



# 基于快速傅里叶变换（FFT）的 模拟非周期信号频谱近似计算

岳志邦 202100171093 自动 21.2 班

朱骥尧 202100171048 自动 21.2 班

朱梓豪 202100171236 自动 21.2 班

2024 年 6 月 22 日

## 目录

<b>1 题目描述</b>	<b>2</b>
<b>2 设计要求</b>	<b>2</b>
<b>3 实验主旨与过程实现</b>	<b>2</b>
3.1 生成一个模拟非周期信号 . . . . .	2
3.2 FFT 频谱近似计算 . . . . .	2
3.3 原始信号显示 . . . . .	3
3.4 计算信号的频谱近似 . . . . .	4
3.5 误差的来源及 FFT 分辨率对计算结果的影响 . . . . .	5
3.5.1 误差的来源 . . . . .	5
3.5.2 不同长度 FFT . . . . .	5
3.6 优化 FFT 计算 . . . . .	6
<b>4 总程序</b>	<b>7</b>
<b>5 结论与讨论</b>	<b>11</b>
5.1 误差的处理方法 . . . . .	11
5.2 采样频率的选取 . . . . .	12
<b>6 组员分工与签名</b>	<b>14</b>
6.1 组员分工 . . . . .	14
6.2 组员签名 . . . . .	14

## 1 题目描述

利用 **FFT 算法** 对 **模拟非周期信号** 进行频谱近似计算，分析计算结果与理论频谱的差异，并研究 **FFT 分辨率** 对计算结果的影响。

## 2 设计要求

1. 生成一个模拟非周期信号（矩形脉冲信号，或者自选），频率和幅度可自行设定。
2. 设计一个 FFT 频谱近似计算实验，选择合适的采样率和 FFT 长度。
3. 比较计算的频谱结果与理论频谱的差异，分析误差来源。
4. 研究不同 FFT 长度对计算结果的影响，探讨 FFT 分辨率的重要性。

## 3 实验主旨与过程实现

本实验旨在通过快速傅里叶变换（FFT）算法对模拟的非周期信号进行频谱近似计算。通过实验，我们将生成一个矩形脉冲信号作为示例信号，利用自行编写的 FFT 算法计算其频谱，并探讨不同采样率、FFT 长度对频谱计算精度的影响。实验还将分析频谱计算误差的来源及如何优化 FFT 计算以提高准确性。

我们使用 **Python** 语言设计程序。

### 3.1 生成一个模拟非周期信号

选择一个矩形脉冲信号作为模拟非周期信号。信号的幅度、宽度和周期可以根据实验要求进行设定。在本实验中，我们选用矩形脉冲信号，设定了信号的周期为 1 秒，幅度为 1，脉冲宽度为 0.1 秒（在生成信号时，通过额外处理使此信号具有非周期性）。

```
1 # 设置信号参数：
2 amplitude = 1
3 # 脉冲宽度
4 width = 0.1
5 # 信号周期
6 period = 1
```

可知，频率即为 1Hz。

### 3.2 FFT 频谱近似计算

设计 FFT 频谱近似计算实验，设置采样率和 FFT 长度，绘制时域波形图。

为了进行频谱计算，实现了一个基于递归的 FFT 算法，并对输入信号进行了补零操作，以确保信号长度达到 FFT 要求的最小长度。这一步骤在保持频谱形状不变的情况下增加了频谱分辨率。

首先设置采样率：

```
1 # 采样率
2 new_sampling_rate = 120
```

```
3 # 采样点
4 num_samples = int(new_sampling_rate * period)
```

(由于采样率太低会出现**混叠效应**，因此设置在 120，尽管奈奎斯特采样定理要求 2 倍，但实测 100 倍以下可能出现混叠)

生成新的时间轴和信号：

```
1 # 时间轴
2 t = np.linspace(0, period, num_samples, endpoint=False)
3 # 信号
4 signal = amplitude * (t % period < width)
```

虽然信号是有周期属性的（周期是 1），但在生成时间轴时只生成了 1 个周期的长度，因此它在时域上仍然是**非周期的**，可以看作一个非周期性质的矩形脉冲信号。

接下来编写 FFT 计算函数（不使用 Numpy 内置 FFT 变换函数）：

```
1 # 检查是否为2的幂
2 def pad_to_power_of_two(x):
3     N = len(x)
4     if np.log2(N) % 1 > 0:
5         next_power_of_two = 2**int(np.ceil(np.log2(N)))
6         padded_x = np.zeros(next_power_of_two)
7         padded_x[:N] = x
8         return padded_x
9     else:
10        return x
11
12 def fft(x):
13     x = pad_to_power_of_two(x)
14     N = len(x)
15     if N <= 1:
16         return x
17     even = fft(x[0::2])
18     odd = fft(x[1::2])
19
20     # W_nk = exp(-2j * pi * k / N)
21     # T是FFT的奇数项:
22     T = [ np.exp(-2j * np.pi * k / N) * odd[k] for k in range(N // 2)]
23     return [even[k] + T[k] for k in range(N // 2)] + \
           [even[k] - T[k] for k in range(N // 2)]
```

### 3.3 原始信号显示

使用 Matplotlib 创建图板（字体使用 Windows 字体微软雅黑）：

```
1 # 生成2×2的图板
2 fig, axs = plt.subplots(2, 2, figsize=(14, 10))
3 # 选定字体微软雅黑
4 my_font = FontProperties(fname=r"c:\windows\fonts\SimHei.ttf", size=12)
```

在第一个子图显示原始信号时域波形，横轴为时间，纵轴为幅度：

```
1 axs[0, 0].plot(t, signal)
2 axs[0, 0].set_title('时域波形', fontproperties=my_font)
3 axs[0, 0].set_xlabel('时间', fontproperties=my_font)
4 axs[0, 0].set_ylabel('幅度', fontproperties=my_font)
5 axs[0, 0].grid(True)
```

### 3.4 计算信号的频谱近似

计算信号的频谱近似，绘制频谱图，并与理论频谱进行比较。

通过计算 FFT 得到的结果，绘制信号的频谱图，并与理论频谱进行对比。频谱图展示了信号在频率域上的成分分布情况，通过分析频谱图的差异，可以深入理解频谱计算误差的来源，例如频谱泄露、分辨率不足等问题。

```
1 # 计算并绘制频谱图
2 N = 1024 # FFT长度
3 fft_result = fft(signal)
4
5 # FFT结果的幅度谱。对FFT结果取绝对值，并除以FFT长度N，以获得每个频率分
  量的幅度值：
6 fft_magnitude = np.abs(fft_result) / N
7
8 # 保留正一半的FFT结果，因为实际分量大小是正负的和，因此需要乘以2：
9 fft_magnitude = fft_magnitude[:N // 2] * 2
10 # 计算频率轴：
11 freqs = np.fft.fftfreq(N, d=t[1] - t[0])[:N // 2]
12 # 做截取：
13 if len(freqs) > len(fft_magnitude):
14     freqs = freqs[:len(fft_magnitude)]
15 axs[0, 1].stem(freqs, fft_magnitude)
16 axs[0, 1].set_title('频谱图', fontproperties=my_font)
17 axs[0, 1].set_xlabel('频率', fontproperties=my_font)
18 axs[0, 1].set_ylabel('幅度', fontproperties=my_font)
19 axs[0, 1].grid(True)
```

原始信号时域波形与 FFT 变换后的频谱如图 1 所示。

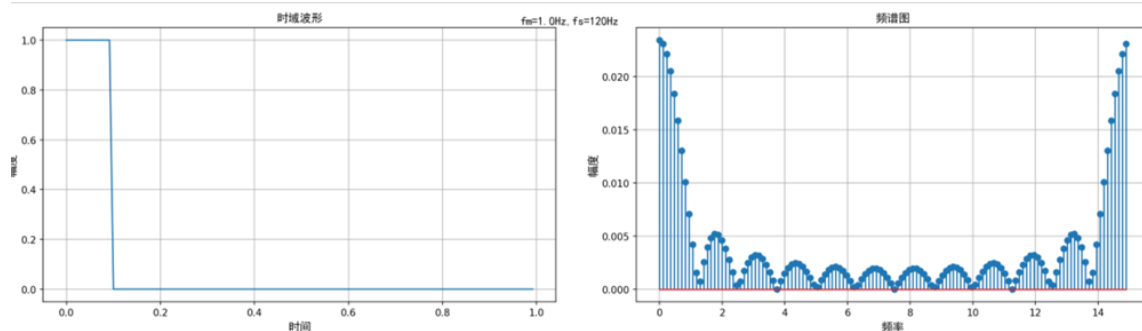


图 1: 时域波形与 FFT 频谱

### 3.5 误差的来源及 FFT 分辨率对计算结果的影响

#### 3.5.1 误差的来源

通过切换不同的采样率进行上述实验，可知采样率的大小是误差的来源之一。此外，对图 1 右侧观察到出现了**泄漏效应**。

泄漏效应的出现主要是因为信号的长度有限，导致在计算 FFT 时无法完整地捕获到信号的周期性特征，从而在频谱图中表现为主瓣变宽，以及频谱图右侧出现了额外的能量。

研究不同 FFT 长度对频谱计算结果的影响。通过对比不同长度下的频谱图，我们发现 FFT 长度越大，频谱分辨率越高，能够更清晰地分辨信号中的频率成分。这也使得栅栏效应减弱，谱线更加清晰。

#### 3.5.2 不同长度 FFT

绘制不同 FFT 长度的频谱图：

```
1 # 长度分别为 256, 512, 1024, 2048
2 for idx, N in enumerate([256, 512, 1024, 2048]):
3     fft_result = fft(signal)
4     fft_magnitude = np.abs(fft_result) / N
5     fft_magnitude = fft_magnitude[:N // 2] * 2
6     freqs = np.fft.fftfreq(N, d=t[1] - t[0])[:N // 2]
7     print(len(freqs), len(fft_magnitude))
8     # 截取:
9     if len(freqs) > len(fft_magnitude):
10         freqs = freqs[:len(fft_magnitude)]
11     axs[1, 0].stem(freqs, fft_magnitude, label=f'FFT Length = {N}',
12                    linefmt=f'C{idx}', markerfmt=f'C{idx}o')
13 axs[1, 0].set_title('频谱图 - 不同FFT长度对比', fontproperties=my_font)
14 axs[1, 0].set_xlabel('频率', fontproperties=my_font)
15 axs[1, 0].set_ylabel('幅度', fontproperties=my_font)
16 axs[1, 0].legend()
17 axs[1, 0].grid(True)
```

结果如图 2 所示：

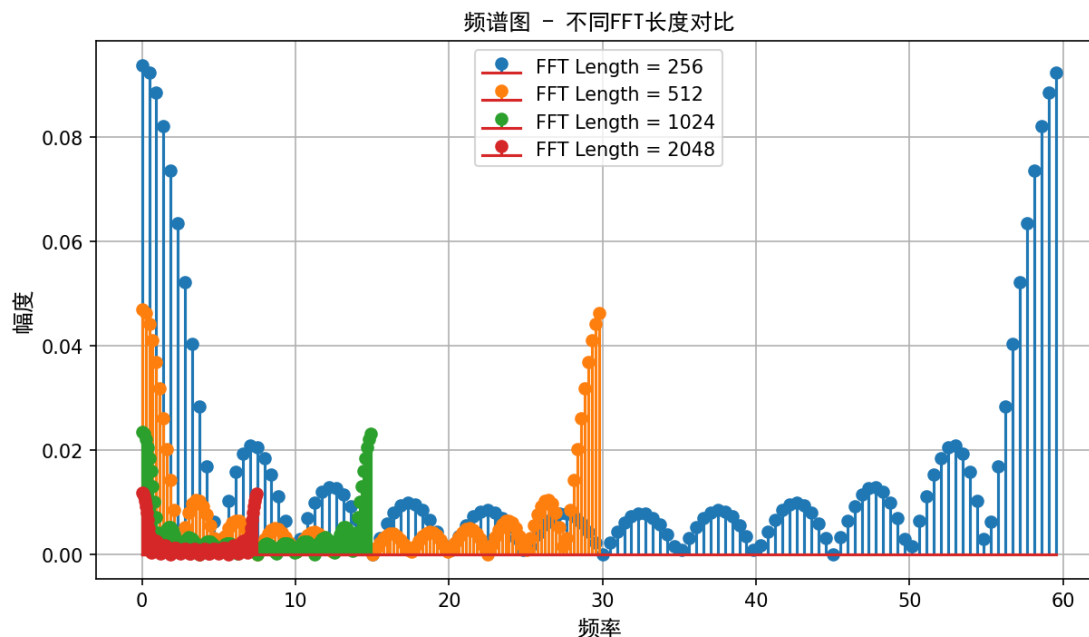


图 2: 不同长度 FFT 对比

$N$  越大，分辨率越强，栅栏效应越弱，能看到更多的谱线。

分辨率越强，每个频率分量之间的间隔越小，信号越窄、越集中。

### 3.6 优化 FFT 计算

为了优化 FFT 计算，提高频谱计算的准确性，我们引入了汉明窗口对信号进行加窗处理。加窗可以减少频谱泄露，从而提高频谱的准确性和清晰度。优化后的频谱图展示了加窗后的效果，能够更精确地反映信号的频谱特征。

优化后的频谱图（加窗，汉明窗，非矩形窗，减少频谱泄露）：

```
1 # 汉明窗
2 windowed_signal = signal * np.hamming(len(signal))
```

末端补零，保证最小记录点数  $N$  达到 1024， $N$  应满足  $\frac{T}{T}$ 。

补零可以在保持原频谱形状不变的情况下，使谱线变密，同时可以有效规避**泄漏效应**。

```
1 fft_result = fft(zero_padded_signal)
2 fft_magnitude = np.abs(fft_result) / len(zero_padded_signal)
3 fft_magnitude = fft_magnitude[:len(zero_padded_signal) // 2] * 2
4 freqs = np.fft.fftfreq(len(zero_padded_signal), d=t[1] - t[0])[:len(
    zero_padded_signal) // 2]
5 axs[1, 1].stem(freqs, fft_magnitude)
6 axs[1, 1].set_title('优化后的频谱图', fontproperties=my_font)
7 axs[1, 1].set_xlabel('频率', fontproperties=my_font)
8 axs[1, 1].set_ylabel('幅度', fontproperties=my_font)
9 axs[1, 1].grid(True)
```

最后调整布局:

```
1 # 调整布局:
2 plt.tight_layout()
3 plt.suptitle(f'fm={1/period}Hz,fs={new_sampling_rate}Hz',
4             fontproperties=my_font)
5 plt.show()
```

优化后的频谱图如图 3 所示:

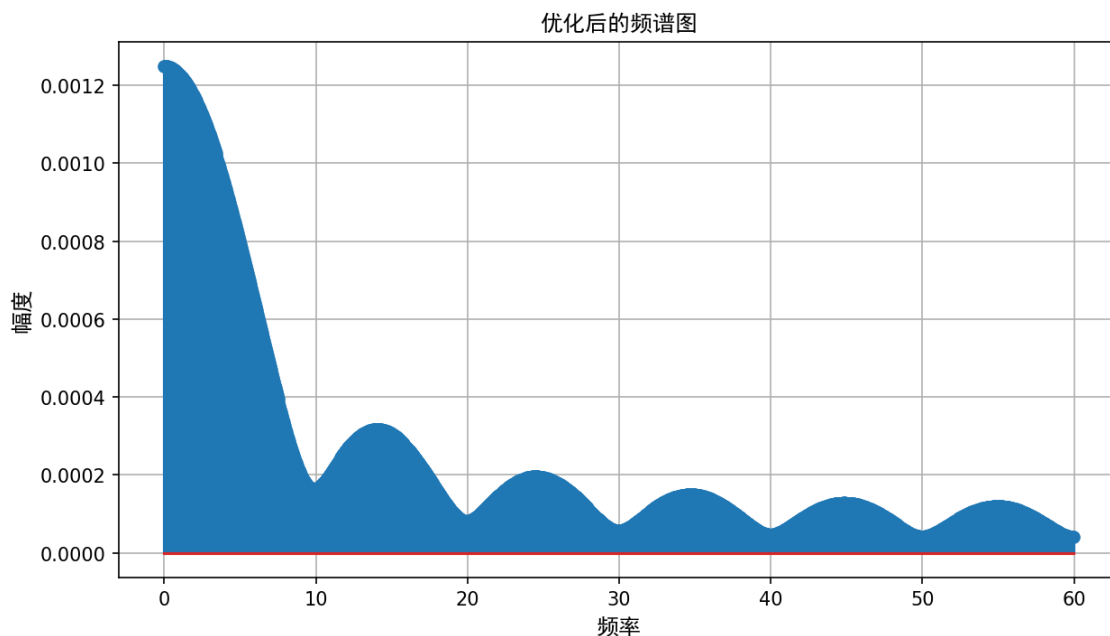


图 3: 优化后的频谱图

## 4 总程序

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from matplotlib.font_manager import FontProperties
4
5 my_font = FontProperties(fname=r"c:\windows\fonts\SimHei.ttf", size=12)
6
7 # 信号参数
8 amplitude = 1
9 width = 0.1 # 脉冲宽度
10 period = 1 # 信号周期
11
12 # 设置新的采样率
13 new_sampling_rate = 120
```



```
14 # 采样率太低会出现混叠效应, 因此设置在120
15 # 尽管奈奎斯特采样定理要求2倍, 但实测100倍以下可能出现混叠
16 num_samples = int(new_sampling_rate * period)
17
18 # 生成新的时间轴和信号
19 t = np.linspace(0, period, num_samples, endpoint=False)
20 signal = amplitude * (t % period < width)
21
22 def pad_to_power_of_two(x):
23     N = len(x)
24     if np.log2(N) % 1 > 0: # 检查N是否为2的幂
25         next_power_of_two = 2**int(np.ceil(np.log2(N)))
26         padded_x = np.zeros(next_power_of_two)
27         padded_x[:N] = x
28         return padded_x
29     else:
30         return x
31
32 def fft(x):
33     x = pad_to_power_of_two(x)
34     N = len(x)
35     if N <= 1:
36         return x
37     even = fft(x[0::2])
38     odd = fft(x[1::2])
39     #  $W_{nk} = \exp(-2j * \pi * k / N)$ 
40     # T是FFT的奇数项
41     T = [ np.exp(-2j * np.pi * k / N) * odd[k] for k in range(N // 2)]
42     return [even[k] + T[k] for k in range(N // 2)] + [even[k] - T[k]
43         for k in range(N // 2)]
44
45 # 创建子图
46 fig, axs = plt.subplots(2, 2, figsize=(14, 10))
47
48 # 绘制时域波形
49 axs[0, 0].plot(t, signal)
50 axs[0, 0].set_title('时域波形', fontproperties=my_font)
51 axs[0, 0].set_xlabel('时间', fontproperties=my_font)
52 axs[0, 0].set_ylabel('幅度', fontproperties=my_font)
53 axs[0, 0].grid(True)
54
55 # 计算并绘制频谱图
```

```
55 N = 1024 # FFT长度
56 fft_result = fft(signal)
57 # FFT结果的幅度谱。对FFT结果取绝对值，并除以FFT长度N，以获得每个频率分
    量的幅度值。
58 fft_magnitude = np.abs(fft_result) / N
59 # 保留正一半的FFT结果，因为实际分量大小是正负的和，因此需要乘以2
60 fft_magnitude = fft_magnitude[:N // 2] * 2
61 # 计算频率轴
62 freqs = np.fft.fftfreq(N, d=t[1] - t[0])[:N // 2]
63 # 截取
64 if len(freqs) > len(fft_magnitude):
65     freqs = freqs[:len(fft_magnitude)]
66 axs[0, 1].stem(freqs, fft_magnitude)
67 axs[0, 1].set_title('频谱图', fontproperties=my_font)
68 axs[0, 1].set_xlabel('频率', fontproperties=my_font)
69 axs[0, 1].set_ylabel('幅度', fontproperties=my_font)
70 axs[0, 1].grid(True)
71
72
73 # 绘制不同FFT长度的频谱图
74 # N越大，分辨率越强，栅栏效应越弱，能看到更多的谱线
75 # 分辨率越强，每个频率分量之间的间隔越小，信号越窄、越集中
76 for idx, N in enumerate([256, 512, 1024, 2048]):
77     fft_result = fft(signal)
78     fft_magnitude = np.abs(fft_result) / N
79     fft_magnitude = fft_magnitude[:N // 2] * 2
80     freqs = np.fft.fftfreq(N, d=t[1] - t[0])[:N // 2]
81     print(len(freqs), len(fft_magnitude))
82     # 截取
83     if len(freqs) > len(fft_magnitude):
84         freqs = freqs[:len(fft_magnitude)]
85     axs[1, 0].stem(freqs, fft_magnitude, label=f'FFT Length = {N}',
        linefmt=f'C{idx}', markerfmt=f'C{idx}o')
86
87 axs[1, 0].set_title('频谱图 - 不同FFT长度对比', fontproperties=my_font)
88 axs[1, 0].set_xlabel('频率', fontproperties=my_font)
89 axs[1, 0].set_ylabel('幅度', fontproperties=my_font)
90 axs[1, 0].legend()
91 axs[1, 0].grid(True)
92
93 # 优化后的频谱图
94 # 加窗，汉明窗，非矩形窗，减少频谱泄露
```

```

95 windowed_signal = signal * np.hamming(len(signal))
96 # 末端补零, 保证最小记录点数N达到1024
97 # 实际上, N应满足 $T1/T$ 
98 # 补零可以在保持原频谱形状不变的情况下, 使谱线变密
99 zero_padded_signal = np.pad(windowed_signal, (0, N - len(
    windowed_signal)), 'constant')
100 fft_result = fft(zero_padded_signal)
101 fft_magnitude = np.abs(fft_result) / len(zero_padded_signal)
102 fft_magnitude = fft_magnitude[:len(zero_padded_signal) // 2] * 2
103 freqs = np.fft.fftfreq(len(zero_padded_signal), d=t[1] - t[0])[:len(
    zero_padded_signal) // 2]
104 axs[1, 1].stem(freqs, fft_magnitude)
105 axs[1, 1].set_title('优化后的频谱图', fontproperties=my_font)
106 axs[1, 1].set_xlabel('频率', fontproperties=my_font)
107 axs[1, 1].set_ylabel('幅度', fontproperties=my_font)
108 axs[1, 1].grid(True)
109
110 # 调整布局
111 plt.tight_layout()
112 plt.suptitle(f'fm={1/period}Hz, fs={new_sampling_rate}Hz',
    fontproperties=my_font)
113 plt.show()

```

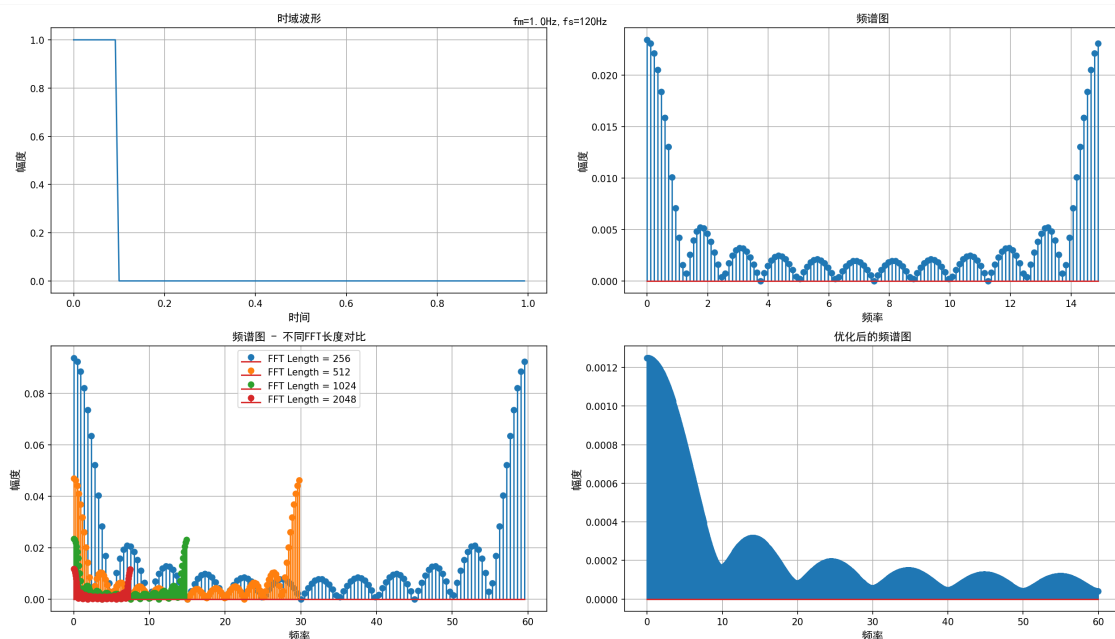


图 4: 最终结果

左上角子图为原始信号的时域波形。

右上角子图为采样频率在 120Hz, FFT 长度  $N$  取 1024 时的结果, 可以观察到出现了明显的

泄漏效应。

左下角子图为不同 FFT 长度的结果，未加窗且未补零，也可以观察到明显的**泄漏效应**。此外可以观察到 FFT 长度越大**栅栏效应**越弱。

由于采样频率满足奈奎斯特采样定理且裕量很大，因此未观察到频谱混叠现象。

右下角为优化后的频谱图，可以看到效果相对其他频谱图极佳（FFT 长度取 1024）。

## 5 结论与讨论

本实验通过实际编程实现了 FFT 算法对模拟非周期信号频谱的计算。通过对比计算结果与理论预期，分析了频谱计算误差的可能来源，并讨论了采样率、FFT 长度对频谱分辨率和准确性的影响。实验结果表明，适当选择采样率和 FFT 长度，以及加窗处理，能有效提高频谱计算的准确性和可信度。

### 5.1 误差的处理方法

为了减小**栅栏效应**，以下两种方法较为有效：

1. **零填充**：在进行 FFT 之前，可以对信号进行零填充，增加信号的长度。零填充可以提高 FFT 的频率分辨率，减少栅栏效应的影响。零填充可以在保持形状不变的前提下，使谱线变密。
2. **增加信号长度（增加 N）**：增加信号长度可以提高 FFT 的频率分辨率，能够看到更多原来看不到的谱线。

为了减小**混叠误差**，可以通过

1. **设置合适的采样频率**：采样频率应满足奈奎斯特采样定理。
2. **控制频谱分辨力的大小**：频率分辨力与最小记录长度成反比，两者大小不能过于极端。

为了减少**泄漏效应**，以下三种方法较为有效：

1. **增加信号长度（增加 N）**：增加信号长度可以提高 FFT 的频率分辨率，从而减少泄漏效应。但是，增加信号长度会增加计算复杂度和内存消耗。
2. **使用窗函数（加窗）**：在计算 FFT 之前，可以对信号应用窗函数来减少泄漏效应。常用的窗函数包括汉明窗（Hamming Window）、黑曼窗（Blackman Window）等。窗函数可以在一定程度上抑制信号在两端的振荡，减少泄漏效应。
3. **零填充**：在计算 FFT 之前，可以对信号进行零填充，使得信号长度达到一个较大的 2 的幂次方。零填充可以增加频谱图的分辨率，但并不能完全消除泄漏效应。

零填充能够非常有效地消除误差，图 5 中上图是零填充后的频谱图，下图是零填充和汉明窗同时应用的频谱图，可以看到两者的效果相差并不很大，但都已非常可观了，特别是相较于无任何处理的频谱图而言（如图 1 所示）。

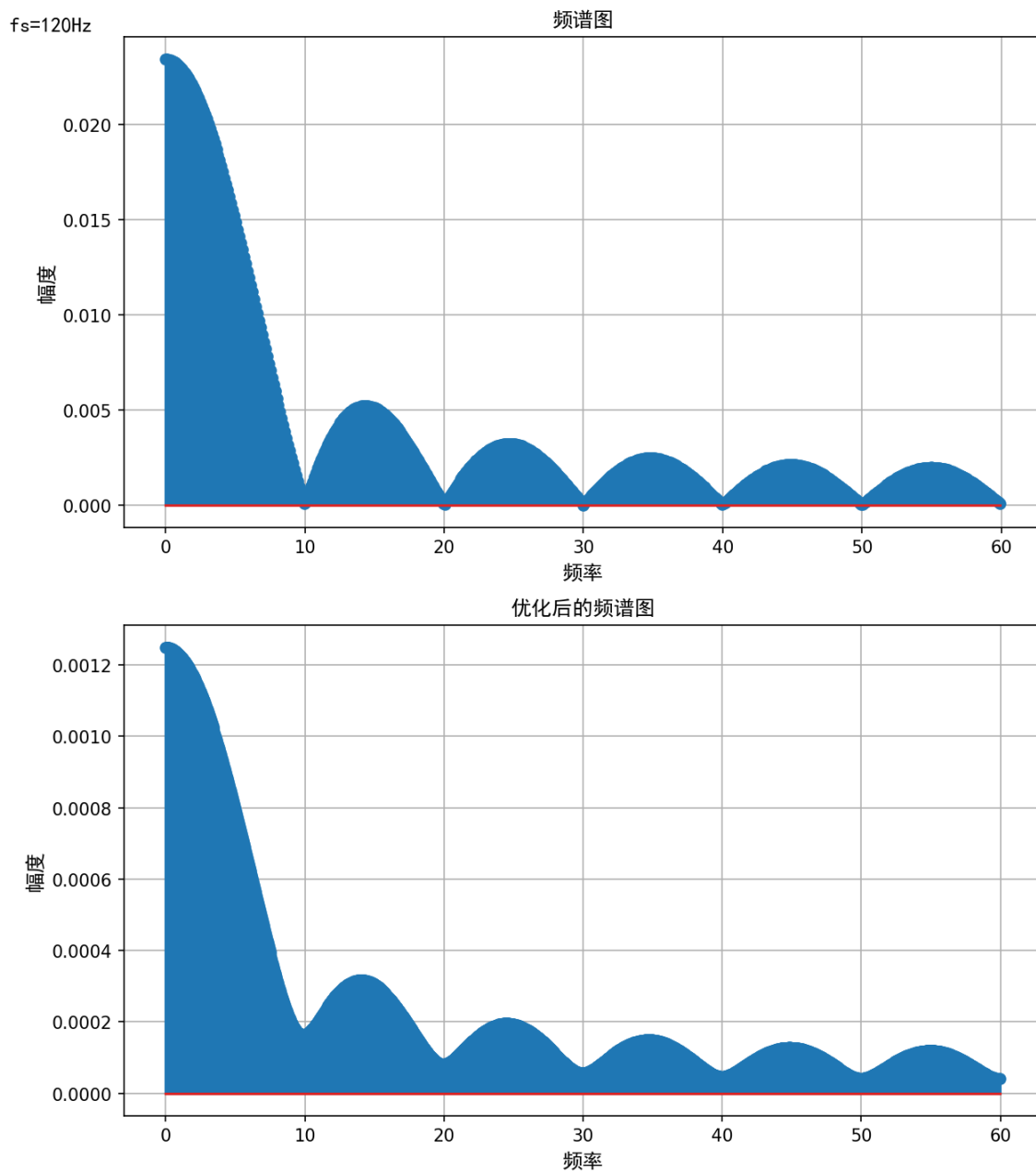


图 5: 零填充

## 5.2 采样频率的选取

如果严格参照奈奎斯特采样定理，设定采样频率为  $2\text{Hz}$ ，可以得到下图所示的结果：

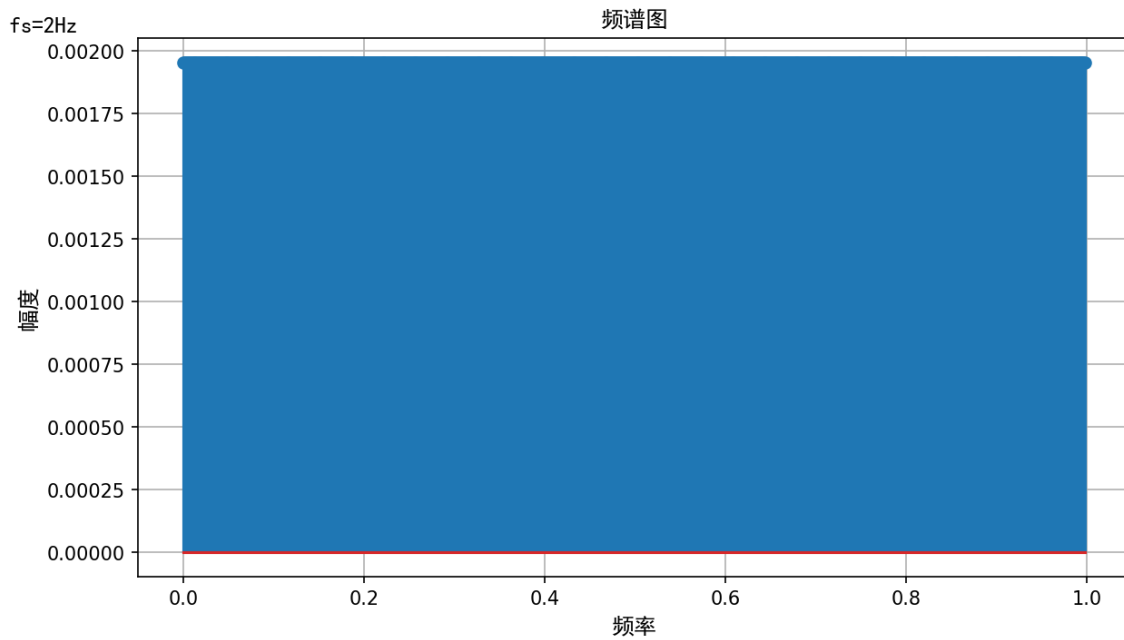


图 6: 2Hz 下频谱

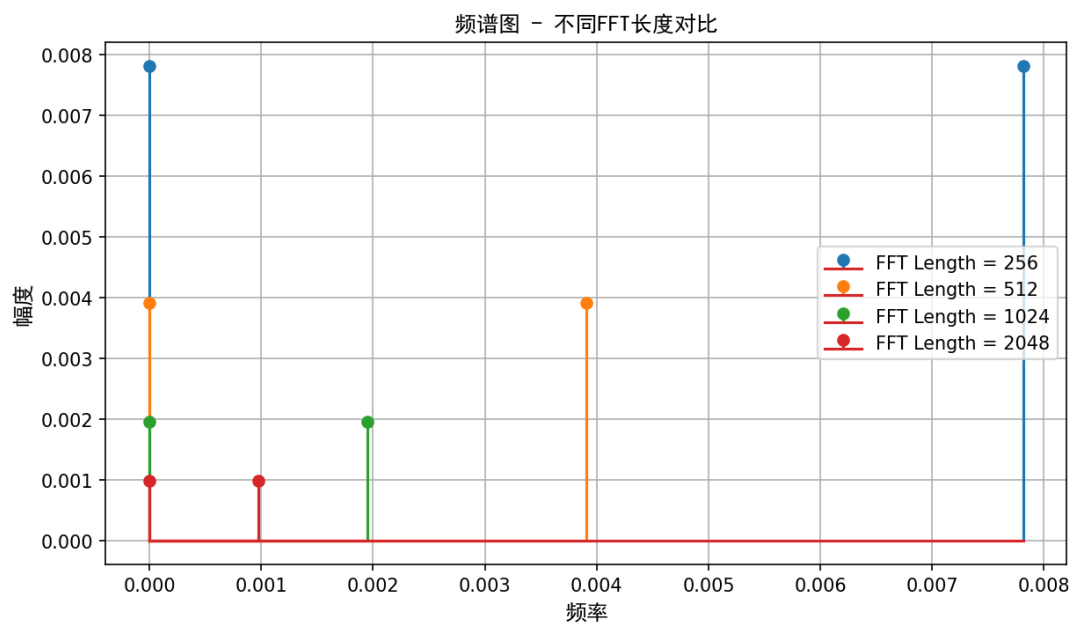


图 7: 2Hz 下不同长度 FFT

可以观察到混叠现象，这是由于实验物理条件的限制（如更换信号生成方式等），并不意味着奈奎斯特采样定理是错误的。

而采样频率过大效果也未必很好，1000Hz 效果与 120Hz 相比较差。

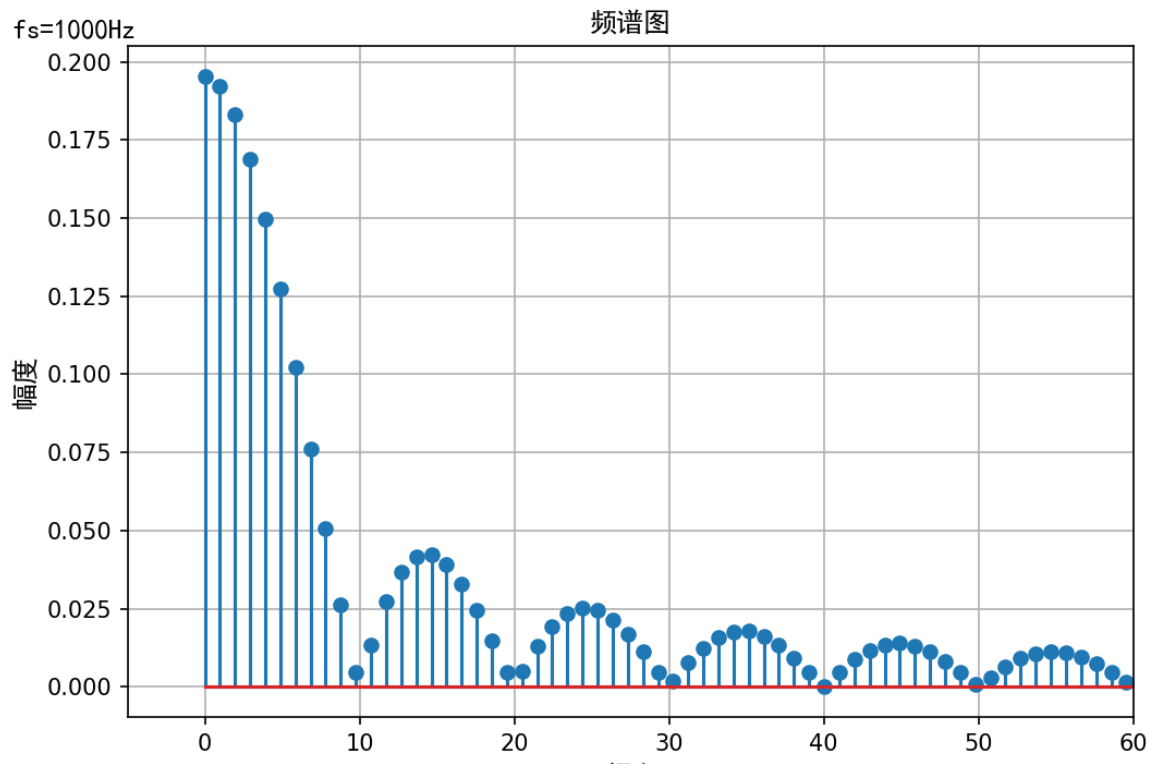


图 8: 1000Hz 下频谱

## 6 组员分工与签名

### 6.1 组员分工

组员分工：

- 岳志邦：生成模拟非周期信号，设计 FFT 频谱近似计算工作量 40%
- 朱骥尧：分析计算误差的来源，绘制不同长度的频谱图工作量 30%
- 朱梓豪：优化 FFT 计算的方法，提高频谱近似的准确性工作量 30%

### 6.2 组员签名

岳志邦

朱彊尧

朱梓豪