

# 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。

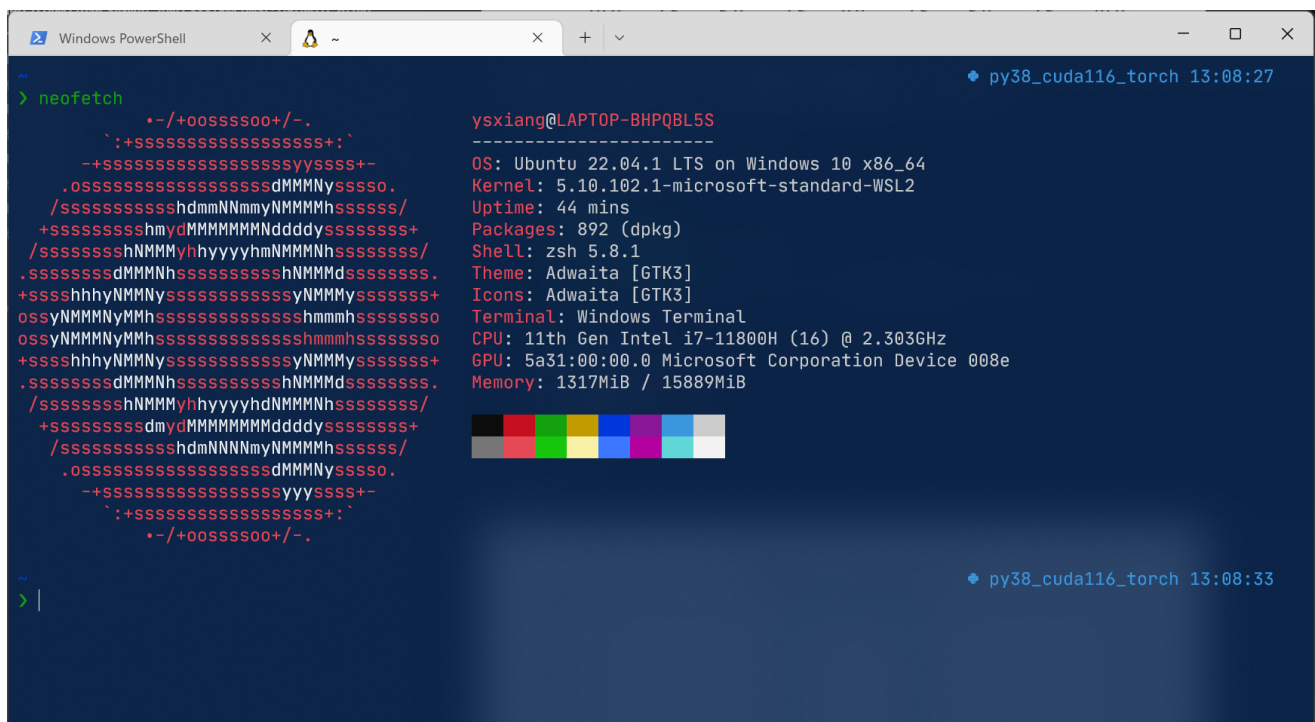
(第一课堂交给施锐，邮箱：[296206140@qq.com](mailto:296206140@qq.com);

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

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



The screenshot shows a Windows PowerShell terminal window with a dark blue background. The title bar indicates it's a Windows PowerShell window. The terminal output includes a neofetch command that displays system information in a stylized ASCII art format. The system information shown is: OS: Ubuntu 22.04.1 LTS on Windows 10 x86\_64, Kernel: 5.10.102.1-microsoft-standard-WSL2, Uptime: 44 mins, Packages: 892 (dpkg), Shell: zsh 5.8.1, Theme: Adwaita [GTK3], Icons: Adwaita [GTK3], Terminal: Windows Terminal, CPU: 11th Gen Intel i7-11800H (16) @ 2.303GHz, GPU: 5a31:00:00.0 Microsoft Corporation Device 008e, Memory: 1317MiB / 15889MiB. The terminal also shows a prompt for 'ysxiang@LAPTOP-BHPQBL5S' and a status bar at the bottom right indicating 'py38\_cuda116\_torch 13:08:27' and 'py38\_cuda116\_torch 13:08:33'.

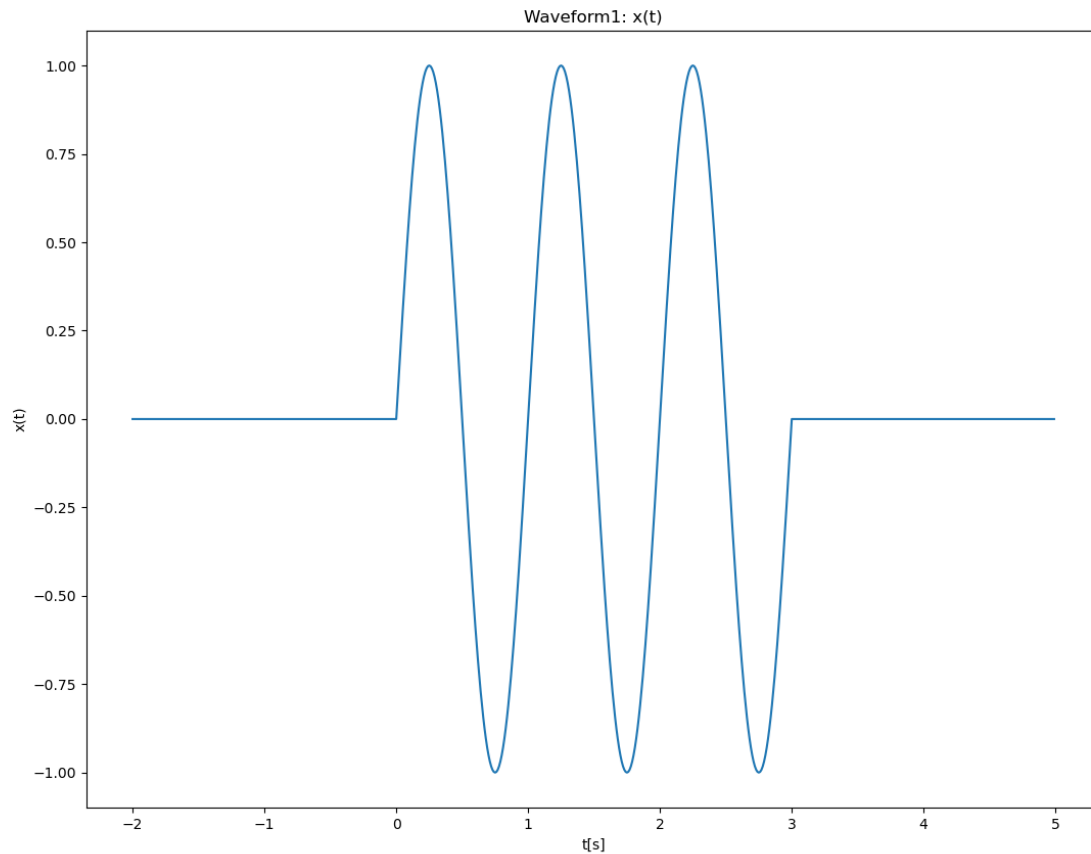
解释器：Anaconda(python=3.8)

## 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

```
1  # 导包
2  import numpy as np
3  import matplotlib.pyplot as plt
4  import scipy.signal as sgn
5
6  #  $x(t) = \sin(2\pi t)[\varepsilon(t) - \varepsilon(t - 3)]$ 
7  def draw_waveform1():
8      # 规定t的范围为-2到5, 梯度0.01递增
9      t = np.arange(-2, 5, 0.01)
10     # x(t) 函数, 使用numpy的heaviside函数作为阶跃函数
11     x = np.sin(2 * np.pi * t) * (np.heaviside(t, 1) -
np.heaviside(t - 3, 1))
12
13     #作图。横轴为t, 纵轴为x
14     plt.plot(t, x)
15     #以下三句声明横轴和纵轴的标签以及图像
16     plt.xlabel("t[s]")
17     plt.ylabel("x(t)")
18     plt.title("Waveform1: x(t)")
19     #绘制图像
20     plt.show()
21
22 draw_waveform1()
```

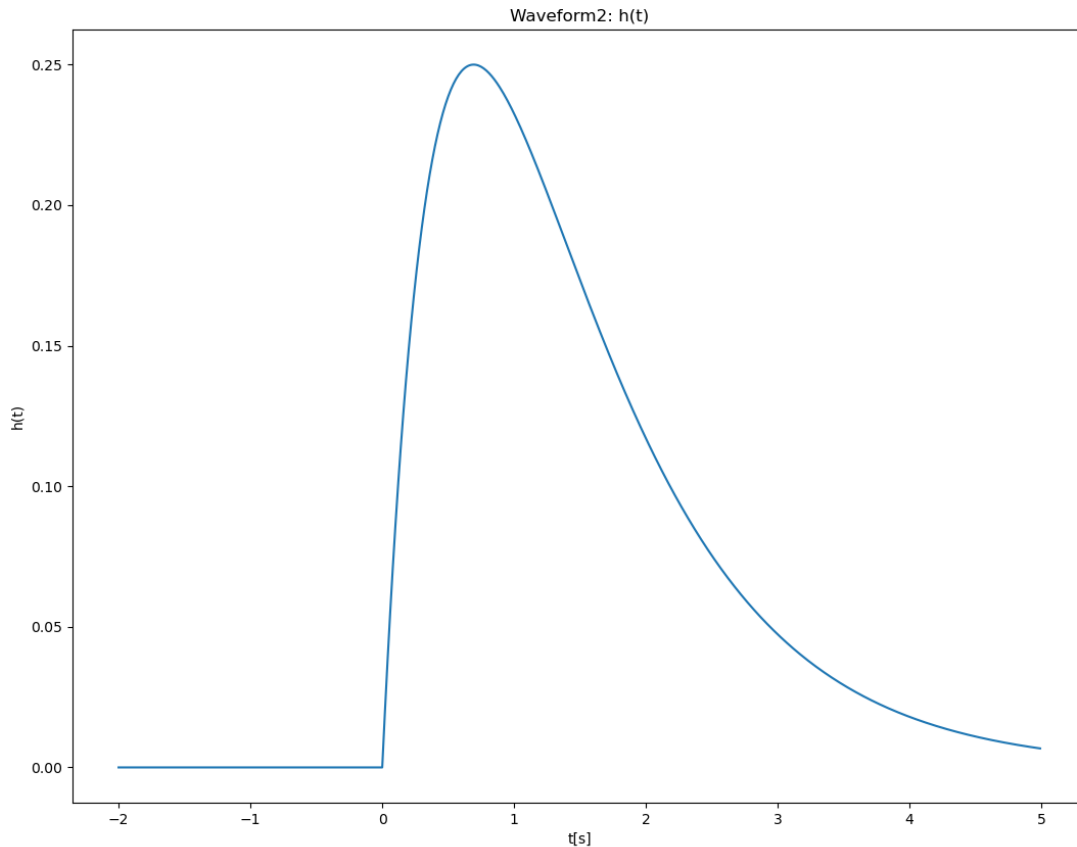


$$(2) h(t) = e^{-t}\varepsilon(t) - e^{-2t}\varepsilon(t)$$

```

1  # $h(t)=e^{-t} \backslash varepsilon(t)-e^{-2 t}$
   \varepsilon(t)$
2  def draw_waveform2():
3      t = np.arange(-2, 5, 0.01)
4      h = np.exp(-t) * np.heaviside(t, 1) - np.exp(-2 *
5      t) * np.heaviside(t, 1)
6
7      plt.plot(t, h)
8      plt.xlabel("t[s]")
9      plt.ylabel("h(t)")
10     plt.title("Waveform2: h(t)")
11     plt.show()
12 draw_waveform2()

```



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

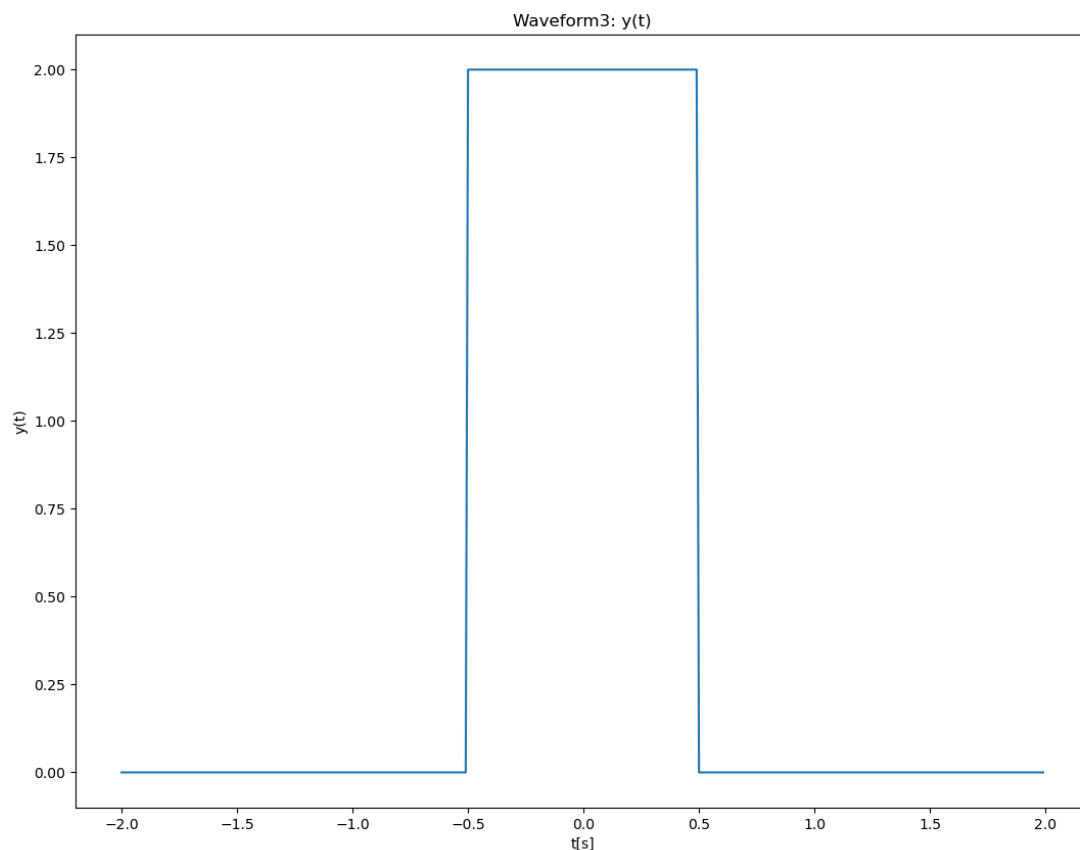
```

1  # 门函数
2  def gate_function(variable: np.ndarray) ->
    np.ndarray:
3      return np.heaviside(variable + 0.5, 1) -
        np.heaviside(variable - 0.5, 1)
4
5
6  # $y(t)=2 G_{2}(t)$
7  def draw_waveform3():
8      t = np.arange(-2, 2, 0.01)
9      y = 2 * gate_function(t)
10
11     plt.plot(t, y)
12     plt.xlabel("t[s]")
13     plt.ylabel("y(t)")
14     plt.title("Waveform3: y(t)")
15     plt.show()

```

16

17 draw\_waveform3()

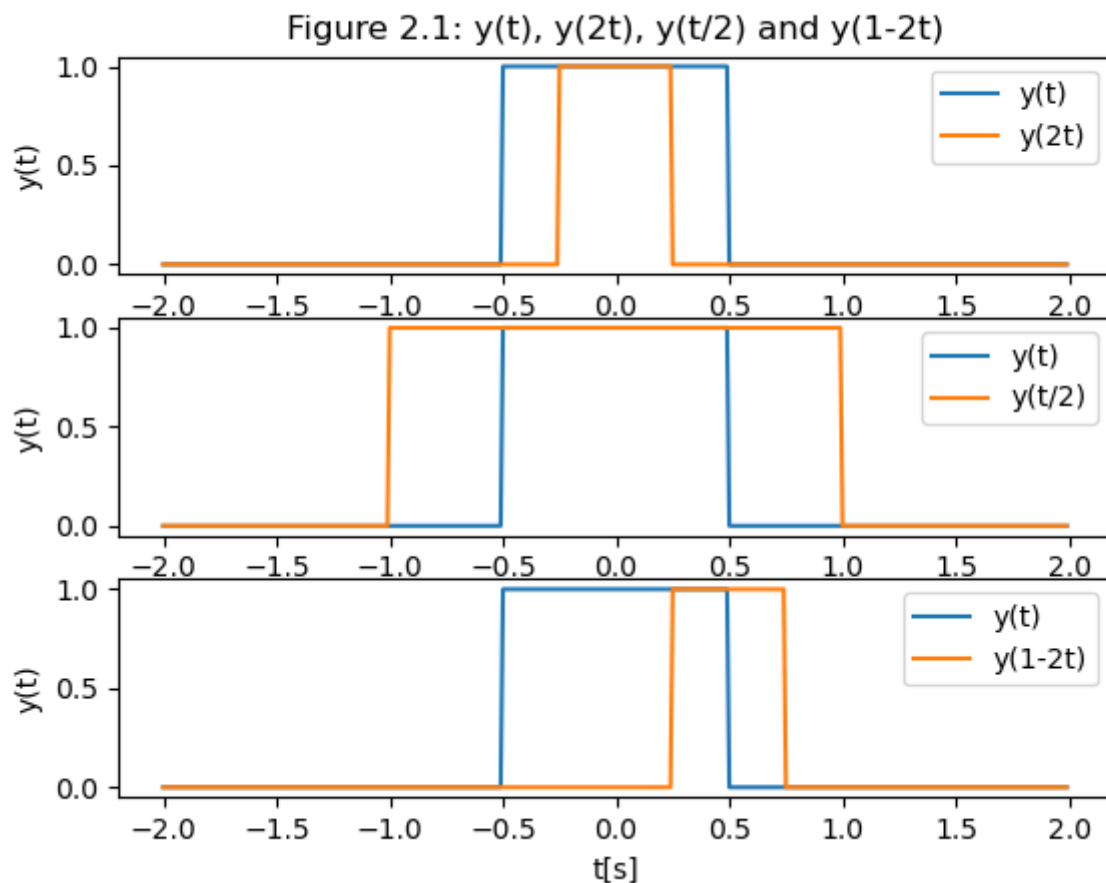


## 2. 利用Python验证信号的基本运算

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

```
1 # 验证信号的基本运算
2 # 导包
3 import numpy as np
4 import matplotlib.pyplot as plt
5
6 # 门函数
7 def gate_function(variable: np.ndarray) ->
  np.ndarray:
8     return np.heaviside(variable + 0.5, 1) -
  np.heaviside(variable - 0.5, 1)
9
```

```
10 def verify1():
11     t = np.arange(-2, 2, 0.01)
12     # G(t)
13     gate = gate_function(t)
14     # y(2 t)
15     y1 = gate_function(2 * t)
16     # y(t / 2)
17     y2 = gate_function(t / 2.0)
18     # y(1 - 2t)
19     y3 = gate_function(1 - 2 * t)
20     # 绘制第一幅图像
21     plt.subplot(3, 1, 1)
22     plt.plot(t, gate, t, y1)
23     plt.xlabel("t[s]")
24     plt.ylabel("y(t)")
25     plt.legend(['y(t)', 'y(2t)'])
26     plt.title("Figure 2.1: y(t), y(2t), y(t/2) and
27 y(1-2t)")
28     # 第二幅图像
29     plt.subplot(3, 1, 2)
30     plt.plot(t, gate, t, y2)
31     plt.xlabel("t[s]")
32     plt.ylabel("y(t)")
33     plt.legend(['y(t)', 'y(t/2)'])
34     # 第三幅图像
35     plt.subplot(3, 1, 3)
36     plt.plot(t, gate, t, y3)
37     plt.xlabel("t[s]")
38     plt.ylabel("y(t)")
39     plt.legend(['y(t)', 'y(1-2t)'])
40     # 绘图
41     plt.savefig("Figure 2.1.png")
42     plt.show()
43 verify1()
```



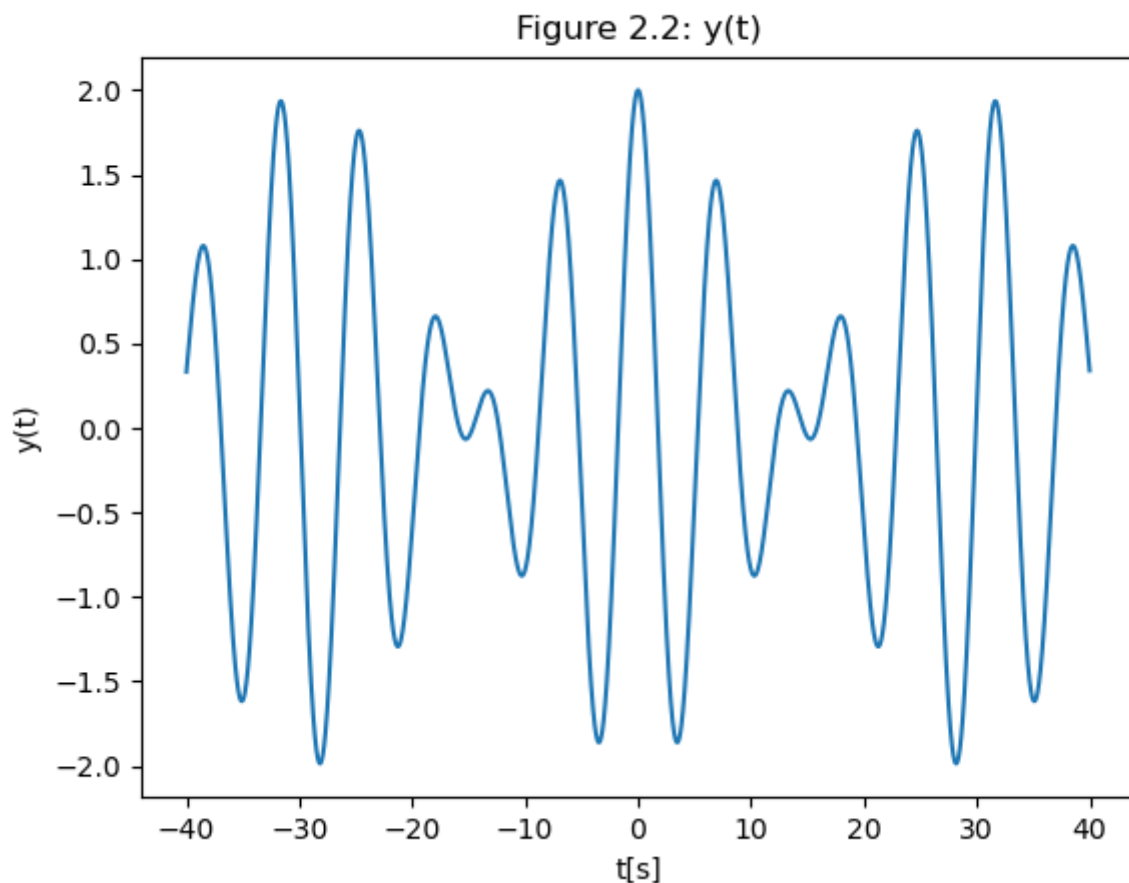
(2) 画出  $\cos(t) + \cos\left(\frac{\pi}{4}t\right)$ ，并观察其是否为周期函数，如果是，周期为多少？

```

1  #  $\cos(t) + \cos\left(\frac{\pi}{4}t\right)$ 
2  def verify2():
3      t = np.arange(-40, 40, 0.01)
4      y = np.cos(t) + np.cos(np.pi * t / 4.0)
5
6      plt.plot(t, y)
7      plt.xlabel("t[s]")
8      plt.ylabel("y(t)")
9      plt.title("Figure 2.2: y(t)")
10     plt.savefig("../lab1/img/Figure 2.2.png")
11     plt.show()
12
13 verify2()

```





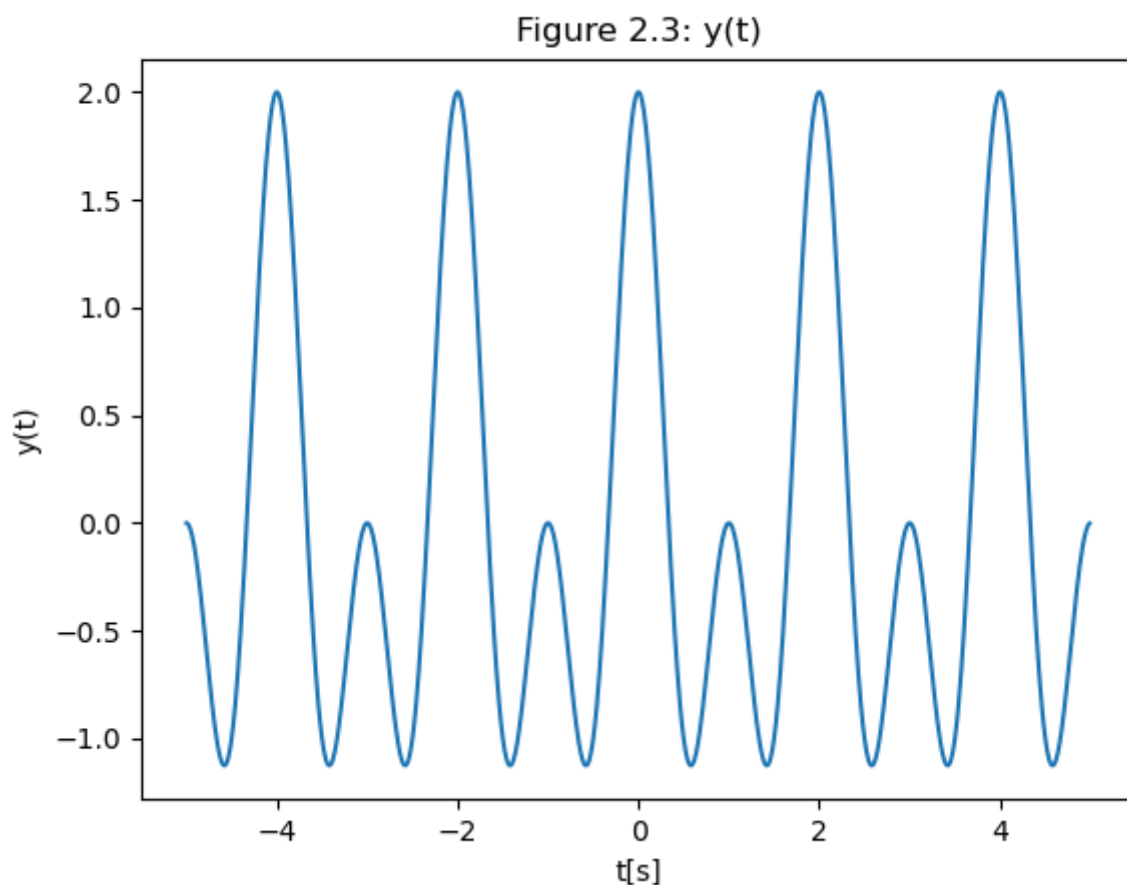
观察得  $\cos(t) + \cos\left(\frac{\pi}{4}t\right)$  不是周期函数

(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？

```

1  # $\cos(\pi t) + \cos(2\pi t)$
2  def verify3():
3      t = np.arange(-5, 5, 0.01)
4      y = np.cos(np.pi * t) + np.cos(2 * np.pi * t)
5
6      plt.plot(t, y)
7      plt.xlabel("t[s]")
8      plt.ylabel("y(t)")
9      plt.title("Figure 2.3: y(t)")
10     plt.savefig("../lab1/img/Figure 2.3.png")
11     plt.show()
12
13 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  $x(t) = [e^{-2t}\varepsilon(t)] * [e^{-3t}\varepsilon(t)]$

(1) 根据卷积的定义，推导得到  $x(t)$  的理论值；

$$\begin{aligned}
 x(t) &= [e^{-2t} \varepsilon(t)] * [e^{-3t} \varepsilon(t)] \\
 &= \int_{-\infty}^{+\infty} e^{-2\tau} \varepsilon(\tau) \cdot e^{-3(t-\tau)} \varepsilon(t-\tau) d\tau \\
 &= \int_0^t e^{\tau-3t} \varepsilon(\tau) \varepsilon(t-\tau) d\tau \\
 &= \left. e^{\tau-3t} \right|_0^t \\
 &= e^{-2t} - e^{-3t}, t > 0
 \end{aligned}$$

(2)利用MATLAB的conv函数获得 $x(t)$ 的数值

```

1 # 卷积运算
2 # $x(t)=\left[e^{-2 t} \backslash \text { varepsilon}(t) \backslash \text { right}\right]$
   * $\left[e^{-3 t} \backslash \text { varepsilon}(t) \backslash \text { right}\right]$
3 # 导包
4 import numpy as np
5 import matplotlib.pyplot as plt
6 import scipy.signal as sgn
7
8 STEP_SIZE = 0.01
9
10
11 def simulate_theory():
12     # 绘制模拟数值图像
13     t = np.arange(0, 20, STEP_SIZE)
14     y1 = np.exp(-2 * t) * np.heaviside(t, 0)
15     y2 = np.exp(-3 * t) * np.heaviside(t, 0)
16     # 将y1和y2进行卷积。
17     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。
18     plt.subplot(2, 1, 1)

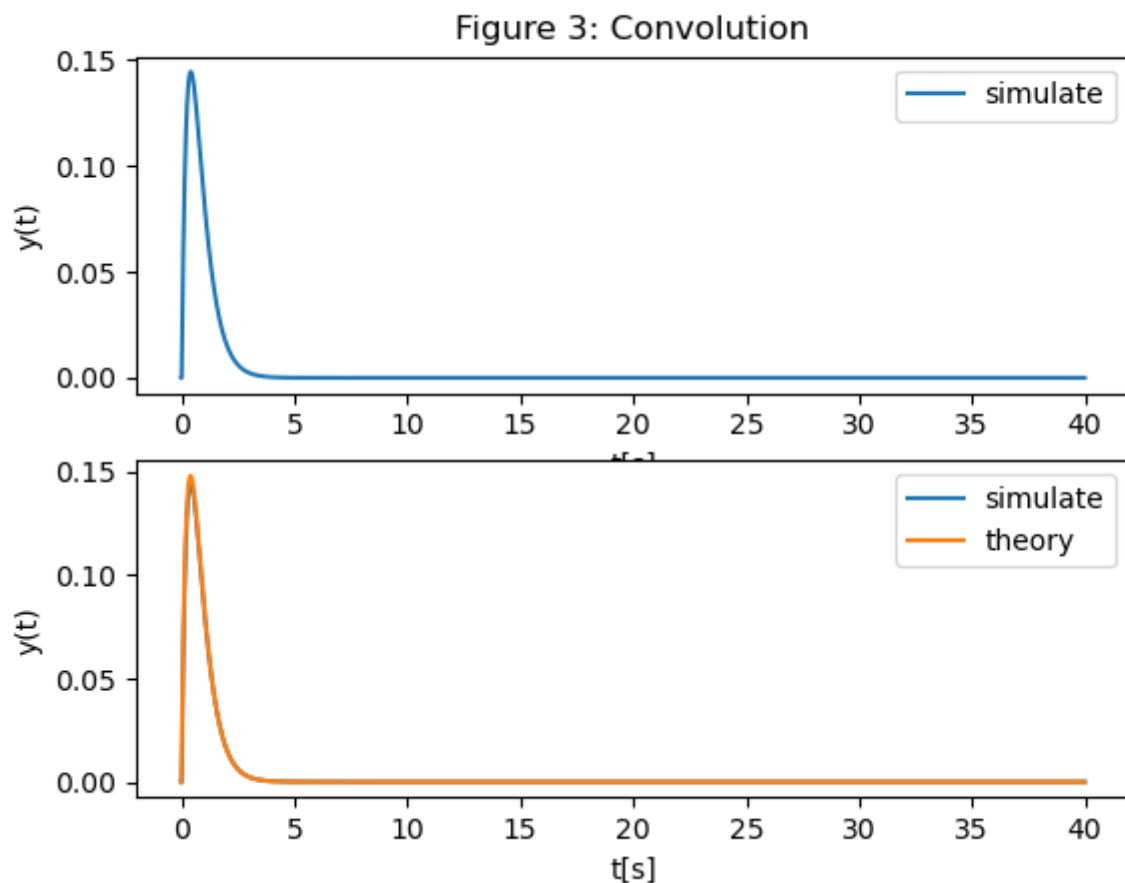
```

```

19     plt.plot(np.arange(0, 39.99, STEP_SIZE), y,
label='simulate')
20     plt.xlabel("t[s]")
21     plt.ylabel("y(t)")
22     plt.legend()
23     plt.title("Figure 3: Convolution")
24     # 绘制理论值图像
25     t_theory = np.arange(0, 39.99, STEP_SIZE)
26     y_theory = (np.exp(-2 * t_theory) - np.exp(-3 *
t_theory)) * np.heaviside(t_theory, 1)
27     plt.subplot(2, 1, 2)
28     plt.plot(t_theory, y, t_theory, y_theory,
label='theory')
29     plt.xlabel("t[s]")
30     plt.ylabel("y(t)")
31     plt.legend(['simulate', 'theory'])
32     plt.savefig("../lab1/img/Figure 3.png")
33     plt.show()
34
35 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 $r_{zs}(t)$ 。

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

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

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:  $r_{zs} = e(t) * h(t)$

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

10/10

(2) 利用MATLAB内置的函数lsim得到零状态响应。

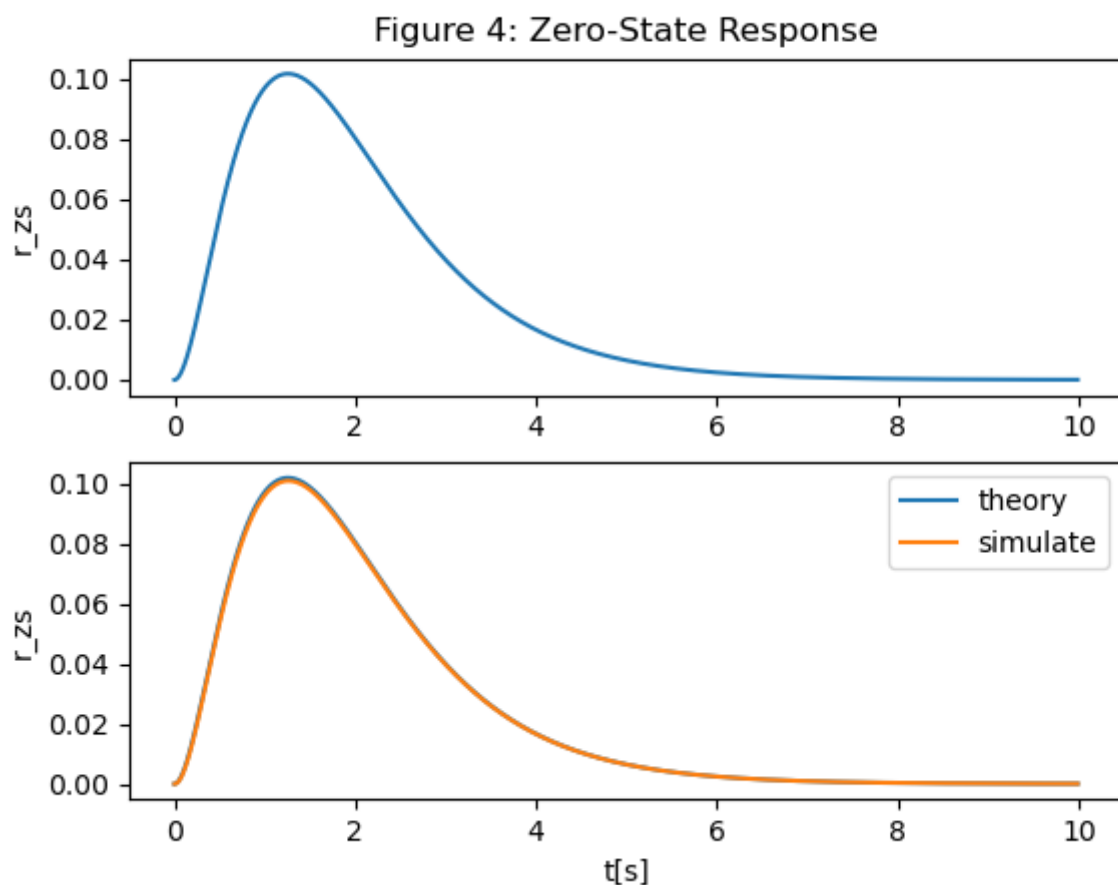
```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 import scipy.signal as sgn
4
5
6 def draw_zs():
7     # 线性时不变系统
8     # $$
9     # r^{\prime \prime}(t)+3 r^{\prime}(t)+2
10    r(t)=e(t)
11    # $$
12    system = sgn.lti([1], [1, 3, 2])
13    # 自变量取值
14    t = np.arange(0, 10, 0.01)
15    # f(t)
16    f = np.exp(-2 * t) * np.heaviside(t, 0)
17    # 理论值
18    y_theory = np.exp(-t) - (1 + t) * np.exp(-2 * t)
19    plt.subplot(2, 1, 1) # 指定图像位置
```

```

19     plt.plot(t, y_theory)    # 作图理论值
20     plt.ylabel("r_zs")
21     plt.title('Figure 4: Zero-State Response')
22     # 用lsim函数获取系统零状态响应
23     tout, yout, xout = sgn.lsim(system, f, t)
24     plt.subplot(2, 1, 2)
25     plt.plot(t, y_theory, label='theory')
26     plt.plot(tout, yout, label='simulate')    # 仿真结果
图
27     plt.xlabel("t[s]")
28     plt.ylabel("r_zs")
29     plt.legend()
30     plt.savefig("./img/Figure 4.png")
31     plt.show()
32
33
34 draw_zs()

```

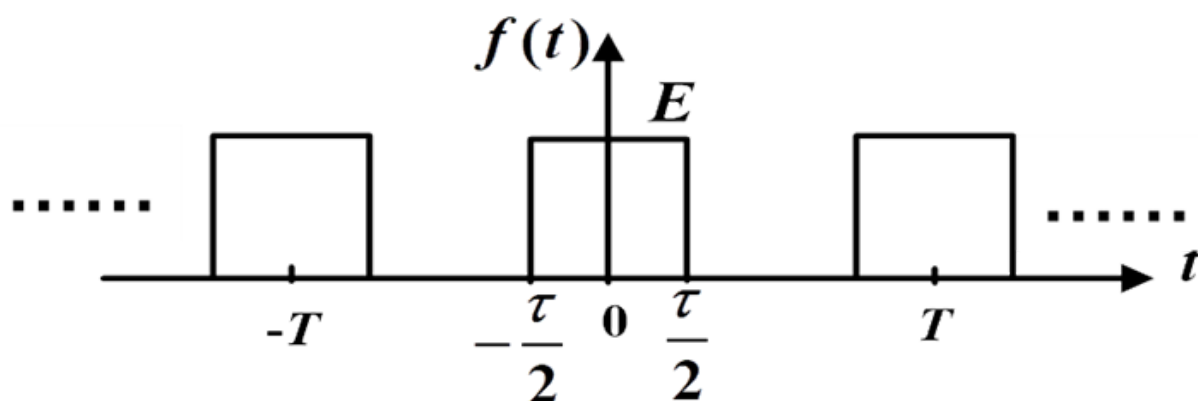
(3) 比较第一步得到的理论值与第二步得到的数值解是否一致。



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：

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



$$\sum_{n=1}^{\infty}$$

其中,  $\Omega = \frac{2\pi}{T}$

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

其指数傅里叶级数展开式为:

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

其中,

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

(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

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

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

$$= \int_{-1}^1 f(t) \cos n\Omega t dt$$

$$= 2 \int_{-\frac{1}{2}}^{\frac{1}{2}} \cos n\Omega t dt$$

$$= \frac{4}{n\Omega} \sin \frac{n\Omega}{2} \quad (3)$$

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

综上所述, 该周期信号利用三角函数的傅里叶展开为:

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

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

(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。

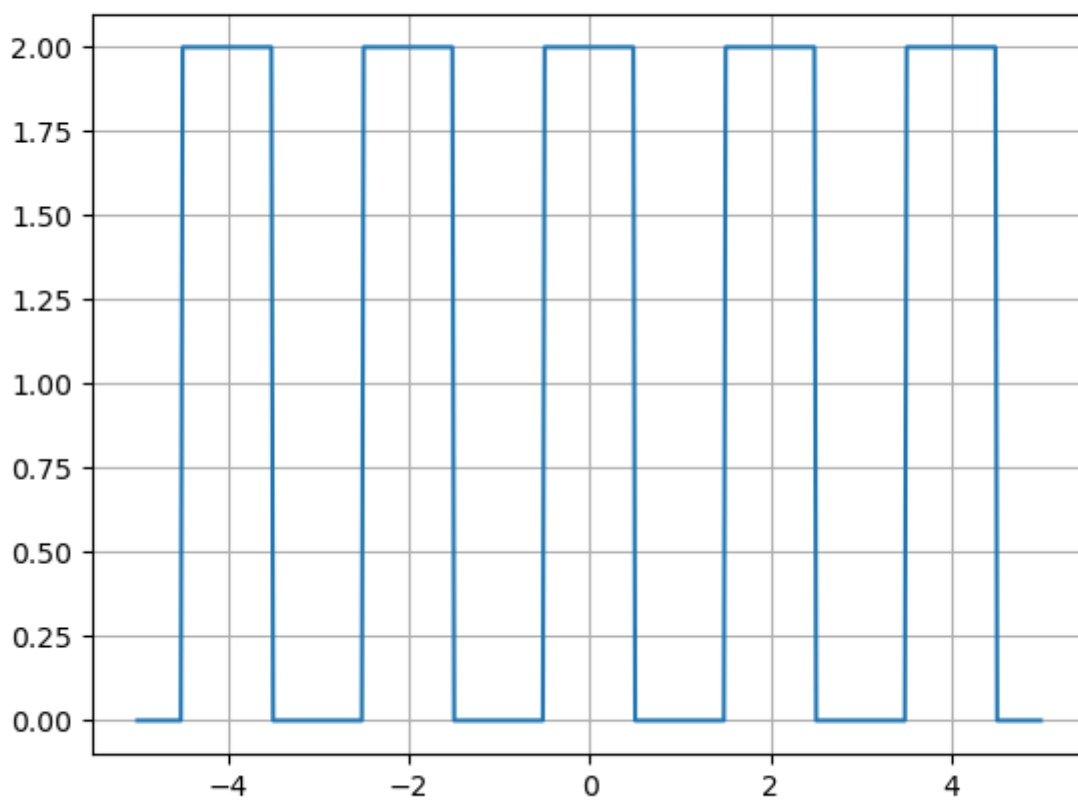
```
1  # 周期信号的傅里叶级数展开, 三角形式展开
2  # 导包
3  import numpy as np
4  import matplotlib.pyplot as plt
5  from scipy import signal
6
7  # 前n项和
8  def first_n_sums(n: int, t: np.ndarray) -> np.array:
9      sum = np.full_like(t, 1)
10     if n == 1:
11         return sum
12     elif n > 1:
13         flag: int = 1
14         for index in range(1, n):
15             sum += 4.0 * np.cos((2 * index - 1) *
np.pi * t) / (flag * np.pi * (2 * index - 1))
16             flag = -flag
17         return sum
18     else:
19         print("[ERROR], invlid input")
20
21 # 原函数
22 def origin():
23     t = np.linspace(-5, 5, 500, endpoint=False)
24     y = 1.0 + signal.square(2 * np.pi * (t + 0.5) /
25     2)
26     plt.plot(t, y)
27     plt.grid()
28     plt.savefig("./img/fourier_origin.png")
29
30 # 傅里叶级数展开与原函数的比较
31 def simulate():
32     t = np.arange(-10, 10, 0.01)
33     # 原函数
```

```

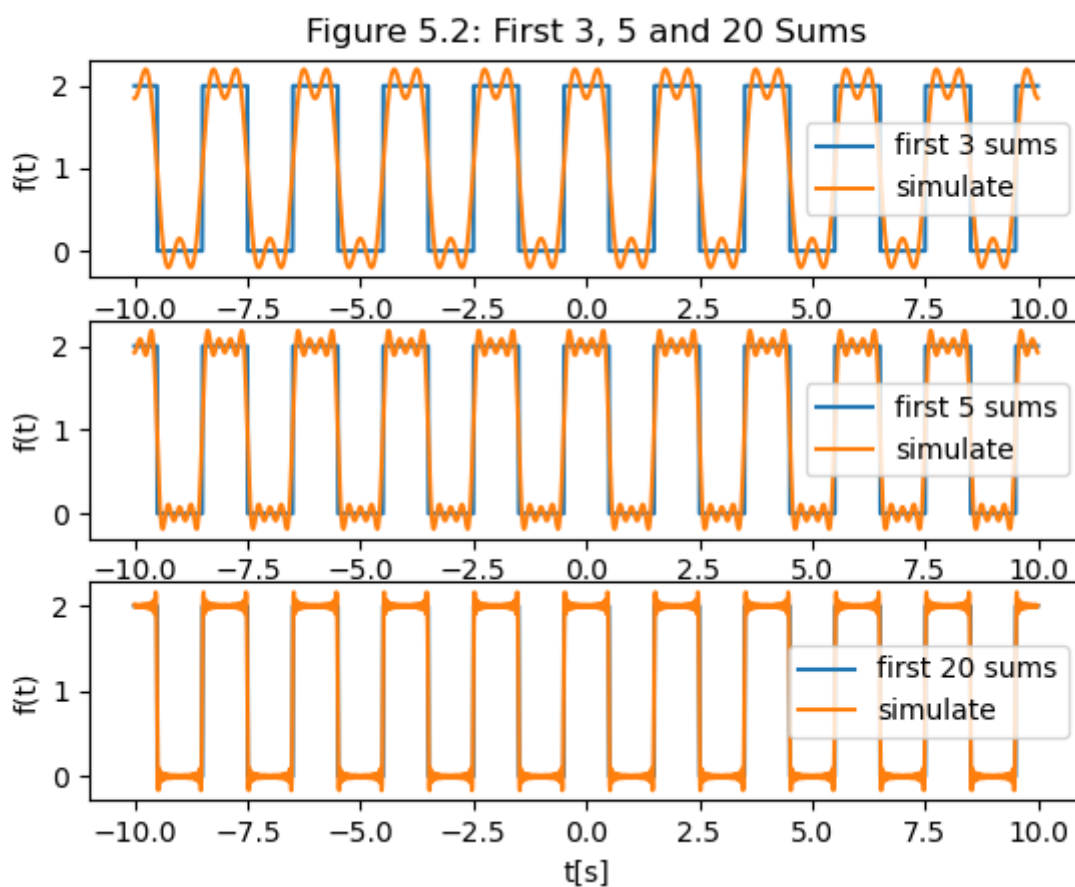
33     y = 1.0 + signal.square(2 * np.pi * (t + 0.5) /
34     2)
35     # 前三项和
36     y3 = first_n_sums(3, t)
37     # 前五项和
38     y5 = first_n_sums(5, t)
39     # 前二十项和
40     y20 = first_n_sums(20, t)
41     # 绘图
42     plt.subplot(3, 1, 1)
43     plt.plot(t, y, t, y3)
44     plt.ylabel("f(t)")
45     plt.legend(['first 3 sums', 'simulate'])
46     plt.title("Figure 5.2: First 3, 5 and 20 Sums")
47     plt.subplot(3, 1, 2)
48     plt.plot(t, y, t, y5)
49     plt.xlabel("t[s]")
50     plt.ylabel("f(t)")
51     plt.legend(['first 5 sums', 'simulate'])
52     plt.subplot(3, 1, 3)
53     plt.plot(t, y, t, y20)
54     plt.xlabel("t[s]")
55     plt.ylabel("f(t)")
56     plt.legend(['first 20 sums', 'simulate'])
57     plt.savefig("./img/Figure 5.png")
58     plt.show()
59 origin() # 绘制原图
60 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。

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

$$\textcircled{1} n=0 \quad F_0 = \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} f(t) dt = \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} E dt = \frac{1}{2} \cdot 2 = 1$$

$$\begin{aligned} \textcircled{2} n \neq 0 \quad F_n &= \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} f(t) e^{-jn\Omega t} dt \\ &= \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} E \cdot e^{-jn\Omega t} dt \\ &= \frac{E}{Tjn\Omega} \cdot (-e^{-jn\Omega t}) \Big|_{-\frac{T}{2}}^{\frac{T}{2}} = \frac{E}{jn\Omega T} (e^{\frac{jn\Omega T}{2}} - e^{-\frac{jn\Omega T}{2}}) \\ &= \frac{1}{jn\pi} (e^{jn\frac{\pi}{2}} - e^{-jn\frac{\pi}{2}}) = \frac{2}{n\pi} \sin \frac{n\pi}{2} \end{aligned}$$

$$\begin{aligned} \text{将 } f(t) &= \sum_{n=-\infty}^{+\infty} F_n(t) \cdot e^{jn\Omega t} = f_0 + \sum_{n=1}^{+\infty} F_n(t) e^{jn\Omega t} + \sum_{n=1}^{\infty} F_{-n}(t) e^{-jn\Omega t} \\ &= 1 + \sum_{n=1}^{+\infty} \frac{2}{n\pi} \sin \frac{n\pi}{2} e^{jn\pi t} + \sum_{n=1}^{+\infty} \frac{2}{n\pi} \sin \frac{n\pi}{2} e^{-jn\pi t} \\ &= 1 + \frac{2}{\pi} e^{j\pi t} + \frac{2}{\pi} e^{-j\pi t} - \frac{2}{3\pi} e^{j3\pi t} - \frac{2}{3\pi} e^{-j3\pi t} + \frac{2}{5\pi} e^{j5\pi t} + \frac{2}{5\pi} e^{-j5\pi t} + \dots \end{aligned}$$

(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  $n = \{-10, -9, \dots, 0, 1, 2, \dots, 10\}$ )，观察它们近似原信号的程度。

```
1 # 周期信号的傅里叶级数展开，指数形式展开
2 import numpy as np
3 import matplotlib.pyplot as plt
4 from scipy import signal
5
6
7 # 前n项和
8 def first_n_sums(n: int, t: np.ndarray) -> np.array:
9     sum: np.ndarray = np.ones(np.size(t),
10 dtype=complex) # 将sum设置为complex类型
```

```

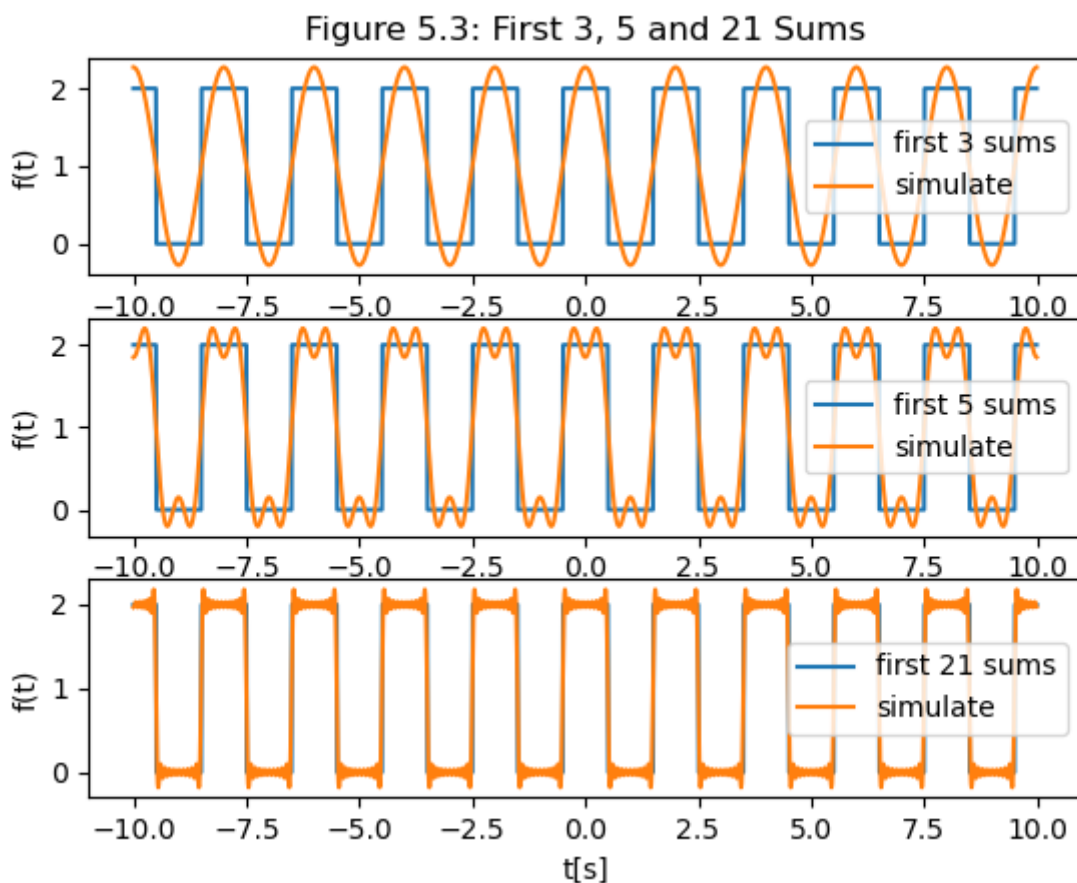
10     if n == 1:
11         return sum
12     elif n > 1:
13         n = int(n / 2) + 1
14         flag: int = 1
15         for index in range(1, n):
16             sum += flag * 2.0 * (np.exp(1j * (2 *
index - 1) * np.pi * t) +
17                                     np.exp(-1j * (2 *
index - 1) * np.pi * t)) / (np.pi * (2 * index - 1))
18             flag = -flag
19         return sum
20     else:
21         print("[ERROR], invlid input")
22
23
24 # 傅里叶级数展开与原函数的比较
25 def simulate_complex():
26     t = np.arange(-10, 10, 0.01)
27     # 原函数
28     y = 1.0 + signal.square(2 * np.pi * (t + 0.5) /
29                               2)
29     # 前三项和
30     y3 = first_n_sums(3, t)
31     # 前五项和
32     y5 = first_n_sums(5, t)
33     # 前二十一项和
34     y21 = first_n_sums(21, t)
35     # 绘图
36     plt.subplot(3, 1, 1)
37     plt.plot(t, y, t, y3.real)
38     plt.ylabel("f(t)")
39     plt.legend(['first 3 sums', 'simulate'])
40     plt.title("Figure 5.3: First 3, 5 and 21 Sums")
41     plt.subplot(3, 1, 2)
42     plt.plot(t, y, t, y5.real)
43     plt.xlabel("t[s]")
44     plt.ylabel("f(t)")

```

```

45 plt.legend(['first 5 sums', 'simulate'])
46 plt.subplot(3, 1, 3)
47 plt.plot(t, y, t, y21.real)
48 plt.xlabel("t[s]")
49 plt.ylabel("f(t)")
50 plt.legend(['first 21 sums', 'simulate'])
51 plt.savefig("./img/Figure 5.3.png")
52 plt.show()
53
54
55 simulate_complex()
56

```



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。