + b_n \sin n\Omega t) \qquad (1)$$

$$n=1$$

其中，$\Omega = \frac{2\pi}{T}$

$$\begin{cases} a_0 = \frac{1}{T}\int_{t_0}^{t_0+T} f(t)dt \\ a_n = \frac{2}{T}\int_{t_0}^{t_0+T} f(t)\cos n\Omega t dt \\ b_n = \frac{2}{T}\int_{t_0}^{t_0+T} f(t)\sin n\Omega t dt \end{cases} \tag{2}$$

其指数傅里叶级数展开式为:

$$f(t) = \sum_{n=-\infty}^{\infty} F_n e^{jn\Omega t} \tag{3}$$

其中，

$$F_n = \frac{1}{T}\int_{t_0}^{t_0+T} f(t)e^{-jn\Omega t}dt \tag{4}$$

(1) 利用三角函数/正余弦正交函数集合，对周期信号$f(t)$进行三角傅里叶级数展开，写出其三角傅里叶级数表达式。

(1) 利用三角函数求傅里叶展开

$$a_0 = \frac{1}{T}\int_{-\frac{T}{2}}^{\frac{T}{2}} f(t)\,dt = \frac{1}{2}\int_{-1}^{1} f(t)\,dt = \frac{1}{2}\int_{-\frac{1}{2}}^{\frac{1}{2}} 2\,dt = 1 \quad ①$$

$$b_n = \frac{2}{T}\int_{-\frac{T}{2}}^{\frac{T}{2}} f(t)\sin n\Omega t\,dt = 0 \quad ③$$

$$\begin{aligned} a_n &= \frac{2}{T}\int_{-\frac{T}{2}}^{\frac{T}{2}} f(t)\cos n\Omega t\,dt \\ &= \int_{-1}^{1} f(t)\cos n\Omega t\,dt \\ &= 2\int_{-\frac{1}{2}}^{\frac{1}{2}} \cos n\Omega t\,dt \\ &= \frac{4}{n\Omega}\sin\frac{n\Omega}{2} \quad ② \end{aligned}$$

取$\Omega = \frac{2\pi}{T} = \pi$得 $a_n = \frac{4}{n\pi}\sin\frac{n\pi}{2}$ ④

综上所述，该周期信号利用三角函数的傅里叶展开为:

$$f(t) = 1 + \sum_{n=1}^{\infty} \frac{4}{n\pi}\sin\left(\frac{n\pi}{2}\right)\cos(n\pi t)$$

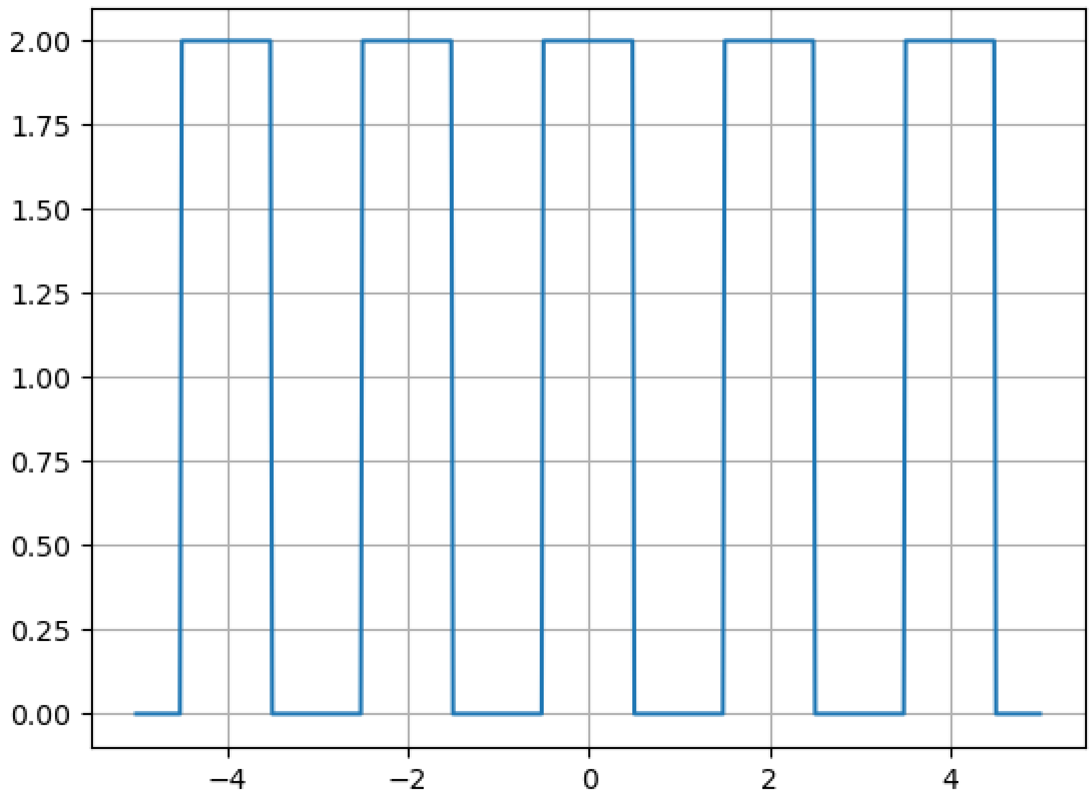
$$f(t) = 1 + \frac{4}{\pi}\cos(\pi t) - \frac{4}{3\pi}\cos(3\pi t) + \frac{4}{5\pi}\cos(5\pi t) - \frac{4}{7\pi}\cos(7\pi t) + \cdots$$

(2) 利用Python画出其三角傅里叶级数展开表达式中的前3项之和(每项系数不为0)，画出其前5项之和(每项系数不为0)，画出其前20项之和(每项系数不为0), 观察它们近似原信号的程度。

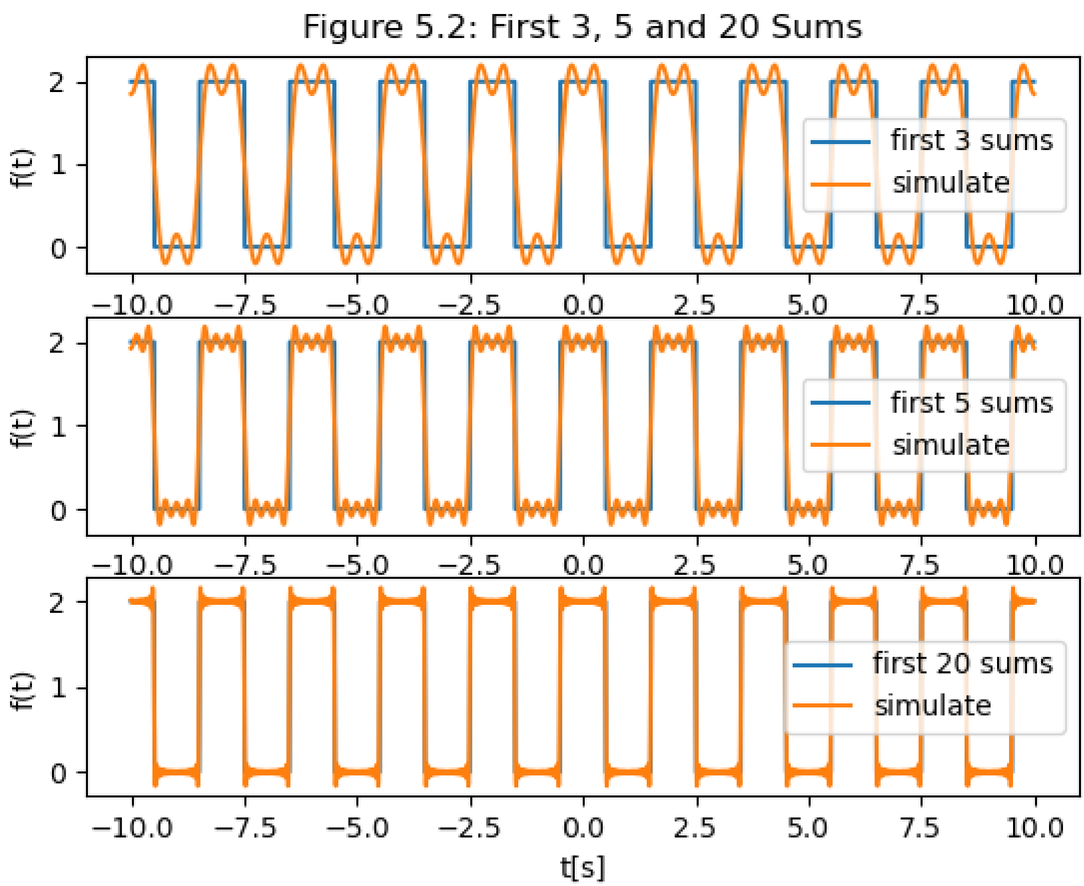
```
# 周期信号的傅里叶级数展开，三角形式展开
# 导包
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal

# 前n项和
def first_n_sums(n: int, t: np.ndarray) -> np.array:
    sum = np.full_like(t, 1)
    if n == 1:
        return sum
    elif n > 1:
        flag: int = 1
        for index in range(1, n):
            sum += 4.0 * np.cos((2 * index - 1) * np.pi * t) / (flag * np.pi * (2 * index - 1))
            flag = -flag
        return sum
    else:
        print("[ERROR], invlid input")

# 原函数
def origin():
    t = np.linspace(-5, 5, 500, endpoint=False)
    y = 1.0 + signal.square(2 * np.pi * (t + 0.5) / 2)
    plt.plot(t, y)
    plt.grid()
    plt.savefig("./img/fourier_origin.png")

# 傅里叶级数展开与原函数的比较
def simulate():
    t = np.arange(-10, 10, 0.01)
    # 原函数
```

```
    y = 1.0 + signal.square(2 * np.pi * (t + 0.5) / 2)
    # 前三项和
    y3 = first_n_sums(3, t)
    # 前五项和
    y5 = first_n_sums(5, t)
    # 前二十项和
    y20 = first_n_sums(20, t)
    # 绘图
    plt.subplot(3, 1, 1)
    plt.plot(t, y, t, y3)
    plt.ylabel("f(t)")
    plt.legend(['first 3 sums', 'simulate'])
    plt.title("Figure 5.2: First 3, 5 and 20 Sums")
    plt.subplot(3, 1, 2)
    plt.plot(t, y, t, y5)
    plt.xlabel("t[s]")
    plt.ylabel("f(t)")
    plt.legend(['first 5 sums', 'simulate'])
    plt.subplot(3, 1, 3)
    plt.plot(t, y, t, y20)
    plt.xlabel("t[s]")
    plt.ylabel("f(t)")
    plt.legend(['first 20 sums', 'simulate'])
    plt.savefig("./img/Figure 5.png")
    plt.show()

origin() # 绘制原图
simulate() # 绘制傅里叶级数展开与原函数的比较
```

原图:



比较傅里叶级数展开结果与原函数:



我们可以看到，傅里叶级数展开的项数越多，与原信号更相近。

(3) 利用虚指数正交函数集合，对周期信号$f(t)$进行指数傅里叶级数展开，写出其指数傅里叶级数表达式。

(3) 利用虚指数正交函数集合，对周期信号 $f(t)$ 进行指数傅里叶级数展开。

$$F_n = \frac{1}{T}\int_{-\frac{T}{2}}^{\frac{T}{2}} f(t)\, e^{-jn\Omega t}\, dt$$

① $n=0$

$$F_0 = \frac{1}{T}\int_{-\frac{T}{2}}^{\frac{T}{2}} f(t)\, dt = \frac{1}{T}\int_{-\frac{\tau}{2}}^{\frac{\tau}{2}} E\, dt = \frac{1}{2}\cdot 2 = 1$$

② $n \neq 0$

$$F_n = \frac{1}{T}\int_{-\frac{T}{2}}^{\frac{T}{2}} f(t)\, e^{-jn\Omega t}\, dt$$

$$= \frac{1}{T}\int_{-\frac{\tau}{2}}^{\frac{\tau}{2}} E\cdot e^{-jn\Omega t}\, dt$$

$$= \frac{E}{Tjn\Omega}\cdot\left(-e^{-jn\Omega t}\right)\Big|_{-\frac{\tau}{2}}^{\frac{\tau}{2}} = \frac{E}{jn\Omega T}\left(e^{\frac{jn\Omega\tau}{2}} - e^{-\frac{jn\Omega\tau}{2}}\right)$$

$$= \frac{1}{jn\pi}\left(e^{j\frac{n\pi}{2}} - e^{-j\frac{n\pi}{2}}\right) = \frac{2}{n\pi}\sin\frac{n\pi}{2}$$

得

$$f(t) = \sum_{n=-\infty}^{+\infty} F_n(t)\cdot e^{jn\Omega t} = F_0 + \sum_{n=1}^{+\infty} F_n(t)\, e^{jn\Omega t} + \sum_{n=1}^{\infty} F_{-n}(t)\, e^{-jn\Omega t}$$

$$= 1 + \sum_{n=1}^{+\infty}\frac{2}{n\pi}\sin\frac{n\pi}{2}\, e^{jn\pi t} + \sum_{n=1}^{+\infty}\frac{2}{n\pi}\sin\frac{n\pi}{2}\, e^{-jn\pi t}$$

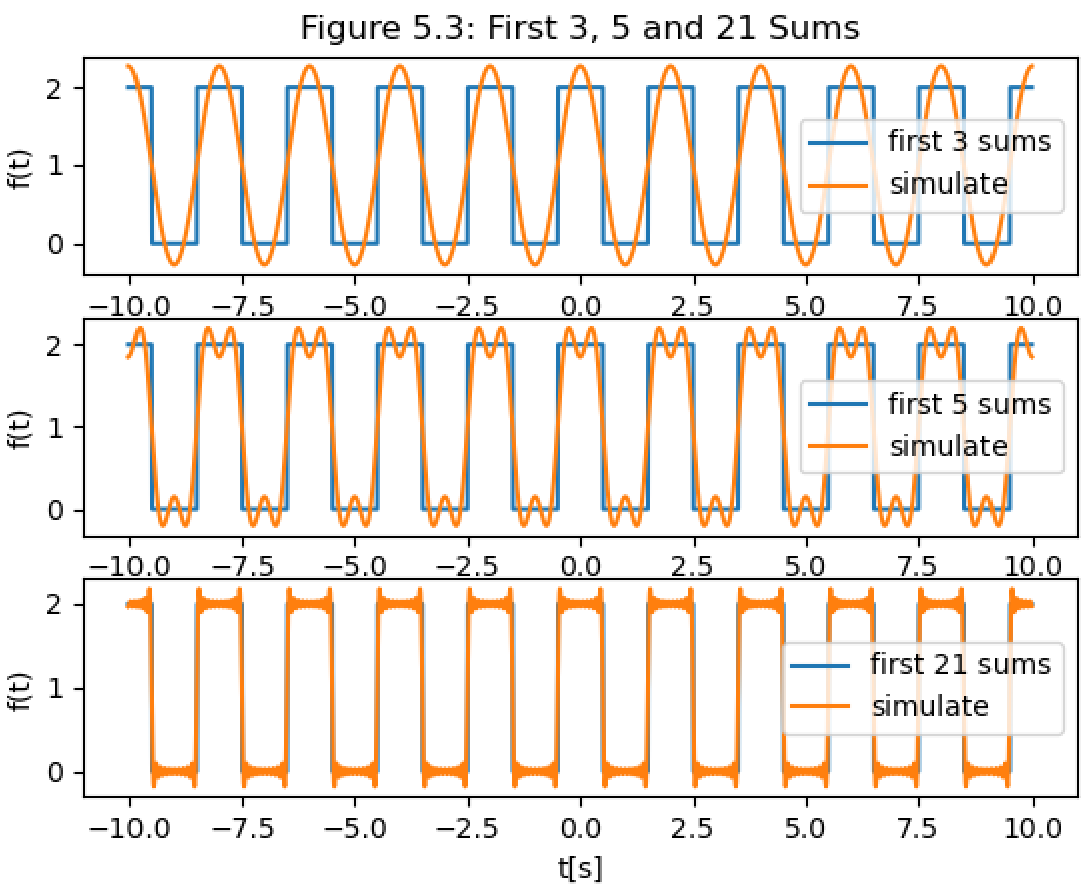
$$= 1 + \frac{2}{\pi}e^{j\pi t} + \frac{2}{\pi}e^{-j\pi t} - \frac{2}{3\pi}e^{3j\pi t} - \frac{2}{3\pi}e^{-3j\pi t} + \frac{2}{5\pi}e^{5j\pi t} + \frac{2}{5\pi}e^{-5j\pi t} + \cdots$$

(4) 利用MATLAB画出其指数傅里叶级数展开表达式中的前3项之和(即$n = \{-1, 0, 1\}$)，并画出其前5项之和(即$n = \{-2, -1, 0, 1, 2\}$)，画出其前21项之和(即$n = \{-10, -9, \cdots, 0, 1, 2, \cdots, 10\}$), 观察它们近似原信号的程度。

```python
# 周期信号的傅里叶级数展开，指数形式展开
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal


# 前n项和
def first_n_sums(n: int, t: np.ndarray) -> np.array:
    sum: np.ndarray = np.ones(np.size(t), dtype=complex) # 将sum设置为complex类型
```

```python
    if n == 1:
        return sum
    elif n > 1:
        n = int(n / 2) + 1
        flag: int = 1
        for index in range(1, n):
            sum += flag * 2.0 * (np.exp(1j * (2 * index - 1) * np.pi * t) +
                                 np.exp(-1j * (2 * index - 1) * np.pi * t)) / (np.pi * (2 * index - 1))
            flag = -flag
        return sum
    else:
        print("[ERROR], invlid input")


# 傅里叶级数展开与原函数的比较
def simulate_complex():
    t = np.arange(-10, 10, 0.01)
    # 原函数
    y = 1.0 + signal.square(2 * np.pi * (t + 0.5) / 2)
    # 前三项和
    y3 = first_n_sums(3, t)
    # 前五项和
    y5 = first_n_sums(5, t)
    # 前二十一项和
    y21 = first_n_sums(21, t)
    # 绘图
    plt.subplot(3, 1, 1)
    plt.plot(t, y, t, y3.real)
    plt.ylabel("f(t)")
    plt.legend(['first 3 sums', 'simulate'])
    plt.title("Figure 5.3: First 3, 5 and 21 Sums")
    plt.subplot(3, 1, 2)
    plt.plot(t, y, t, y5.real)
    plt.xlabel("t[s]")
    plt.ylabel("f(t)")
```

```python
    plt.legend(['first 5 sums', 'simulate'])
    plt.subplot(3, 1, 3)
    plt.plot(t, y, t, y21.real)
    plt.xlabel("t[s]")
    plt.ylabel("f(t)")
    plt.legend(['first 21 sums', 'simulate'])
    plt.savefig("./img/Figure 5.3.png")
    plt.show()


simulate_complex()

```



我们可以看到，傅里叶级数展开的项数越多，与原信号更相近。