实验 2　傅里叶变换和滤波器

姓名：丁彦添
学号：1911406
日期：2021.12.21

一、实验平台：

实验开发环境：
Windows 10
Anaconda3 jupyter notebook
环境选项：

```
name: dsp
channels:
  - defaults
dependencies:
  - jupyter=1.0.0
  - python=3.7.3
- psutil=5.6.1
```

本实验所包含的 python 核心函数库 python：

```
import numpy as np
import math
import copy
import timeit
import random
import matplotlib.pyplot as plt
```

二、实验目的

1. 手动完成一个傅里叶和一个逆傅里叶变换的函数，并观察效果

A. 手动编写一个傅里叶变换和一个逆傅里叶变换；

B. 使用自定的输入序列，连续经过傅里叶变换和逆变换后，观察输出和输入序列是否有差异；

C. 调用 Python 编程环境下 Numpy 库中的快速傅里叶函数，观察手写傅里叶变换和快速傅里叶变换的效率比较，通过不同长度序列测试，观察时间消耗的关系；

2. 概念滤波器的设计（此题需要的所有函数都可以调库实现）

A. 在频域设计高通或者低通理想方砖滤波器（滤波器格式、截止频率等可以自行设计）；

B. 将该滤波器通过逆傅里叶逆变换到时域，并与输入信号进行卷积操作；

C. 观察结果；

D. 将 B 步骤所使用的输入信号进行傅里叶变换，然后与 A 中自行设计的滤波器进行乘法滤波，并将得到的结果进行傅里叶逆变换回去；

E. 比照 C 的结果和 D 的结果；

三、实验核心公式或核心问题：

部分验证、分析性实验，请简要给出该实验涉及的核心公式或核心问题，并用少量文字适当描述。部分实验可能没有公式。

离散傅里叶变换 DFT 公式：$X[k]=\frac{1}{n}\sum_n x[n]e^{-\frac{j2k\pi}{N}}$

离散傅里叶逆变换 IDFT 公式：$x[n]=\sum_K X[k]e^{\frac{j2k\pi}{N}}$

由$x[n]\Rightarrow X[k]$称为（离散（时间））傅立叶变换；而由$X[k]\Rightarrow x[n]$称为傅立叶逆变换。

由公式很容易就能写出朴素 DFT 的代码。

四、实验设计

务必表明那些工作为手动编写完成，哪些工作是通过调库实现。

1. 傅里叶变换部分实验

傅里叶变换 DFT 输入为一个序列，经过傅里叶变换得到的结果作为傅里叶逆变换的输入。傅里叶逆变换得到的结果和原序列进行比对即可。

要比较不同数据规模下手工朴素 DFT 算法和调库快速傅里叶变换算法 np.fft.fft(x)的时间消耗，使用 timeit 函数即可统计时间，然后绘制成图打印出来。

2. 概念滤波器设计部分

本部分实验将设计一个低通滤波器。

手工编写 get_lowpass_mask 获取低通遮罩，filt_by_frequency 通过频率滤波，filt_by_timedomain 将该滤波器通过逆傅里叶逆变换到时域，并与输入信号进行卷积操作。比较两种结果是否相同。

五、代码及结果展示及分析：

代码应有至少一段文字说明，说明为两个方面：一，这个代码的作用；二、为什么这段代码是核心代码（对比工作 3 中的核心公式或核心问题）

手写离散傅里叶变换及手写离散傅里叶逆变换

```
def fft1(src, dst = None):
 l = len(src)
 n = int(math.log(l,2))
 bfsize = np.zeros((l), dtype = "complex")
 for i in range(n + 1):
     if i == 0:
         for j in range(l):
             bfsize[j] = src[Dec2Bin_Inverse2Dec(j, n)]
```

```
        else:
            tmp = copy.copy(bfsize)
            for j in range(l):
                pos = j%(pow(2,i))
                if pos < pow(2, i - 1):
                    bfsize[j] = tmp[j] + tmp[j + pow(2, i - 1)] *
np.exp(complex(0, -2*np.pi*pos/pow(2,i)))
                    bfsize[j + pow(2, i - 1)] = tmp[j] - tmp[j + pow(2, i -
1)] * np.exp(complex(0, -2*np.pi*pos/(pow(2,i))))
    return bfsize
    def ifft1(src):
    for i in range(len(src)):
        src[i] = complex(src[i].real, -src[i].imag)

    res = fft1(src)

    for i in range(len(res)):
        res[i] = complex(res[i].real, -res[i].imag)
    return res/len(res)
```

这段代码为手写傅里叶变换及傅里叶逆变换的核心代码，根据公式直接得到，理论推算时间复杂度为 $\Theta(n^2)$。

滤波器设计代码：

```
    def cov_in_time(a, b):
        c = np.tile(b, 3)
        ma = max(len(a), len(b))
        cov = np.convolve(a, c)
        r = cov[ma:2 * ma]
        return r
    def get_lowpass_mask(l, k):
        a = np.zeros_like(l, np.double)
        c = int(len(l) / 2)
        a[c - k:c + k] = 1
        return a
    def filt_by_frequency(s, mask):
        f = np.fft.fft(s)
        r = f * mask
        return np.fft.ifft(r)
    def filt_by_timedomain(b, mask):
        ift = np.fft.ifft(mask)
        return cov_in_time(b, ift)
    def filter():
        l = 50
        k = 10
        rang = np.arange(l)
        #print(rang)
        s = np.sin(rang)+2*np.cos(rang)
        mask = get_lowpass_mask(s, k)
        print(mask)
        res_byf = filt_by_frequency(s, mask)
        res_byt = filt_by_timedomain(s, mask)
        fig, subs = plt.subplots(2, 2)
        subs[0][0].plot(s)
        subs[0][1].plot(mask)
        subs[1][0].plot(res_byf)
        subs[1][1].plot(res_byt)
```

```
        plt.show()
    return
```

结果与分析：

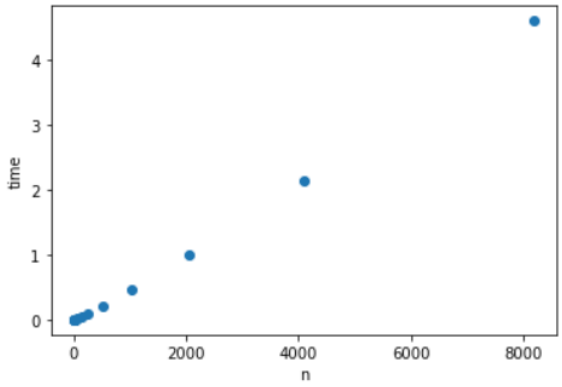
手写傅里叶变换结果正确性验证：

对一串随机值组成的序列（src = np.random.randint(20,size=8)）分别用手写 DFT 和系统调用的 DFT 进行变换后，再将变换的结果逆变换回去，得到的结果截图如下：

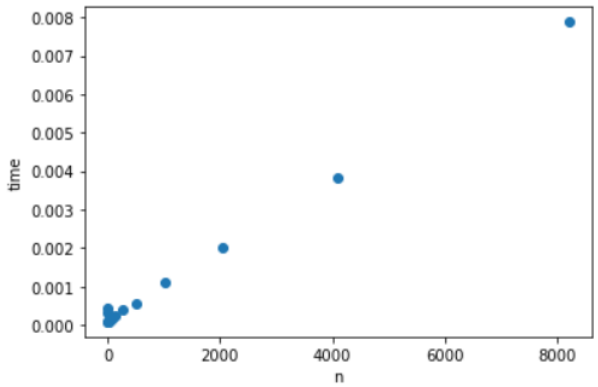
```
任务一：手工实现fft和ifft并进行比对
自己实现的fft:
[74.          +0.j          6.77817459-10.12132034j
  5.          -5.j         -8.77817459 +5.87867966j
 -8.          +0.j         -8.77817459 -5.87867966j
  5.          +5.j          6.77817459+10.12132034j]
自己实现的ifft:
[ 9.-0.00000000e+00j 15.+3.67548785e-16j 11.-2.44929360e-16j
  7.+5.20629635e-16j 10.-0.00000000e+00j  8.-5.20629635e-16j
  3.+2.44929360e-16j 11.-3.67548785e-16j]
np里面的fft:
[74.          +0.j          6.77817459-10.12132034j
  5.          -5.j         -8.77817459 +5.87867966j
 -8.          +0.j         -8.77817459 -5.87867966j
  5.          +5.j          6.77817459+10.12132034j]
np里面的ifft:
[ 9.+0.j 15.+0.j 11.+0.j  7.+0.j 10.+0.j  8.+0.j  3.+0.j 11.+0.j]
```

手写傅里叶变换与调用系统的快速傅里叶变换耗时比较：

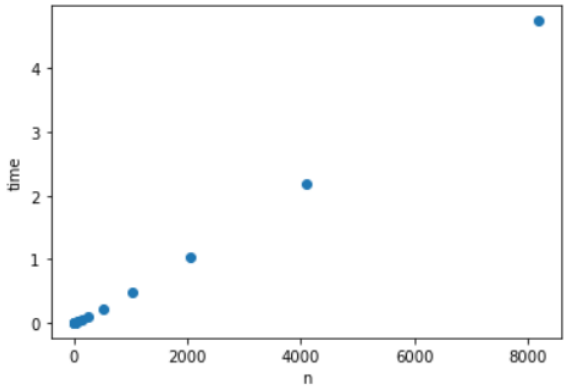
手写傅里叶变换数据规模和耗时的关系如下：



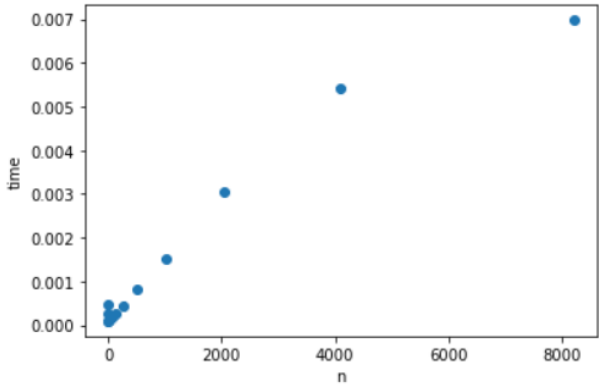
快速傅里叶变换数据规模和耗时的关系如下：



手写傅里叶逆变换数据规模和耗时的关系如下：



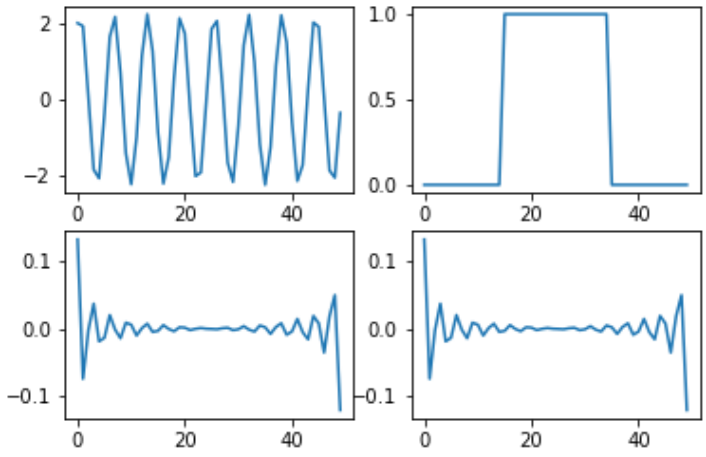
快速傅里叶逆变换数据规模和耗时的关系如下：



可以看出系统调用的快速傅里叶变换的时间消耗远远小于手写的傅里叶变换。这与理论推导出的时间复杂度（快速傅里叶变换 $\Theta(n \log n)$，普通傅里叶变换 $\Theta(n^2)$）的结论基本相符合。

本部分实验不符合预期的情况：手写傅里叶变换的时间和数据规模的关系图像并不是很明显的二次函数的形状，而像线性的形状。经过是因为测试的数据规模太小导致的。

低通滤波器实验结果：



其中左上角图片为原信号，右上角图片为为屏蔽遮罩。左下角图片为将该滤波器通过逆傅里叶逆变换到时域，并与输入信号进行卷积操作得到的结果。右下角图片为将输入信号进行傅里叶变换，然后用滤波器进行乘法滤波，并将得到的结果进行傅里叶逆变换回去。比对图片，分析实验结果，可以发现两者完全一致，符合预期。