快速傅里叶变换

Author[a,1]

[a] 武汉大学，物理科学与技术学院

Keywords—快速傅里叶变换，贝塞尔函数，*Python*

Contents

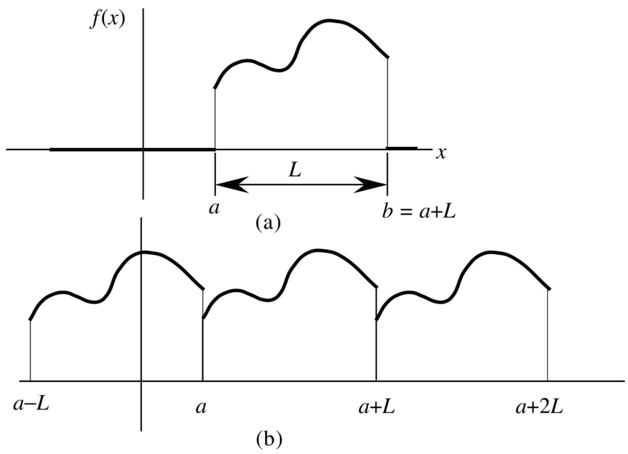

Figure 1. (a) The function we want to represent. (b) The Fourier series representation of the function

1. Introduction

数学物理中最常出现的主题是傅里叶分析。例如，它出现在经典力学和非简正模的分析中，出现在电磁理论和波的频率分析中，出现在噪声考虑和热物理中，出现在量子理论和动量与坐标表示之间的转换中，出现在相对论量子场论中，出现在非正则态的产生和湮灭运算中.[1] 在介绍快速傅里叶变换的算法之前，我们首先简单回顾一下傅里叶变换的定义。

1.1. 傅里叶级数

在关于贝塞尔函数的迭代计算作业中，我们提到了魏尔斯特拉斯定理，它告诉我们任意关于 x 的连续函数都可以展开成 x 的代数多项式。这个结论可以推广到 n 个自变量的情形：

Generalized Stone-Weierstrass Theorem

如果函数 $f(x_1, x_2, \cdots, x_n)$ 是区间 $[a_i, b_i]^n$ 上的连续函数，那么 $f(x_1, x_2, \cdots, x_n)$ 可以展开成多项式 $x_1^{k_1} x_2^{k_2} \cdots x_n^{k_n}$, k_i是非负整数.

由 $x = r\cos\theta, y = r\sin\theta$，我们得到了极坐标下展开一个二元函数的方法：

$$f(r,\theta) = \sum_{m,k=0}^{\infty} a_{mk} x^k y^m = \sum_{m,k=0}^{\infty} a_{m,k} r^{m+k} \cos^k\theta \sin^m\theta \tag{1}$$

如果我们令 $r = 1$（或者其他常数）就得到了一个只关于 θ 的函数：

$$f(\theta) = \sum_{n=-\infty}^{\infty} b_n e^{in\theta} = b_0 + \sum_{n=1}^{\infty} (A_n \cos\theta + B_n \sin\theta) \tag{2}$$

这是一个周期为 2π 的周期性函数，它可以表示 $[-\pi, \pi]$ 上的连续函数（或者分段连续函数，准确地讲是满足狄利克雷条件的函数）。利用正交性和归一性，不难求出 [1]

$$b_n = \frac{1}{\sqrt{2\pi}} \int_{-\pi}^{\pi} e^{-in\theta} f(\theta)\,\mathrm{d}\theta \tag{3}$$

但是，我们感兴趣的函数并不总是定义在 $-\pi, \pi$ 上。更一般的，对于定义在 (a, b) 上，周期为 L=(b-a) 的函数 F(x), 通过换元

$$\theta = \frac{2\pi}{L}(x - a - \frac{L}{2})$$

可以得到

$$F(x) = \frac{1}{\sqrt{L}} \sum_{-\infty}^{\infty} F_n e^{2\pi i n x/L} \tag{4}$$

$$F_n = \frac{1}{\sqrt{L}} \int_a^b e^{-2\pi i n x/L} F(x)\,\mathrm{d}x \tag{5}$$

1.2. 傅里叶变换

傅里叶级数可以展开任意周期性函数 F(x), 但是物理学中遇到的大多数函数并不是周期性的。一种简单的想法是使用周期 $L \to \infty$, 此时 $\Delta\omega = \frac{2\pi}{L} \to 0$。[2] 我们直接给出傅里叶积分的定义，为了避免和频率 f 混淆，我们改用 h(t) 表示函数。

$$H(\omega) = \int_{-\infty}^{\infty} h(t) e^{i\omega t}\,\mathrm{d}t \tag{6}$$

$$h(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} H(\omega) e^{-i\omega t}\,\mathrm{d}\omega \tag{7}$$

2. 快速傅里叶变换

2.1. Sampling Theorem and Aliasing

在通常情况下，h(t) 是通过均匀采样得到的，令 Δ 表示两次相邻采样的时间间隔，称为采样率 (sample rate)。对任何采样率 Δ，有一个相对应的临界频率，称为 Nyquist 临界频率（Nyquist critical frequency）[3]

$$f_c = \frac{1}{2\Delta} \tag{8}$$

关于二者有一个重要的定理：

sampling theorem

If a continuous function h(t), sampled at an interval Δ, happens to be *bandwidth* limited to frequencies smaller in magnitude than f_c, i.e., if $H(f) = 0$ for all $|f| \geq f_c$, then the function h(t) is completely determined by its samples h_n.

2.2. 离散傅里叶变换（Discrete Fourier Transform）

对于周期为无穷的函数 h(t), 我们只需要对有限的的部分进行采样 (h(t) 等于零的部分积分仍然是 0)，我们将 h(t) 开始的初始时刻记为 $t=0$，

$$h_k = h(k\Delta), \qquad k = 0, 1, \cdots, N-1 \tag{9}$$

利用这 N 个输入的 h_k，我们可以得到

$$H(f_n) = \int_{-\infty}^{\infty} h(t) e^{2\pi i f_n t}\, \mathrm{d}t = \int_{0}^{(N-1)\Delta} e^{2\pi f_n k\Delta}\, \mathrm{d}t \tag{10}$$

根据前面的叙述，为了将 h(t) 用 $h(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} H(\omega) e^{-i\omega t}\, \mathrm{d}\omega$ 表示，h(t) 的周期看作无穷，$\Delta\omega = \frac{2\pi}{L} \to 0$, 我们应当取无数个 f_n 的值。在推导傅里叶积分的过程中，这当然是没有问题的，$\Delta\omega = \frac{2\pi}{L} \to 0$ 意味着求和被积分所取代，但当我们实际进行数值计算时却不能取无穷个 f_n。出于对称性的考虑，我们同样也取 N 个 f_n 的值，这样可以重复利用一部分代码

$$f_n = \frac{n}{N\Delta}, \qquad n = -\frac{N}{2}, \cdots, \frac{N}{2} \tag{11}$$

这样 f_n 刚好均匀分布在 $-f_c, f_c$。f_n 取更大的值没有意义，因为如采样定理所描述，在当前的 Δ 下，提取不到更高频率的信号的信息（从实际操作过程看，我们的 Δ 是根据信号中存在的频率的上限选取的，f_c 是采样的信号中存在的最大的频率，信号中本来不具备比 f_c 频率更高的信号）。

引入记号 $H_n = H(f_n)\Delta$。

$$H_n = \sum_{k=0}^{N-1} h_k e^{2\pi i k n/N} = \sum_{k=0}^{N-1} h_k W_N^{nk} \tag{12}$$

其中 $W_N = e^{2\pi i/N}$ 不难验证 $H_{-n} = H_{N-n}$, 所以

$$h_k = \frac{1}{N}\sum_{n=0}^{N-1} H_n e^{-2\pi i k n/N} \tag{13}$$

2.3. 快速傅里叶变换（Fast Fourier Transform）

由公式12可知，进行离散傅里叶变换相当于一个 $N \times N$ 矩阵和一个 N 维向量相乘，所需要的时间为 $O(N^2)$, 利用单位复数根的性质可以将时间缩短为 $O(N \log N)$[4]。

单位复数根

单位复数根是满足 $W^N = 1$ 的复数 W，这样的复数刚好有 N 个，$e^{2\pi i n/N}, n = 0, 1, \cdots, N-1$。它们均匀地分布在以原点为中心的单位圆上。$W_N = e^{2\pi i/N}$ 称呼为 N 次主单位根。其它根都是 W_N 的幂。

N 个 N 次单位复数根以乘法为群乘法构成一个群，与加法群同构，我们给出单位复数根的几个基本性质：

1. $W_N^j W_N^k = W_N^{j+k} = W_N^{(j+k) \bmod N}$
2. 消去引理：$W_{mN}^{mk} = W_N^k$
3. 折半引理：N 个 N 次单位复数根的平方的集合就是 N/2 个 N/2 次单位复数根的集合：$W_N^{k+N/2} = -W_N^k, (W_N^k)^2 = W_{N/2}^k$

现在回到计算 H_n 的问题上。我们采取分治策略，利用 H_n 的偶数项（even）和奇数项 (odd) 分别定义 H^e 和 H^o.

$$\begin{aligned} H_n(h, W_N^n) =& h_0 + W_N^n h_1 + W_N^{2n} h_2 + \cdots + W_N^{(N-1)n} h_{N-1} \\ =& (h_0 + W_N^{2n} h_2 + \cdots + W_N^{(N-2)n} h_{N-2}) + \\ & W_N^n (h_1 + W_N^{2n} h_3 + \cdots + W_N^{(N-2)n} h_{N-1}) \\ =& H_n(h^e, W_{N/2}^n) + W_N^n H_n(h^o, W_{N/2}^n) \\ =& H_n^e + W_N^n H_n^o \end{aligned} \tag{14}$$

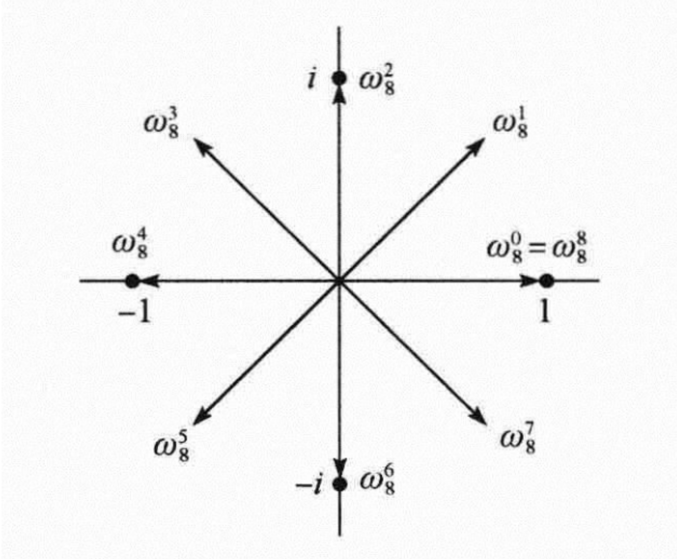

Figure 2. 8 次单位复数根

这样就把求 $H_n, n = 0, 1, 2, \cdots, N-1$ 的问题转化为求 H_n^e 和 $H_n^o, n = 0, 1, 2, \cdots, N/2-1$ 的问题。(折半引理)。这两个子问题的形式与原始问题相同。重复以上步骤，我们可以得到：(一个元素的 DFT 就是它自身)

$$H_n^{eoooee\cdots} = h_k \tag{15}$$

那么，n 和 k 之间有什么关系呢？我们来回顾刚刚的过程，h_k 被归入 H^e 系数的原因是 k 是偶数，将 k 写成二进制，k 的末位是 0；而末位是 1 的 k 对应的 h_k 则被归入 h^o。然后对分出的两组，重复以上过程，看第二末位是 0 还是 1。这个过程和判断二进制数的大小是相似的，当我们判断二进制数大小时，首先看最高位是 0 还是 1，然后将所有数分为较小的半组和较大的半组，然后对第二高位重复这个过程。因此我们只需要将 k 转为二进制，然后左右翻转，对 $bit\_rev(k)$ 按大小进行排序。在对 h_k 在二进制倒序之后，

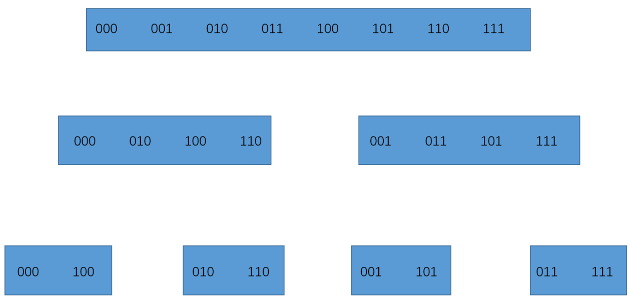

Figure 3. 对 2^M 个数按奇偶划分直至每组只有一个，相当于将下标按二进制反转，并从小到大排列

就可以按照迭代结构而不是递归结构计算。 N=2^M 时要进行蝶形运算，我们要解决的问题有：

1. 两个输入数据之间的间隔 B
2. 旋转因子 W 的确定，包括：
   (a) 第 L 级旋转指数
   (b) 第 L 级 W 的种类确定
   (c) 第 L 级中同一 W 之间的间隔

观察蝶形图可知第 L 级：

1. 两个输入数据间距为 $B = 2^{l-1}$
2. 有 2^{L-1} 个旋转因子
3. 旋转因子 W 增量为 2^{M-L}

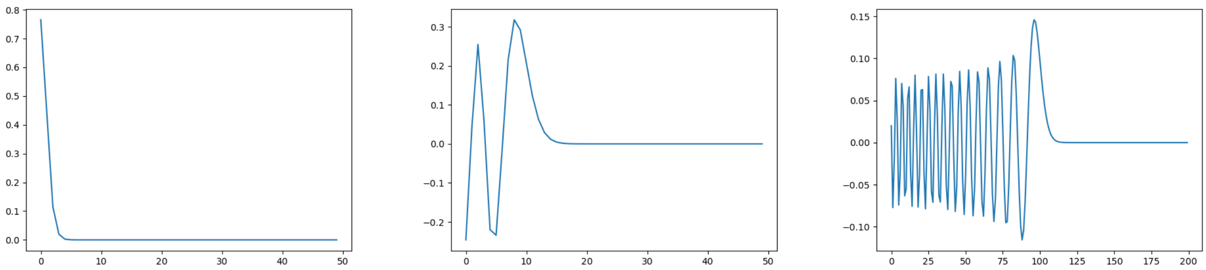

Figure 4. z = 1, 10 and 100 时的 $J_n(z)$

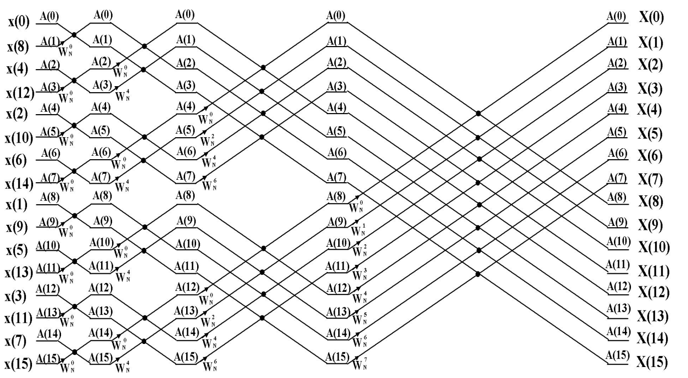

Figure 5. 在二进制倒序之后，就可以按照迭代结构而不是递归结构计算。

4. 同一 W 间隔为 $istep = 2B = 2^L$
5. 同种蝶形运算次数为 2^{M-L}

3. 使用 FFT 算法计算 Bessel 函数

利用 FFT 程序可以按照如下的积分计算 Bessel 函数:

Bessel function

$$J_n(z) = \frac{i^{-n}}{2\pi}\int_0^{2\pi} e^{izcos\theta}e^{in\theta}d\theta \tag{16}$$

Bessel 函数的积分可以写成如下的 FFT 形式:

$$\begin{aligned} J_n(z) =& \frac{i^{-n}}{N}\sum_{k=0}^{N-1} e^{izcos(2\pi k/N)}e^{2\pi ink/N} \\ =& \frac{i^{-n}}{N}H_n[e^{izcos(2\pi k/N)}, W_N^n] \end{aligned} \tag{17}$$

4. Codes

```
import numpy  as np
import matplotlib.pyplot as plt

def DataRev(data):
    N=len(data)
    data_rev=[]
    width=int(np.log(N)/np.log(2))#计算位宽width，即 nmax 表示的二进制数的位数。
    for i in range(0,N):
        j= '{:0{width}b}'.format(i, width=width)#使用格式化字符串'{:0{width}b}'将整数 i 转换为 width 位的二进制字符串，前面用 0 补齐。
        data_rev.append(data[int(j[::-1], 2)])#将生成的二进制字符串 j 反转（j[::-1]），然后将其转换回整数并返回。
    return data_rev
def Coeff(N,k,isign):
    return  np.exp(isign*2*np.pi*1j*k/N)
def FFT(data,isign):
    data=np.array(data)
    data=DataRev(data)
    Length=len(data)
    Width=int(np.log(Length)/np.log(2))
    for i in range(Width):
        for j in range(0,Length,2**(i+1)):
            for k in range(0,2**i):
                x=k+j
                y=x+2**i
                W = Coeff(2**(i+1),k,isign)
                data[x],data[y]= data[x]+W*data[y], data[x]-W*data[y]
    if isign==-1:
        i=0
        while(i<Length):
            data[i]=data[i]/Length
            i+=1
    return data
def bessl(z,M):
    N=2**M
    data=[]
    for i in range(0,N):
        data.append(np.cos(z*np.cos(2*np.pi*i/N))+np.sin(z*np.cos(2*np.pi*i/N))*1j)
        #data.append(np.exp(z*np.cos(2*np.pi*i/N)))
    data=FFT(data,1)
    for i in range(0,N):
        data[i]=data[i]/(N*(1j)**i)
        data[i]=data[i].real
    return data
if __name__=='__main__':
    J1=bessl(1,12)
    j1=[]
    xx=np.arange(0,50)
    for i in range(0,50):
        j1.append(J1[i])
    plt.plot(xx,j1)
    plt.show()
```

Code 1. code

References

[1] S. Hassani, *Mathematical Physics*. Springer, 2013.

[2] 姚端正 and 周国全 and 贾俊基, 数学物理方法. 科学出版社, 2019.

[3] Richard L. Burden and J. Douglas Faires, *Numerical Analysis*. Richard Stratton, 2010.

[4] T. H.Cormen and C. E.Leiserson, 算法导论, 殷建平等, Ed. 机械工业出版社, 2013.

[5] *PGFPlots - A LaTeX package to create plots.* [Online]. Available: https://pgfplots.sourceforge.net/.