计算方法（A）

实验（二） 实现FFT

姓　　名	熊恪峥
学　　号	22920202204622
日　　期	2022年4月5日
学　　院	信息学院
课程名称	计算方法（A）

实验（二） 实现FFT

目录

1 原理

傅里叶变换将信号的时域采样变换到频域，这样就可以对信号进行操作，比如加噪声、滤波、抽样等。通过傅里叶变换，可以在图像的频域增加信息，用于追溯信息泄露等工作。快速傅里叶变换（FFT）是一种用于快速计算离散傅里叶变换（DFT）的算法。它借助了$\omega = e^{-\frac{2\pi i}{N}}$的对称性 (1)

$$\omega_N^{jk+N/2} = -\omega_N^{jk} \tag{1}$$

可以得到 (2)和 (3)

$$c_{2j} = \sum_{k=0}^{N/2-1} \left(x_k + x_{N/2+k}\right) w_{N/2}^{jk}, \tag{2}$$

$$c_{2j+1} = \sum_{k=0}^{N/2-1} \left(x_k - x_{N/2+k}\right) w_N^k w_{N/2}^{jk} \tag{3}$$

因此可以由附录A：实现代码中的代码 1递归地计算FFT，它的时间复杂度是$\mathcal{O}(N\log N)$。这叫做Cooley-Tukey算法。其中$padding$函数可以用来在输入点数不是2的倍数时填充0，以使得输入点数变成2的倍数。它使用了位运算技巧来获取离 x 长度最接近的2的幂。

2 结果

对$y = x^2 \cdot \cos x$在$[-\pi, \pi]$上逼近，等距离取16个点，逼近结果和函数$y = x^2 \cdot \cos x$如图 2。可见FFT逼近效果较好，误差主要出现在区间两端点附近。其余部分与函数较为相近。进一步增加点数可以使得逼近效果更佳。

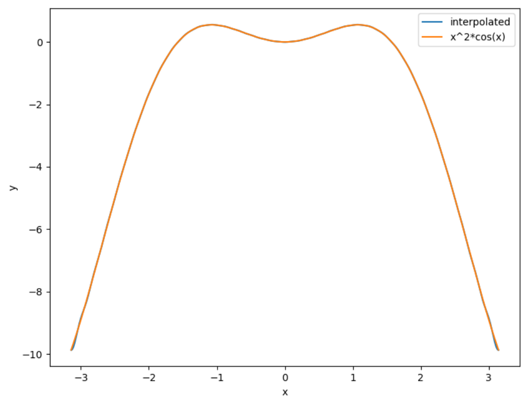


图 1: 逼近结果和$y = x^2 \cdot \cos x$

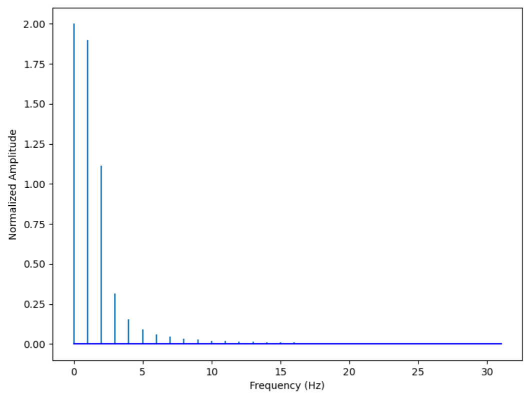


图 2: 逼近结果的频率和幅度

FFT使用三角函数的组合对函数进行逼近，各三角函数的频率和幅值如图 2。图示幅值经过(4) 的归一化处理来减少幅值绝对值之间的差，便于展示。

$$A_{normalized} = \frac{|A|}{N/2} \tag{4}$$

实践表明增加取点个数可以很好地提高FFT逼近函数的效果。

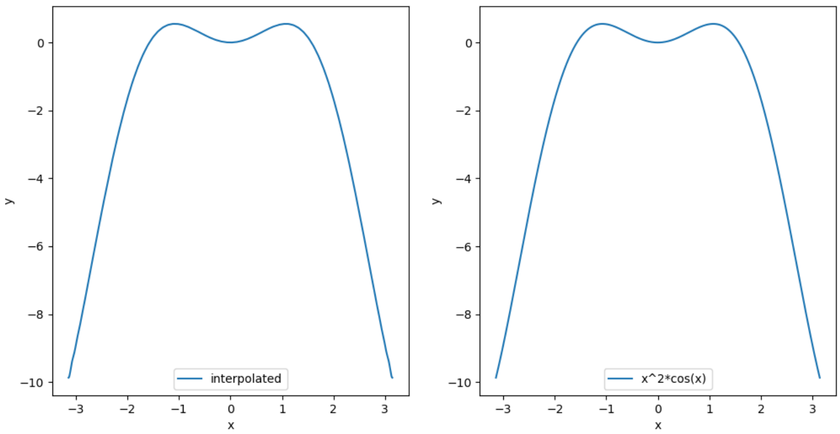
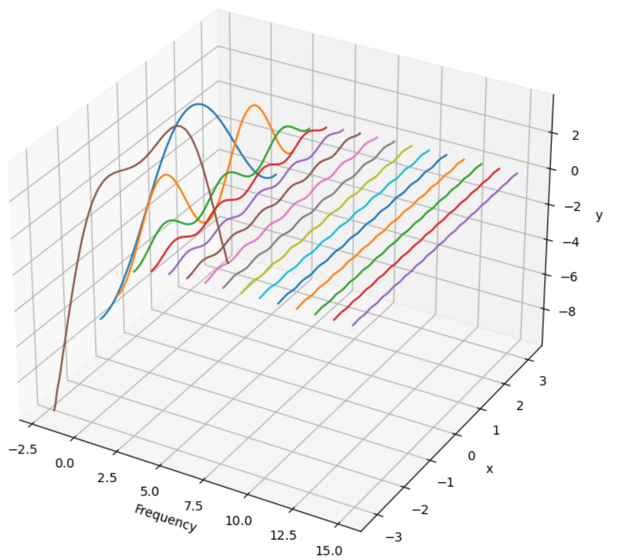


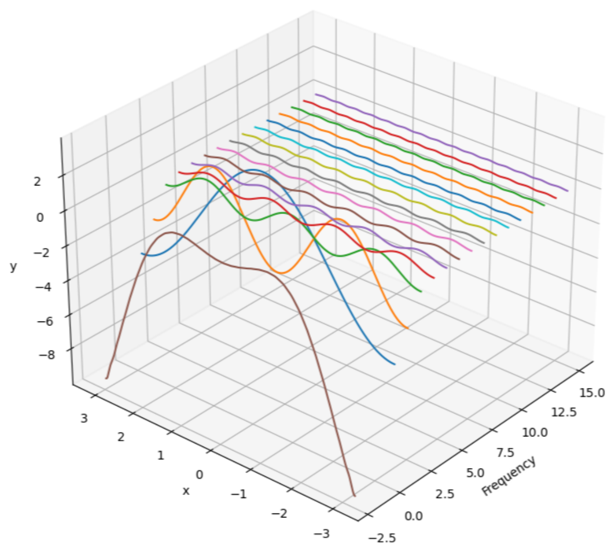
图 3: 取点数$N = 64$逼近结果和$y = x^2 \cdot \cos x$

如图 2取$N = 64$，可见逼近曲线光滑、与函数曲线较为相近。进一步提高了逼近效果。

将这些三角函数和函数$y = x^2 \cdot \cos x$画在同意坐标系中如图，$y = x^2 \cdot \cos x$人为画在$f = -2.5$处，可以直观地展现出FFT将函数分解为频率不同、幅值不同的三角函数的作用。



(a) 视角1



(b) 视角2

图 4: FFT结果的各频域分量

A 附录A：实现代码

代码 1: FFT实现

```
def padding(x: np.ndarray) -> np.ndarray:
    n:  int = x.shape[0]

    if (n & (n - 1)) == 0:
        return x

    n -= 1
    n |= n >> 1
    n |= n >> 2
    n |= n >> 4
    n |= n >> 8
    n |= n >> 16
    n += 1

    return np.pad(x, (0, n - x.shape[0]))

def fft(x: np.ndarray) -> np.ndarray:
    x = padding(x)
    N: Final[ float] = x.shape[0]

    if N == 1:
        return x
    else:
        even = fft(x[::2])
        odd = fft(x[1::2])

        fact = np.exp(-2j * np.pi * np.arange(N) / N)

        return np.hstack((even + fact[: int(N / 2)] * odd, even + fact[ int(N / 2):] * odd))
```