

信号与系统设计作业

项目 目 报 告

题目：基于快速傅里叶变换 (FFT) 的
模拟非周期信号频谱近似计算

作者：

2024 年 6 月

设计要求

- 一、生成一个模拟非周期性信号
- 二、设计一个 FFT 频谱近似计算实验
- 三、比较计算的频谱结果与理论频谱的差异，分析误差来源
- 四、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

项目过程

一、模拟非周期性信号生成

除了利用 python 生成的矩形方波信号以外，还采用了两段音频文件验证了 FFT：

- (1) test1.wav 文件为加入小幅度随机噪声后的 2700Hz 纯音：

```
1. sample_rate, samples = wavfile.read('test1.wav')
2. noise = 3000 * np.random.randn(len(samples))
3. samples = list(np.array(samples) + noise)
4. cut_samples_1 = sample_array(samples, 5)
```

其平均噪音幅度/平均纯音幅度比值为 3: 10。

(2) test2.wav 文件采用手机外放一段人声视频，模拟加入噪音后的人声。时长 20.0s。

二、FFT 频谱近似计算

输入的音频文件样本点密度足够大(如 test1.wav 的号文件长 5s, 有 240000 个样本点，每秒 48000 个)，即使已经完成离散化，仍可以近似看作是模拟信号。然而，将音频文件直接输入函数中，可能出现样本点数过多，从而导致绘图时过多样本点挤在一起，使作图不直观的情况。为了分散过于密集的样本点，定义以下 sample_array 函数，其输入为读取的音频文件的列表 Samp 以及采样间隔 n：

```
1. def sample_array(Samp, n):
2.     """
3.         The quantity of the sound signal samples captured may be excessive,
4.         The function is introduced to reduce sample density.
5.         This will facilitate an easier visualization.
6.
7.         n : Sampling period
8.     """
9.     sampled_values = []
10.    for i in range(0, len(Samp), n):
11.        sampled_values.append(Samp[i])
12.    return sampled_values
```

适当选取采样间隔，能够使得在对准确度影响不大的情况下，大幅减小运算时间并适当提高绘图精度。经过该函数“采样”的列表，完成了输入信号的预处理。

定义 `HandmadeFFT` 类，其中有 `fft` 方法与绘制频率图的 `plot_fft` 方法。其输入参量为数组 `signal` 以及音频时间 `time`。一些基本的参量，如信号长度 `N` 等，在初始化的 `__init__` 函数里定义。

由于采用 2 为基的时间抽选 FFT 算法，需保证运算的数组总长度为 2 的整数次幂。因此，判断输入数组的长度并将其增广为 2 的整数次幂。得到用于参与运算的两个参数：

```
1. self.ext_N = 2**(len(signal) - 1).bit_length()
2. self.ext_signal = list(self.signal) + [0.0] * (self.ext_N - len(self.signal))
```

`fft` 函数实现具体的运算过程：迭代地将序列拆分，并采用蝶形运算。

`plot_fft` 方法实现了对单侧频谱的绘制。

以下是关于类的完整定义：

```
1. class HandmadeFFT:
2.     """
3.     Handmade FFT
4.     signal : A discretized signal in the form of list or np.
               ndarray.
5.     time : The time of sampling.
6.     """
7.     def __init__(self, signal, time):
8.         self.signal = signal
9.         self.N = len(signal)
10.        self.time = time
11.        self.ext_N = 2**(len(signal) - 1).bit_length()
12.        self.ext_signal = list(self.signal) + [0.0] * (self.
            ext_N - len(self.signal))
13.
14.
15.        def fft(self):
16.            if self.ext_N <= 1:
17.                return self.ext_signal
18.
19.            even = HandmadeFFT(self.ext_signal[::2], self.time).f
                ft()
20.            odd = HandmadeFFT(self.ext_signal[1::2], self.time).f
                ft()
21.
22.            T = [np.exp(-2j * np.pi * k / self.ext_N) * odd[k] f
                or k in range(self.ext_N // 2)]
23.
24.            return [even[k] + T[k] for k in range(self.ext_N //
                2)] + \
```

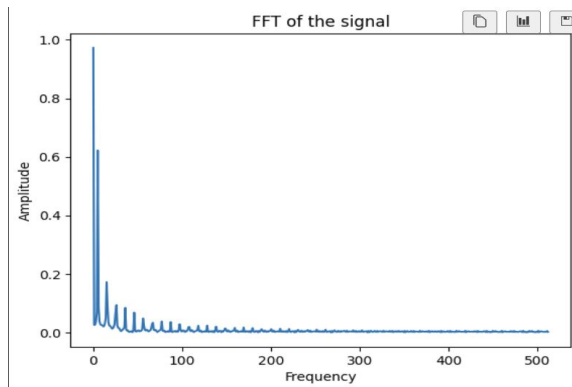
```

25.             [even[k] - T[k] for k in range(self.ext_N //
26.             2)]
27.     def plott(self):
28.         """
29.         Plot the waveform.
30.         """
31.         P = np.abs(self.signal / self.N)
32.         t = np.linspace(0, self.time, self.N)
33.
34.         plt.figure()
35.         plt.plot(t, P)
36.         plt.xlabel('Time')
37.         plt.ylabel('Amplitude')
38.         plt.title('Waveform of the signal')
39.         plt.show()
40.
41.
42.     def plotf(self):
43.         """
44.         Plot the frequency spectrum.
45.         """
46.         fft_vals = np.array(self.fft())
47.         P2 = np.abs(fft_vals / self.ext_N)
48.         P1 = P2[:self.ext_N // 2] * 2
49.         f = np.linspace(0, (len(self.ext_signal)/self.time)
50.         / 2, self.ext_N // 2)
51.
52.         plt.figure()
53.         plt.plot(f, P1)
54.         plt.xlabel('Frequency')
55.         plt.ylabel('Amplitude')
56.         plt.title('FFT of the signal')
57.         plt.show()

```

三、频谱结果误差分析

(1) 矩形方波



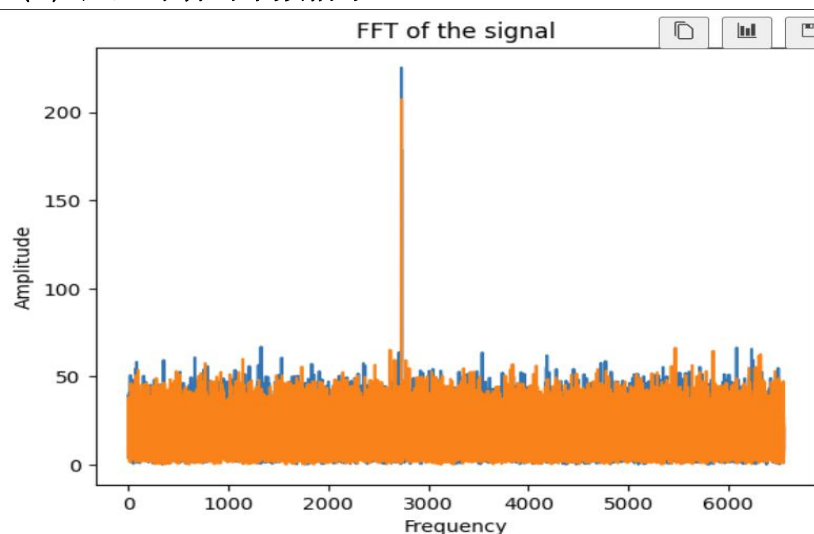
矩形方波的参数为幅度 $E=1.0$ ，脉冲时长 $T=0.1$ 。采样时间为 $1s$ ，采样频率 $f_s = 1000Hz$ 。根据理论分析，其离散傅里叶变换为

$$DFT(R_{100}(n)) = \sum_{k=0}^{100} e^{-j \frac{\pi k n}{500}}$$

当且仅当 $k = 10m$ ($m = 0, 1, 2, 3, \dots$) 时，值不为零。从图像看出， $0 \sim 100Hz$ 的范围内有 10 个峰。初步判断运算正确。

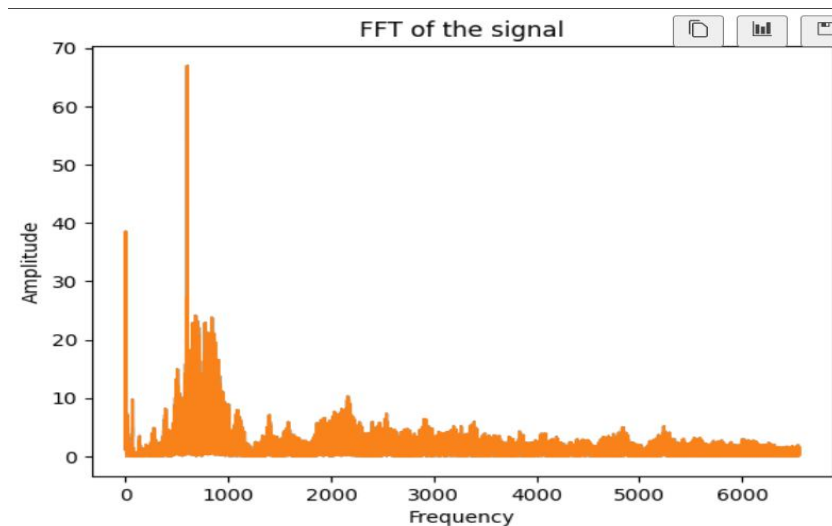
值得注意的是，每个峰周围的序列并非严格取 0，而是有幅值的迅速衰减，这与 DFT 的理论分析有所出入。这将在第四节被分析。

(2) 加入噪音的单频信号



选取的纯音频率为 $2700Hz$ ，加入幅度约为纯音幅度 0.3 倍的随机噪音，从频谱图上看，与预期结果大致相当。

(3) 录制的人声音频信号



人谈话的声音频率在 500 ~ 2000Hz 之间。频谱图中主要的声音信号集中在这一区间，证明程序设计的正确性。

在大约 600 ~ 700Hz 的区域内出现了一个突出峰，尚不清楚其原因。

高于 2000Hz，频域幅度逐渐衰减。此范围内主要为高频噪音。

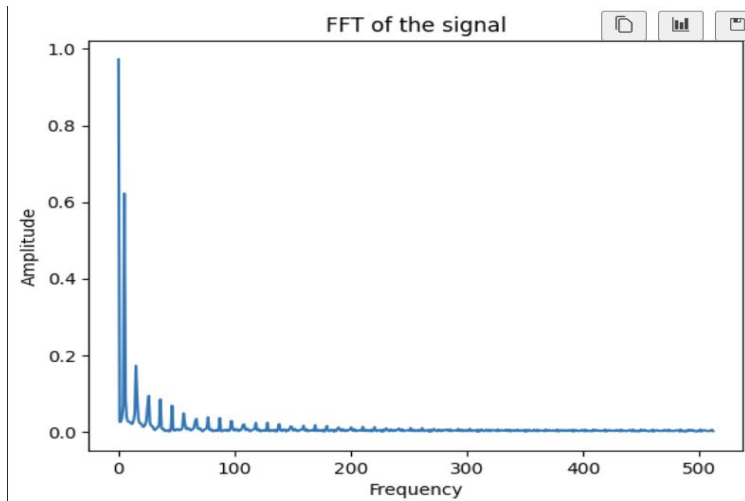
四、进一步分析

以矩形方波信号为例。

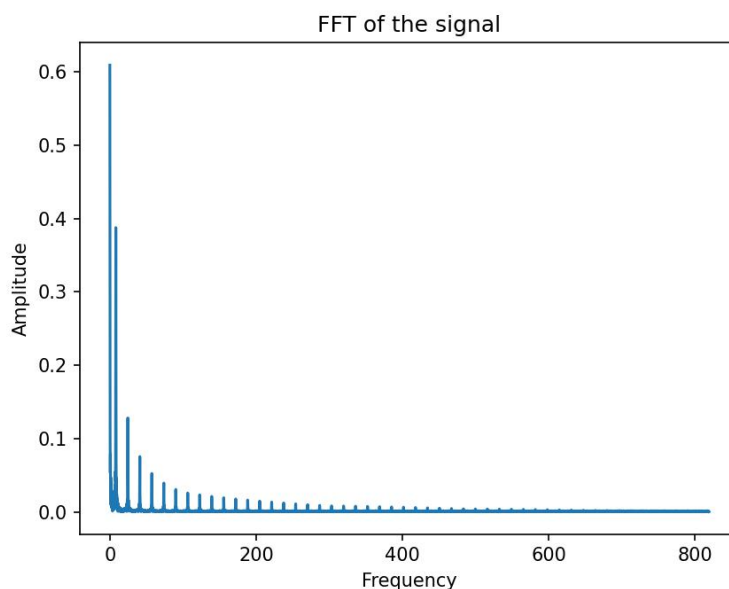
前面提到矩形方波频谱应该是 10 的整数倍时出现的陡峭的峰。如果信号足够理想，应该仅在 $f = 10m$ (m 为整数) 处观察到峰的出现，其余值均为 0。然而实际上，无论如何优化采集方法，我们一定得不到这样的峰。我们小组认为，这是由于信号采样时长不是无穷大引起的。下面结合所学知识，对此展开分析。

直观上说，对一个自然信号的观察（采集）时间愈长，我们能够断言其一定是某频率信号的可能性愈大。举生活中的例子，譬如对两盏闪烁的灯持续观察，一开始二者的亮灭在肉眼看来同步，但随着观察时间延长，二者频率之间细微的差异会被逐渐放大，直至可以断言异步。这就是延长时间能够增加频率分辨率的道理。

将采样时间推至极限来说，对一个信号做无穷时长的傅里叶分析，我们一定能够清楚地区分里面的所有频率的信号。然而实际的信号采样时长都是有限的，这就导致可能对于频率十分接近的两个谐波，我们分不清二者到底谁是这个自然信号的组成部分，抑或是二者都有。从频谱上说，有限时长的信号，其频谱中各个峰的变化一定是“缓变”的。



在延长抽样总时长至 **5s** 的基础上重复分析，得到下面的频谱图，其频谱的离散化进一步提高，频率分辨率亦得到提高。通过分析，我们可以断言矩形信号中确切地含有某些值的谐波成分：



可见提高抽样时长能够提高频率分辨率。

五、程序展示

```
57. import numpy as np
58. from scipy.io import wavfile
59. import matplotlib.pyplot as plt
60.
61.
62. def sample_array(Samp, n):
63.     """
64.         The quantity of the sound signal samples captured may be excessive,
```

```

65.         The function is introduced to reduce sample density.
66.         This will facilitate an easier visualization.
67.
68.         n : Sampling period
69.         """
70.         sampled_values = []
71.         for i in range(0, len(Samp), n):
72.             sampled_values.append(Samp[i])
73.         return sampled_values
74.
75. def zero_pad_array(array, target_length):
76.     """
77.     Extend N to an integer multiple of 2.
78.     Haven't been used ever since it was born.
79.     Someone has had its mission done for him.
80.     """
81.     current_length = len(array)
82.
83.     if current_length >= target_length:
84.         return array
85.
86.     num_zeros = target_length - current_length
87.     zero_padded_array = array + [0.0] * num_zeros
88.
89.     return zero_padded_array
90.
91.
92. class HandmadeFFT:
93.     """
94.     Handmade FFT
95.     signal : A discretized signal in the form of list or np.
96.             ndarray.
97.     time : The time of sampling.
98.     """
99.     def __init__(self, signal, time):
100.         self.signal = signal
101.         self.N = len(signal)
102.         self.time = time
103.         self.ext_N = 2**(len(signal) - 1).bit_length()
104.         self.ext_signal = list(self.signal) + [0.0] * (self.
105.             ext_N - len(self.signal))

```



```

106.     def fft(self):
107.         if self.ext_N <= 1:
108.             return self.ext_signal
109.
110.         even = HandmadeFFT(self.ext_signal[::2],self.time).
            fft()
111.         odd = HandmadeFFT(self.ext_signal[1::2],self.time).
            fft()
112.
113.         T = [np.exp(-2j * np.pi * k / self.ext_N) * odd[k]
            for k in range(self.ext_N // 2)]
114.
115.         return [even[k] + T[k] for k in range(self.ext_N //
            2)] + \
116.             [even[k] - T[k] for k in range(self.ext_N //
            2)]
117.
118.     def plott(self):
119.         """
120.         Plot the waveform.
121.         """
122.         P = np.abs(self.signal / self.N)
123.         t = np.linspace(0, self.time, self.N)
124.
125.         plt.figure()
126.         plt.plot(t, P)
127.         plt.xlabel('Time')
128.         plt.ylabel('Amplitude')
129.         plt.title('Waveform of the signal')
130.         plt.show()
131.
132.
133.     def plotf(self):
134.         """
135.         Plot the frequency spectrum.
136.         """
137.         fft_vals = np.array(self.fft())
138.         P2 = np.abs(fft_vals / self.ext_N)
139.         P1 = P2[:self.ext_N // 2] * 2
140.         f = np.linspace(0, (len(self.ext_signal)/self.time)
            / 2, self.ext_N // 2)
141.
142.         plt.figure()
143.         plt.plot(f, P1)

```

```

144.         plt.xlabel('Frequency')
145.         plt.ylabel('Amplitude')
146.         plt.title('FFT of the signal')
147.         plt.show()
148.
149.
150. if __name__ == '__main__':
151.     """
152.     Several tests for the class.
153.     """
154. #####
155.     """
156.     A rectangular signal for test.
157.     """
158.     Fs = 1000
159.     T = 1 / Fs
160.     duration = 5.0
161.     N = int(Fs * duration)
162.     rec_t = np.arange(0, duration, T)
163.     pulse_width = 0.1
164.     amplitude = 1.0
165.
166.     signal = amplitude * np.heaviside(np.mod(rec_t, pulse_w
        idth * 2) - pulse_width, 1)
167.     result = HandmadeFFT(signal, 5.0)
168.     result.plotf()
169.
170. #####
171.     """
172.     'test1.wav' lasts for 5 sec and is a pure sound mixxed
        with random noise.
173.     """
174.     sample_rate, samples_1 = wavfile.read('test1.wav')
175.     noise = 3000 * np.random.randn(len(samples_1), 2)
176.     samples_1 = np.array(samples_1) + noise
177.
178.     cut_samples_1 = sample_array(samples_1, 5)
179.     result_wav1 = HandmadeFFT(cut_samples_1, 5.0)
180.     result_wav1.plotf()
181.
182. #####
183.     """
184.     'test2.wav' lasts for 20 sec and is a piece of record f
        rom 3b1b.

```

```
185.      """
186.      sample_rate, samples_2 = wavfile.read('test2.wav')
187.      cut_samples_2 = sample_array(samples_2, 5)
188.      result_wav2 = HandmadeFFT(cut_samples_2, 20.0)
189.      result_wav2.plotf()
```