基于 Rader 算法实现 FFT

麦金杰 16337179　刘上华 16337157　麦显忠 16337180
唐诗佳 16337221　田晓晴 16337223

摘要 本文基于 Rader 算法实现 FFT，与 matlab 自带函数包 FFT 进行结果比对，验证所实现 Rader 算法的正确性，最后与实现的 DFT 及指基 2 的 Cooley-Turkey 快速傅立叶变换进行复杂度比较。

关键词：FFT ，Rader 算法，卷积

1. 文献综述

J.W.Cooley 和 J.W.Tukey 提出快速傅立叶算法[1]，按照基本蝶形运算的构成，将尺寸 N=N1*N2 重新表达为在 N1 尺寸下更小的 N2，将每种算法分为基 2、基 4、基 8、基 16 及任意因子的 FFT 算法。一般来说基数越高总计算量越少，但是考虑算法的复杂性，基 2、基 4 是目前使用最广泛的算法。

Duhamel 和 Hollmann 提出分裂基快速算法[2]，该算法基本思路是对偶序号输出使用基 2 算法，奇序号输出使用基 4 算法。分裂基算法只能应用于 N 是 4 的倍数，但算法可以形成更小的 DFT 从而与任何其他的 FFT 算法根据需要进行组合。在已知的 $N=2^M$ 的各种算法中，分裂基算法所需的乘法数最小，并接近理论上的最小值。

Kolba 和 Parks 提出素质数的 DFT 算法——素因子算法（PFA）[3].这种算法用中国余数定理映射，将 N=N1N2 一维 DFT 变换为多维 DFT，当 N1 和 N2 两个为互素因子时，大小为 N1N2 的二维 DFT 可以变换为 N1 个 N2 点一维 DFT 和 N2 个 N1 点一维 DFT，比 FFT 所需算法次数少[4]。

本文实现的 Rader 算法[5]，通过将 DFT 重新表达为循环卷积来计算素数大小的离散傅里叶变换（DFT），由于 Rader 算法只取决于 DFT 内核的周期性。因此可直接应用具有素数阶任何变换。

2. 原理技术

本次项目中我们复现 Rader's FFT，这种方法是计算素数长度的离散傅里叶变换。其思想是通过循环卷积重新表达 DFT，从而将素数情形转换为非素数情形。

2.1 Rader 算法

Rader 算法的基本思想是利用 DFT 矩阵 F_p 中 ω_p 的幂级数的特殊排列结构将 DFT 形式

$$H_n = \sum_{k=0}^{N-1} h_k e^{2\pi ikn/N}, n = 0,1,2,..., N-1 \qquad ①$$

以卷积形式重构，然后利用非素数长度的 FFT 进行计算。下面将详细介绍计算步骤。[6]

（1）将 h_0 从①式中分离，令 $\omega_p = e^{\frac{2\pi i}{N}}$，得到表达式为

$H_0 = \sum^{p-1} h_p$ ，

$$H_n = h_0 + \sum^{p-1} h_k \omega_p^{nk}, n = 1,2,..., p-1 \quad ②$$

（2）求解模 p 单位群的生成因子

对 ω_p 构成的矩阵 F_p, 我们设法将其转换为循环矩阵。

设 $\mathbf{Z}_p^{\times}$表示模 p 单位群，即 $\mathbf{Z}_p^{\times}$关于模 p 剩余类的乘法构成群。对于素数 p，1,2,...,p-1 均在 $\mathbf{Z}_p^{\times}$中。

对于循环群 $\mathbf{Z}_p^{\times}$，我们要找到其生成因子 g(1≤g≤p-1，g∈**Z**)。即对于任意 a∈$\mathbf{Z}_p^{\times}$，存在唯一的一对 *i, j* (*i,j*∈[0,*p*-2]), $a=g^i(mod\ p)$且 $a=g^{-j}(mod\ p)$，其中 g^{-j} 表示 *g* 的模 *p* 乘积逆元的 j 次幂，即 $g^{-j}=(g^{-1})^j$。

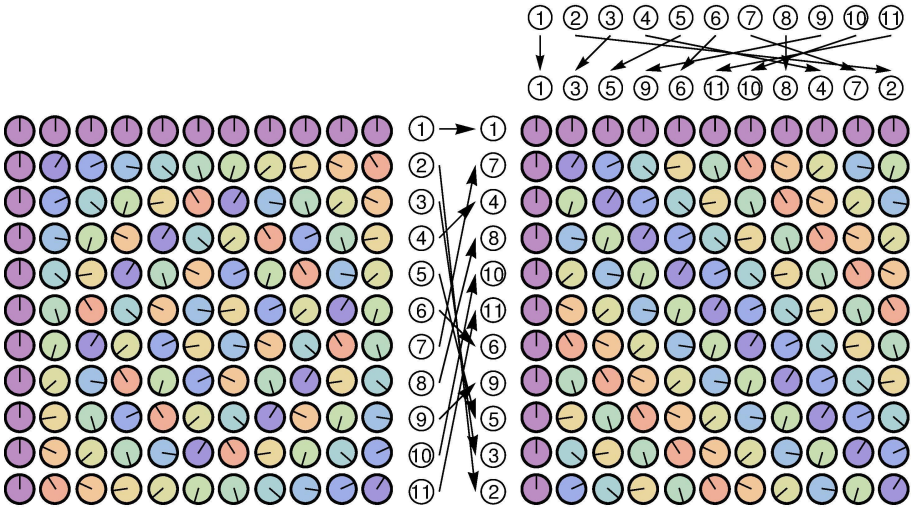

图 1

例如，图 1 为秩为 11 的 DFT 矩阵通过行列变换转换为循环矩阵的直观表示。其中 p=11,g=2.对一维数组，与循环矩阵相乘等价于循环卷积。[7]

（3）卷积形式重构

确定群 $Z_p^{\times}$的生成因子 g，等式②可以被重写为

$$H_{g^{-r}} = h_0 + \sum_{q=0}^{p-2} h_{g^q} \omega_p^{g^{q-r}} = h_0 + \sum_{q=0}^{p-2} h_{g^q} \omega_p^{g^{-(r-q)}} \quad ， r=0,1,...,p-2 \quad ③$$

这里用到 h_k，H_k 都和 p 互素以及$\omega_p^p=1$ 的性质。

设长度为 p-1 的序列 $a_q = h_{g^q}, b_q = \omega_p^{g^{-q}}$ (0≤q≤p-2),③中的求和部分可以表示为 a_q 与 b_q 的卷积形式。

（4）计算 FFT

令 $\widetilde{h} = a_q = h_{g^q}$， $\widetilde{\omega}_p = b_q = \omega_p^{g^{-q}}$， $\widetilde{H} = H_{g^{-r}}$ 则

$$\widetilde{H} - h_0 = IDFT[DFT[\widetilde{h}] \cdot DFT[\widetilde{\omega}_p]] \quad ④$$

从而求得原表达式结果。

2.2 局限性及改进

Rader’s FFT 的不足之处在于若对于素数 p，p-1 具有很大的素因子时，需要

递归调用 Rader's FFT, 这可能会比直接递归调用传统 FFT 更低效。

为了解决这一问题，可以采用补零的方法。设有一维数组 *f[i]*和 *g[i]*,长度均为 M-1(M 为素数),两序列卷积形式为 $(f*g)[n]=\sum_{m=0}^{M-1} f[m]g[n-m]$，其中序列下标均从 0 开始且在模 M 剩余系中。

我们构造长度为 M'(M'≥2M-3)的新序列 *f'[j]*和 *g'[j]*，在实现时我们取 M'=2^n。*f'[j]*通过在 *f[0]*和 *f[1]*之间填充 M'-M 个 0 得到，*g'[j]*通过循环重复 *g[j]*中的值填充至长为 M'。可以证得(*f'*g'*)[n]=(*f*g*)[n].

3. 实验结果

本次实验中实现的是一维 FFT。我们采用不同的素数 p 进行测试，将自己实现的 Rader 算法、利用 Matlab 内置 FFT 实现的结果，以及实现的作差进行比较。如图 2 所示，验证了 Rader 算法对于素数长度的傅里叶变换运算的正确性。

本次实验还进行了一维 DFT、一维 FFT（Matlab 内置函数）、一维 rader 算法的运行时间比较，如图 3 所示，DFT 具有 O(N^2)的时间复杂度，而 Matlab 内置 FFT 和 rader 算法时间几乎重合；去除 DFT 干扰，如图 4 所示，rader 算法运行时间大于直接调用 Matlab 内置的 FFT 运行时间。

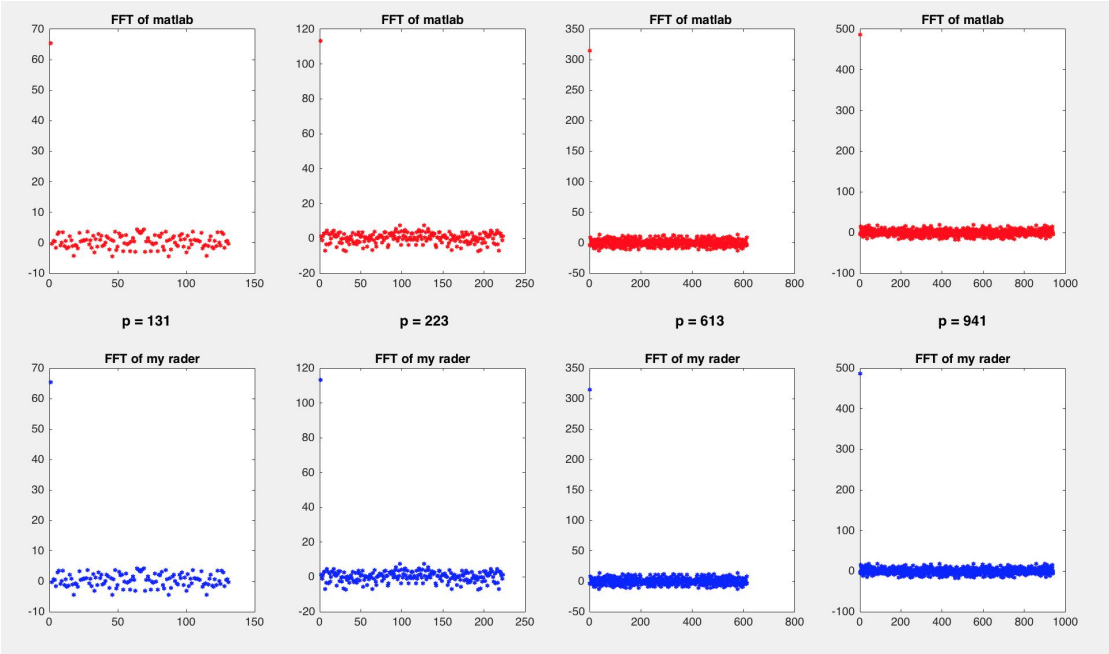

图 2. matlabFFT 与 Rader's FFT 结果对比图

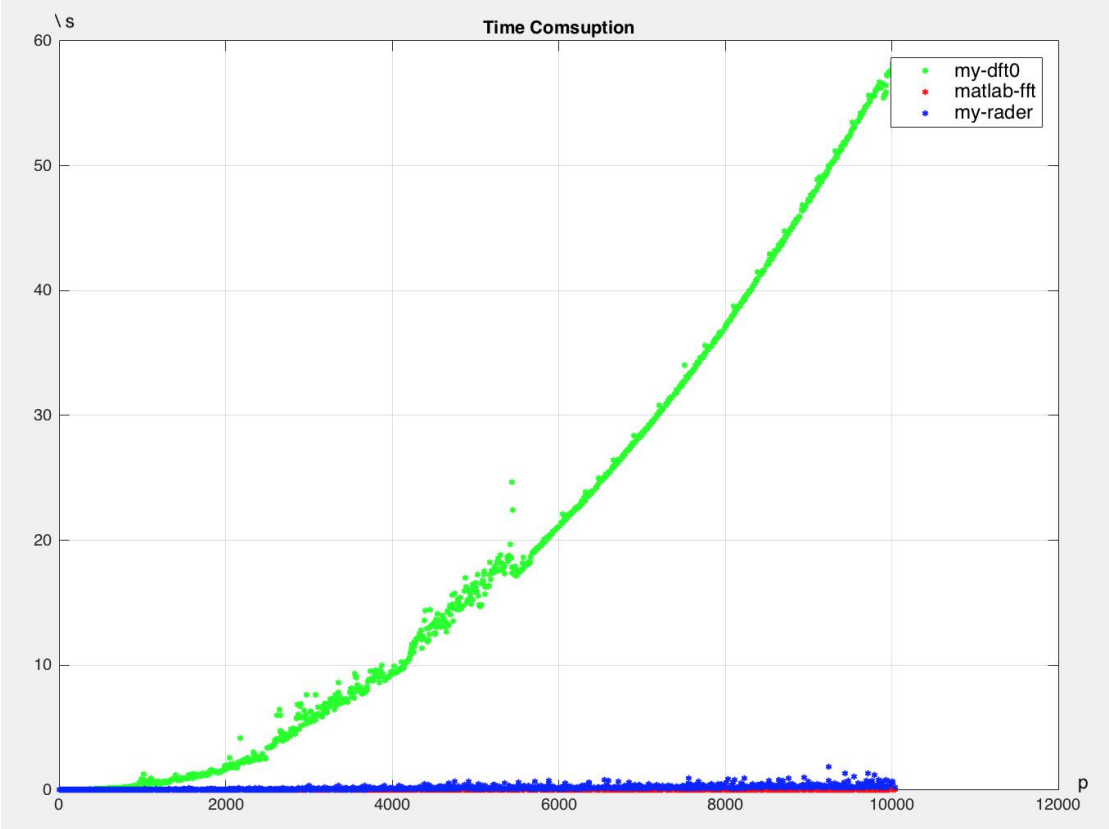

图 3.测试 Rader 算法与 matlab 内置 FFT 以及实现的 DFT 运行时间差异，横坐标为素数 p，纵坐标为 p 时的运行时间

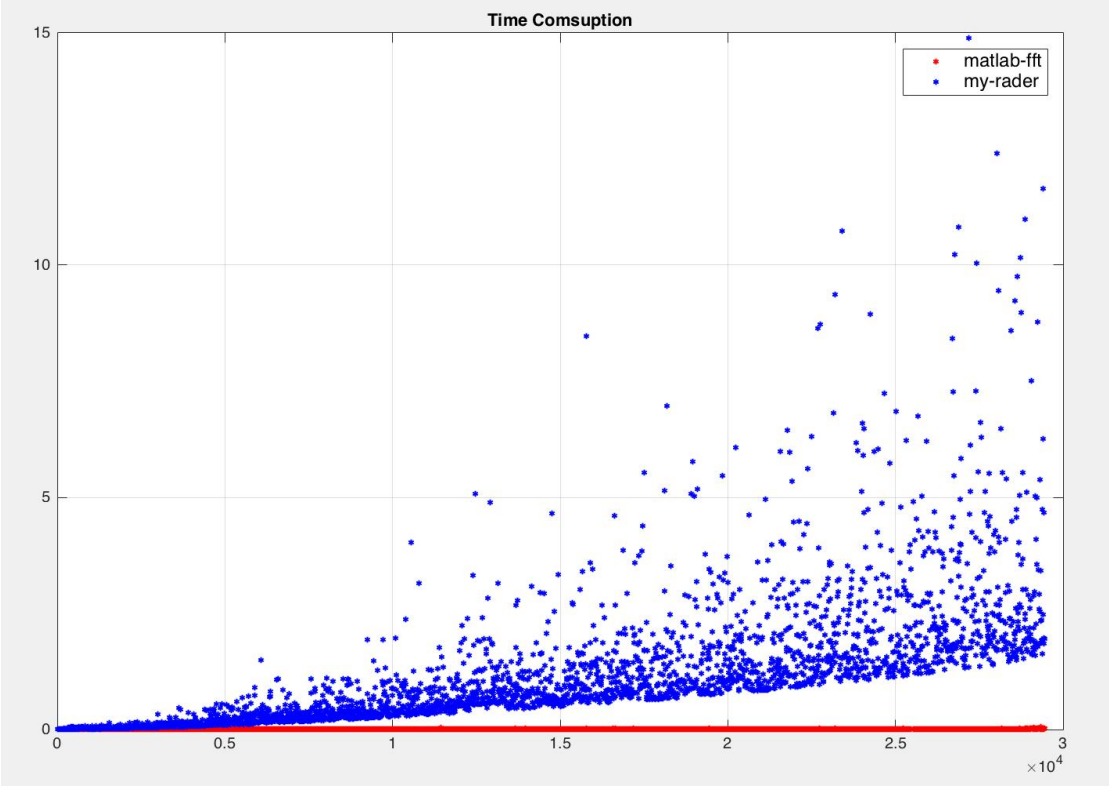

图 4.比较 Rader 算法与 matlab 内置 FFT，横坐标为素数 p，纵坐标为运行时间

4. 复杂度分析

假设给定长度为 p 的离散数据样本.

4.1 生成元

根据以上的算法描述，我们首先需要寻找循环群的生成元，这里根据数论的一些知识，我们通过查找文献[8]得出，如果 GRH 为真，那么寻找一个循环群的生成元的复杂度为 $O(\log^k p)$，k 为常数。

GRH: (Generalized Rieman Hypothesis) 广义黎曼猜想，狄利克雷 L 函数的所有非平凡零点的实部都为$\frac{1}{2}$。

4.2 重排序

接下来需要对输入的 p 个数据样本重新排序, 并求出对应的逆元的顺序。即将 h=[h_1 h_2 h_3 h_4]和初始的权重矩阵顺序进行重排。上文提到的求幂运算取模, 数论逆元等操作可以直接用快速幂算法($O(\log_2 N)$)实现, 一共有 p-1 轮迭代, 每轮的复杂度为 $O(\log_2 i)$, 因此累乘下去的复杂度为:

$$
\begin{aligned}
\log_2 1 + \log_2 2 + \log_2 3 + \ldots + \log_2 (p-2) &= \log_2((p-2)!) \\
&= \sum_{i=0}^{p-2} \log(i) \\
&\leqslant \sum_{i=0}^{p-2} \log_2(p-2) \\
&= (p-2)\log_2(p-2) \\
&= O((p-2)\log_2(p-2))
\end{aligned}
$$

4.3 卷积运算和 FFT

最后 FFT 的结果需要由重排序的样本数据与重排序后$\tilde{\omega}$的相乘再进行逆变换得到, 即为④式：$\tilde{\mathrm{H}} - h_0 = IDFT[DFT[\tilde{h}] \cdot DFT[\tilde{\omega}_p]]$.

其中$\tilde{\omega}$和$\tilde{h}$长度都为 p-1. 显然, 由于任何大于 2 的质数必定是奇数, 因此 p-1 必定是合数, 因此我们接下来可以对这两个长度为 p-1 的序列采用任一种基于合数的快速傅里叶变换(如混合基等).

事实上, 依据上面的讨论, 我们也可以在$\tilde{h}$,$\tilde{\omega}$后面补零一直到最近的 2 的幂数, 然后直接运用最简单的指基 2, 逐次加倍的 FFT, 而补零带来的影响是常数因子, 在复杂度讨论中可以不计.

因此, 两次 DFT 操作可以用 FFT 操作替代, 由 FFT 的复杂度为 $O(n\log(n))$, 这两次 DFT 带来的时间开销为: $O((p-1)\log_2(p-1))$.

变换后的点乘再进行逆傅里叶变换, 由于序列长度进行 element-wise product 长度不变, 因此这里的开销也与上面完全一致. 现在我们回头考虑$\tilde{\mathrm{H}}_0$,它只需要一次累加就可得到, 因此复杂度为 $O(\log(p))$.

4.4 总平均复杂度

依据上面的讨论, 我们就得到了 Rader 算法的总平均复杂度:

$O(\log^k p)+O((p-2)\log_2(p-2))+O((p-1)\log(p-1))+O(\log(p))=O(p\log(p))$

因此我们不难发现, 该算法使得对于素数的傅里叶变换也能像基于合数的

FFT 算法一样达到 *O(p log(p))*的数量级,尽管它可能要比它们慢若干常数倍.

但比起 brute-force DFT, 我们依旧获得 $c(p)=\frac{p^2}{p\log(p)}=\frac{p}{\log(p)}$ 的计算优势.

参考文献

[1] Cooley, James W.; Tukey, John W. (1965). "An algorithm for the machine calculation of complex Fourier series". Math. Comput. 19: 297–301.

[2] P. Duhamel and H. Hollmann, "Split-radix FFT algorithm," Electron. Lett. 20 (1), 14–16 (1984).

[3] Good, I. J. (1958). "The interaction algorithm and practical Fourier analysis". Journal of the Royal Statistical Society, Series B. 20 (2): 361–372.

[4] HEIDEMAN MT.Multiplicative complexity, convolution, and the DFT.New York:Springer---Verlag,1998

[5] C. M. Rader, "Discrete Fourier transforms when the number of data samples is prime," Proc. IEEE 56, 1107–1108 (1968).

[6] https://skybluetrades.net/blog/posts/2013/12/31/data-analysis-fft-9.html

[7] https://en.wikipedia.org/wiki/Rader's_FFT_algorithm

[8] Adamski, T., & Nowakowski, W. (2015). The average time complexity of probabilistic algorithms for finding generators in finite cyclic groups. Bulletin of the Polish Academy of Sciences Technical Sciences, 63(4), 989-996.