信号与系统设计作业

项
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：

```python
sample_rate, samples = wavfile.read('test1.wav')
noise = 3000 * np.random.randn(len(samples))
samples = list(np.array(samples) + noise)
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 的原文件长 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`：

```python
def sample_array(Samp, n):
    """
    The quantity of the sound signal samples captured may be excessive,
    The function is introduced to reduce sample density.
    This will facilitate an easier visualization.

    n : Sampling period
    """
    sampled_values = []
    for i in range(0, len(Samp), n):
        sampled_values.append(Samp[i])
    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：

```
self.ext_N = 2**(len(signal) - 1).bit_length()
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。

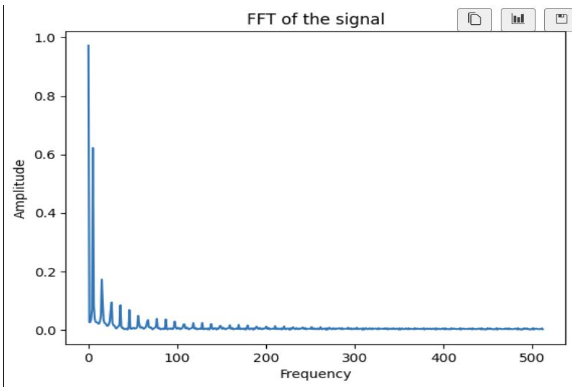
以下是关于类的完整定义：

```
class HandmadeFFT:
    """
    Handmade FFT
    signal : A discretized signal in the form of list or np.ndarray.
    time : The time of sampling.
    """
    def __init__(self, signal, time):
        self.signal = signal
        self.N = len(signal)
        self.time = time
        self.ext_N = 2**(len(signal) - 1).bit_length()
        self.ext_signal = list(self.signal) + [0.0] * (self.ext_N - len(self.signal))


    def fft(self):
        if self.ext_N <= 1:
            return self.ext_signal

        even = HandmadeFFT(self.ext_signal[::2],self.time).fft()
        odd = HandmadeFFT(self.ext_signal[1::2],self.time).fft()

        T = [np.exp(-2j * np.pi * k / self.ext_N) * odd[k] for k in range(self.ext_N // 2)]

        return [even[k] + T[k] for k in range(self.ext_N // 2)] + \
```

```
                [even[k] - T[k] for k in range(self.ext_N // 2)]

    def plott(self):
        """
        Plot the waveform.
        """
        P = np.abs(self.signal / self.N)
        t = np.linspace(0, self.time, self.N)

        plt.figure()
        plt.plot(t, P)
        plt.xlabel('Time')
        plt.ylabel('Amplitude')
        plt.title('Waveform of the signal')
        plt.show()


    def plotf(self):
        """
        Plot the frequency spectrum.
        """
        fft_vals = np.array(self.fft())
        P2 = np.abs(fft_vals / self.ext_N)
        P1 = P2[:self.ext_N // 2] * 2
        f = np.linspace(0, (len(self.ext_signal)/self.time) / 2, self.ext_N // 2)

        plt.figure()
        plt.plot(f, P1)
        plt.xlabel('Frequency')
        plt.ylabel('Amplitude')
        plt.title('FFT of the signal')
        plt.show()
```

三、频谱结果误差分析

（1）矩形方波



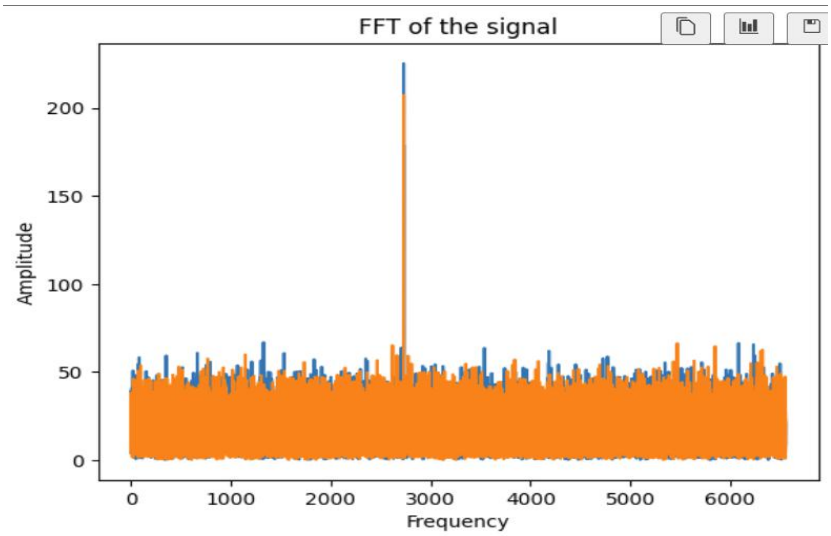
矩形方波的参数为幅度 E=1.0，脉冲时长 T=0.1。采样时间为 1s，采样频率 f_s = 1000Hz。根据理论分析，其离散傅里叶变换为

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

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

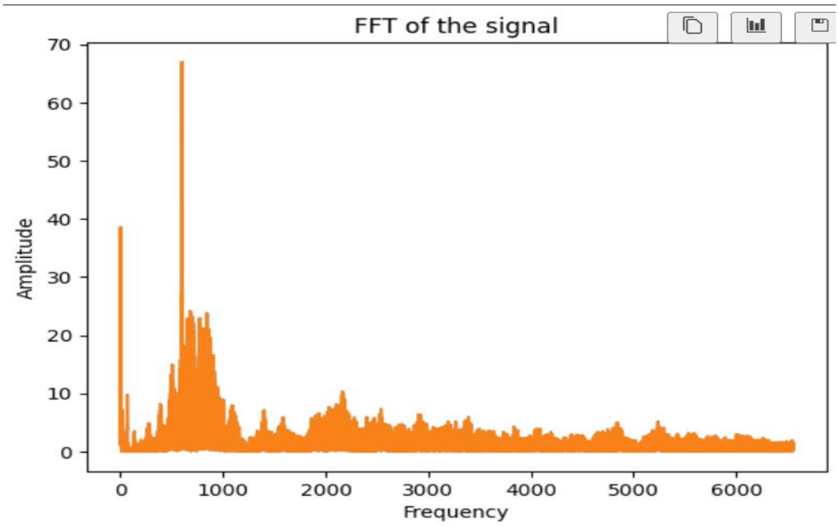
值得注意的是，每个峰周围的序列并非严格取 0，而是有幅值的迅速衰减，这与 DFT 的理论分析有所出入。这将在第四节被分析。

（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。

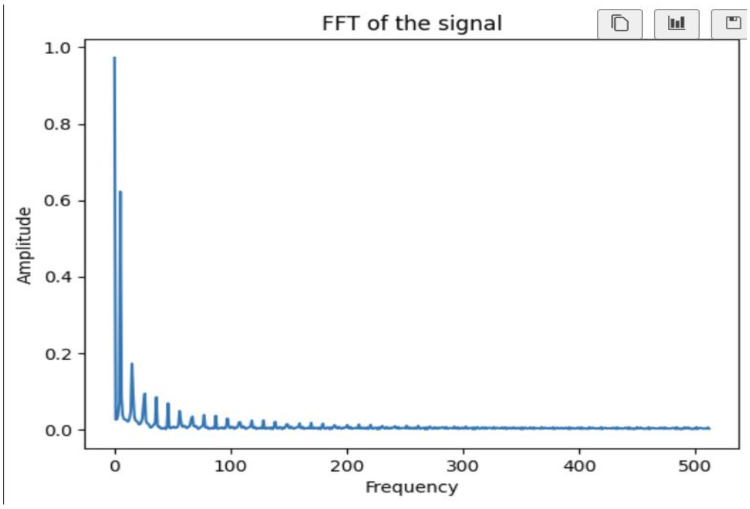
四、进一步分析

以矩形方波信号为例。

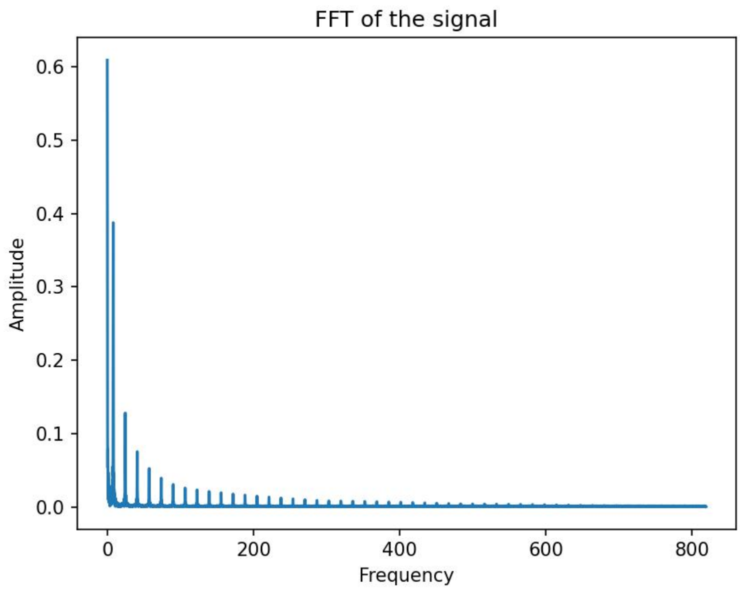
前面提到矩形方波频谱应该是 10 的整数倍时出现的陡峭的峰。如果信号足够理想，应该仅在 f = 10m（m 为整数）处观察到峰的出现，其余值均为 0。然而实际上，无论如何优化采集方法，我们一定得不到这样的峰。我们小组认为，这是由于信号采样时长不是无穷大引起的。下面结合所学知识，对此展开分析。

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开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

```
import numpy as np
from scipy.io import wavfile
import matplotlib.pyplot as plt


def sample_array(Samp, n):
        """
        The quantity of the sound signal samples captured may be excessive,
```

```
        The function is introduced to reduce sample density.
        This will facilitate an easier visualization.

        n : Sampling period
        """
        sampled_values = []
        for i in range(0, len(Samp), n):
            sampled_values.append(Samp[i])
        return sampled_values

def zero_pad_array(array, target_length):
    """
    Extend N to an integer multiple of 2.
    Haven't been used ever since it was born.
    Someone has had its mission done for him.
    """
    current_length = len(array)

    if current_length >= target_length:
        return array

    num_zeros = target_length - current_length
    zero_padded_array = array + [0.0] * num_zeros

    return zero_padded_array


class HandmadeFFT:
    """
    Handmade FFT
    signal : A discretized signal in the form of list or np.ndarray.
    time : The time of sampling.
    """
    def __init__(self, signal, time):
        self.signal = signal
        self.N = len(signal)
        self.time = time
        self.ext_N = 2**(len(signal) - 1).bit_length()
        self.ext_signal = list(self.signal) + [0.0] * (self.ext_N - len(self.signal))


```

```
    def fft(self):
        if self.ext_N <= 1:
            return self.ext_signal

        even = HandmadeFFT(self.ext_signal[::2],self.time).fft()
        odd = HandmadeFFT(self.ext_signal[1::2],self.time).fft()

        T = [np.exp(-2j * np.pi * k / self.ext_N) * odd[k] for k in range(self.ext_N // 2)]

        return [even[k] + T[k] for k in range(self.ext_N // 2)] + \
               [even[k] - T[k] for k in range(self.ext_N // 2)]

    def plott(self):
        """
        Plot the waveform.
        """
        P = np.abs(self.signal / self.N)
        t = np.linspace(0, self.time, self.N)

        plt.figure()
        plt.plot(t, P)
        plt.xlabel('Time')
        plt.ylabel('Amplitude')
        plt.title('Waveform of the signal')
        plt.show()


    def plotf(self):
        """
        Plot the frequency spectrum.
        """
        fft_vals = np.array(self.fft())
        P2 = np.abs(fft_vals / self.ext_N)
        P1 = P2[:self.ext_N // 2] * 2
        f = np.linspace(0, (len(self.ext_signal)/self.time) / 2, self.ext_N // 2)

        plt.figure()
        plt.plot(f, P1)
```

```python
        plt.xlabel('Frequency')
        plt.ylabel('Amplitude')
        plt.title('FFT of the signal')
        plt.show()


if __name__ == '__main__':
    """
    Several tests for the class.
    """
#################################
    """
    A rectangular signal for test.
    """
    Fs = 1000
    T = 1 / Fs
    duration = 5.0
    N = int(Fs * duration)
    rec_t = np.arange(0, duration, T)
    pulse_width = 0.1
    amplitude = 1.0

    signal = amplitude * np.heaviside(np.mod(rec_t, pulse_width * 2) - pulse_width, 1)
    result = HandmadeFFT(signal, 5.0)
    result.plotf()

#################################
    """
    'test1.wav' lasts for 5 sec and is a pure sound mixxed with random noise.
    """
    sample_rate, samples_1 = wavfile.read('test1.wav')
    noise = 3000 * np.random.randn(len(samples_1),2)
    samples_1 = np.array(samples_1) + noise

    cut_samples_1 = sample_array(samples_1, 5)
    result_wav1 = HandmadeFFT(cut_samples_1, 5.0)
    result_wav1.plotf()

#################################
    """
    'test2.wav' lasts for 20 sec and is a piece of record from 3b1b.
```

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