Digital Image Processing | Rader's algorithm

Fast Fourier Transform

1. Members

Sort by ID

16337157 刘上华

16337179 麦金杰

16337180 麦显忠

16337221 唐诗佳

16337223 田晓晴

2. Introduction

2.1 "逐次加倍"法的局限性

我们首先观察课本上提到的"逐次加倍"法. 它是基于应用广泛的Cooley-Tukey算法, 可以将DFT所需的N^2次复数相乘和$N(N-1)$次复数相加, 简化为$(N/2)log_2(N)$次复数乘法.

但是,它只适用于当N为2的幂时的情况.

但是, 在很多实际情况中, 我们往往很难保证去获得一个2的幂的样本, 或者说, 有时候只能以2的幂来进行采样带来了很多的局限性.

所以, 我们试着去找到一种更为通用的**FFT**方法**.** 或者说我们想对**FFT**进行更通用的扩展**.**

2.2 DFT When the Number of Data Sample Is Prime

因此, 基于以上的想法, 我们选择了Rader's algorithm进行复现. 它基于CHARLES M. RADER于1968年的论文*Discrete Fourier Transforms When the Number of Data Samples Is Prime*.

在当时的背景下, 已有的FFT算法都有一个比较大的局限性, 都要求样本数为2的幂, 或者是合数.

因此, 基于已提出的算法, Rader's algorithm尝试着解决当样本数为质数时, 如何进行快速傅里叶变换的问题.

3. Rader's algorithm

3.1 DFT的矩阵展开

首先, 假设我们有长度为p (**p为质数**)的样本, 那么根据经典的DFT, 它的傅里叶变换就可以写为:

$$H_n = \sum_{k=0}^{p-1} h_k e^{-2\pi \frac{kn}{p}},\ n = 0, 1, 2, \ldots, p-1$$

假如我们把H_0分开来写, 就得到:

$$H_0 = \sum_{k=0}^{p-1} h_p$$

$$H_n = h_0 + \sum_{k=1}^{p-1} h_k w_p^{nk}, n = 1, 2, \ldots, p-1$$

为了展示方便, 接下来我们以$p = 5$时的情况进行说明. 按照我们以上的公式和代换, 我们就可以得到后面一项组成的矩阵:

$$F_5 = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & w_5^1 & w_5^2 & w_5^3 & W_5^4 \\ 1 & w_5^2 & w_5^4 & w_5^6 & w_5^8 \\ 1 & w_5^3 & w_5^6 & w_5^9 & w_5^{12} \\ 1 & w_5^4 & w_5^8 & w_5^{12} & w_5^{16} \end{pmatrix}$$

借助上面的系数矩阵, 我们就可以通过矩阵乘法$F_5 * h_0, h_1, h_2, h_3, h_4$来得到样本的傅里叶变换$H_n$.

而我们还可以利用傅里叶变换的性质对上面的系数矩阵再进行化简.

注意到, 傅里叶变换也可以写为:

$$H_n = \sum_{k=0}^{p-1} h_k e^{-2\pi i \frac{kn}{p}} = \sum_{k=0}^{p-1} h_k \bullet [cos(2\pi \frac{kn}{p}) - i \bullet sin(2\pi \frac{kn}{p})],\ n = 0, 1, 2, \ldots, p-1$$

因此我们以w_5^{16}为例, 就有$w_5^{16} = e^{-2\pi i \frac{16}{5}} = cos(2\pi(3 + \frac{1}{5})) - i \bullet sin(2\pi(3 + \frac{1}{5})) = cos(2\pi \frac{1}{5}) - i \bullet sin(2\pi \frac{1}{5}) = e^{-2\pi i \frac{1}{5}} = w_5^1 = w_5^{16 mod 5}$

因此有:

$$F_5 = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & w_5^1 & w_5^2 & w_5^3 & W_5^4 \\ 1 & w_5^2 & w_5^4 & w_5^6 & w_5^8 \\ 1 & w_5^3 & w_5^6 & w_5^9 & w_5^{12} \\ 1 & w_5^4 & w_5^8 & w_5^{12} & w_5^{16} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & w_5^1 & w_5^2 & w_5^3 & W_5^4 \\ 1 & w_5^2 & w_5^4 & w_5^1 & w_5^3 \\ 1 & w_5^3 & w_5^1 & w_5^4 & w_5^2 \\ 1 & w_5^4 & w_5^3 & w_5^2 & w_5^1 \end{pmatrix}$$

3.2 整数模p乘法群

同样, 因为p是一个素数, 因此所有小于它的正整数$1, 2, \ldots, n-1$构成了一个模p乘法群. 通俗的理解即是, 群中任意两个元素相乘取模p后, 所得到的结果仍然会在群中.

根据数论的相关知识, 这是一个循环群, 一定能找到一个生成元:

> 循环群: 设 $(G, \cdot)$为一个群，若存在一G内的元素g，使得 $G = \langle g \rangle = \left\{ g^k;\ k \in \mathbb{Z} \right\}$，则称G关于运算“ $\cdot$ ”形成一个循环群。由群G内的任意一个元素所生成的群也都是循环群，而且是G的子群。

比如当$p = 5$时, 它对应的生成元$g = 2$, 因此它就生成了群:

$$2^0 = 1 \quad 2^1 = 2 \quad 2^2 = 4 \quad 2^3 = 3 \quad (mod\ 5)$$

这里我们再引入逆元的概念, 那么对于$p = 5, g = 2$, 解$gx \equiv 1\ mod\ 5$易得g的逆元为3.

> 乘法逆元: 指群G中的任意一个元素a, 都在G中有唯一的逆元a', 具有性质$a \bullet a' = a' \bullet a = e$. 这里有$e = 1$, 故$aa' \equiv 1\ mod\ p$.
>
> g^{-j}: 表示g的乘法逆元的j次幂.

而随着n递增性的, g^n, g^{-n}生成的序列是**周期性**的. 即:

$$2^0 = 1 \quad 2^1 = 2 \quad 2^2 = 4 \quad 2^3 = 3 \quad 2^4 = 1 \ \ldots \ (mod\ 5)$$
$$2^{-0} = 1 \quad 2^{-1} = 3 \quad 2^{-2} = 4 \quad 2^{-3} = 2 \quad 2^{-4} = 1 \ \ldots \ (mod\ 5)$$

3.3 将傅里叶变换表示为卷积

因此, 根据3.2中发现的规律, 我们可以把傅里叶变换的公式改写成:

$$H_g^{-r} = h_0 + \sum_{q=0}^{p-2} h_{g^q} w_p^{g^{q-r}} = h_0 + \sum_{q=o}^{p-2} h_g w_p^{g^{-(r-q)}}, r = 0, 1, \ldots, p-2$$

例如, 我们在$p = 5$时依据上述公式进行展开, 就有:

$$r = 0, g^{-r} = 1 \to H_1 = h_0 + \sum_{q=0}^{p-2} h_{g^q} w_5^{g^q} = h_0 + h_1 w_5 + g_2 w_5^2 + h_4 w_5^4 + h_3 w_5^3$$

$$r = 1, g^{-r} = 3 \to H_3 = h_0 + \sum_{q=0}^{p-2} h_{g^q} w_5^{g^{q-1}} = h_0 + h_1 w_5^3 + g_2 w_5^1 + h_4 w_5^2 + h_3 w_5^4$$

$$r = 2, g^{-r} = 4 \to H_4 = h_0 + \sum_{q=0}^{p-2} h_{g^q} w_5^{g^{q-2}} = h_0 + h_1 w_5^4 + g_2 w_5^3 + h_4 w_5^1 + h_3 w_5^2$$

$$r = 0, g^{-r} = 1 \to H_1 = h_0 + \sum_{q=0}^{p-2} h_{g^q} w_5^{g^{q-3}} = h_0 + h_1 w_5^2 + g_2 w_5^4 + h_4 w_5^3 + h_3 w_5^1$$

仔细观察上式, 我们发现通过使用生成元, 乘法逆元等进行下标代换, 其实就相当于实现了下面两个序列的卷积, 最终结果除了和DFT相比只是顺序上有些许差别:

q	**0**	**1**	**2**	**3**
a_q	h_1	h_2	h_4	h_3
b_q	w_5^1	w_5^3	w_5^4	w_5^2

3.4 卷积定理

根据卷积定理我们有:

$$(f \star g)(t) = \int_{-\infty}^{\infty} f(\tau) g(t-\tau) \, \mathrm{d}t$$

$$F(f \star g) = F(f) \bullet F(g)$$

因此我们先对这两个序列做傅里叶变换, 点乘后, 再进行一次逆傅里叶变换就可以得到我们的结果:

$$\widetilde{H} - h_0 = DFT^{-1}[DFT[\tilde{h}] \bullet DFT[\widetilde{w}_p]]$$

之后我们根据之前生成元的序列进行调整, 就能得到DFT的正确结果.

4. Analysis

4.1 Time Complexity

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[1]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$\frac{1}{2}$,

但是目前$O(log^k p)$的算法实现有一定的难度, 又需要找新的参考文献复现新的论文, 因此我们在实际的复现过程中采用了如下的循环迭代方法:

In [ ]:
```
%get generator of p
g=0;
for i=2:p-1
    unique_seq=zeros(1,p-1);
    found_g=1;
    trace=1;
    for j=0:p-2
        %temp=mod(power(i,j),p);
        temp=quick_mod(i,j,p);
        if ismember(temp,unique_seq)
            found_g=0;
            break;
        end
        unique_seq(trace)=temp;
        trace=trace+1;
    end
    if (found_g==1)
        g = i;
        break;
    end

end
```

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:

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

In [1]:
```
%%file quick_mod.m

%a^b mod k
function [ ans ] = quick_mod( a,b,k )

ans = 1;
a = mod(a,k);
while(b>0)
    if(mod(b,2)==1)
        ans = mod(ans*a,k);
    end
    b = b/2;
    b = fix(b);
    a = mod(a*a,k);
end
```

```
Created file 'F:\Acadamic_17_4_22\Digital_Image_Processing\project1\quick_mod.m'.
```

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的$\widetilde{w}$相乘再进行逆变换得到, 即由下式:

$$\widetilde{H} - h_0 = DFT^{-1}[DFT[\tilde{h}] \bullet DFT[\widetilde{w}_p]]$$

其中$\widetilde{w}$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$\tilde{h}, \widetilde{w}$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$O(nlog(n))$, 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$\widetilde{H_0}$, 它只需要一次累加就可得到, 因此复杂度为$O(p)$.

在实现的代码中我们直接调用了MATLAB内置的fft和ifft.

In [ ]:
```
hs_dft=fft(e_ro_hs);
ws_dft=fft(e_ws);
Hs=zeros(1,p);
Hs(1)=sum(hs);

temp_Hs=hs_dft.*ws_dft;
temp_Hs=ifft(temp_Hs);
temp_Hs=temp_Hs+h0;
```

4.4 总平均复杂度

依据上面的讨论, 我们就得到了Rader's algorithm的总平均复杂度:

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

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$O(plog(p))$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, 我们依旧获得了$c(p) = \frac{p^2}{plog(p)} = \frac{p}{log(p)}$的计算优势.

5. Result

5.1 正确性分析

我们首先将我们实现好的Rader's Algorithm和MATLAB的内置标准DFT做对比. 时间关系, 我们在本次Pre中只打表小于1000的素数进行测试.

In [26]:
```
primes=[2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53, ...
59,61,67,71,73,79,83,89,97,101,103,107,109,113,127,131, ...
137,139,149,151,157,163,167,173,179,181,191,193,197,199,211,223, ...
227,229,233,239,241,251,257,263,269,271,277,281,283,293,307,311, ...
313,317,331,337,347,349,353,359,367,373,379,383,389,397,401,409, ...
419,421,431,433,439,443,449,457,461,463,467,479,487,491,499,503, ...
509,521,523,541,547,557,563,569,571,577,587,593,599,601,607,613, ...
617,619,631,641,643,647,653,659,661,673,677,683,691,701,709,719, ...
727,733,739,743,751,757,761,769,773,787,797,809,811,821,823,827, ...
829,839,853,857,859,863,877,881,883,887,907,911,919,929,937,941, ...
947,953,967,971,977,983,991,997];
epsilon=0;
for p=primes
    hs = rand(1,p);
    x = (1:1:p);
    fft_ans = fft(hs);
    [g,order,inv_order,rad_ans] = my_rader(hs);
    epsilon=epsilon+norm(fft_ans-rad_ans);
end

disp(['l2 norm of dif  = ' num2str(epsilon)])
```

```
l2 norm of dif  = 5.0682e-11
```

我们再测试一个比较大的素数:

In [32]:
```
primes=[20441];

epsilon=0;
for p=primes
    hs = rand(1,p);
    x = (1:1:p);
    fft_ans = fft(hs);
    [g,order,inv_order,rad_ans] = my_rader(hs);
    epsilon=epsilon+norm(fft_ans-rad_ans);
end

disp(['l2 norm of dif  = ' num2str(epsilon)])
```

```
l2 norm of dif  = 7.6522e-11
```

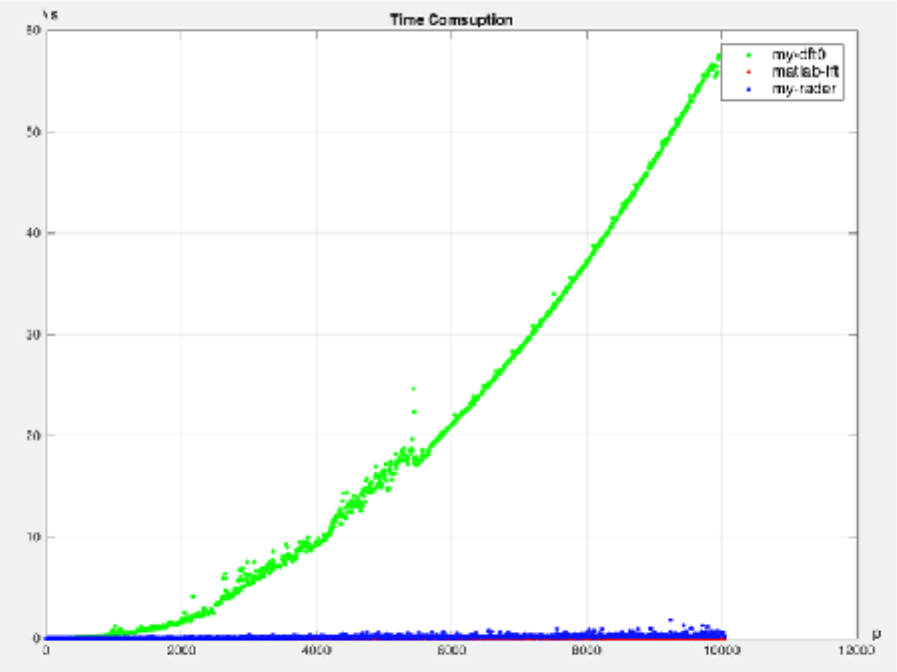
5.2 运行时间分析

我们首先将我们实现的算法与暴力DFT算法进行比对. 也就是下面的需要两层循环迭代展开的DFT算法:

In [36]:
```
%%file  dft0.m

function [ xk ] = dft0( xn )

 N=length(xn);
 WN=exp(-1i.*2.*pi./N);
 xk=zeros(1,N);
  n = 0:N-1;
 for k=0:N-1
   xk(k+1)=xn*WN.^(k.*n');  %此处下标一定得从1开始，因为matlab的下标是从1开始的

 end

end
```

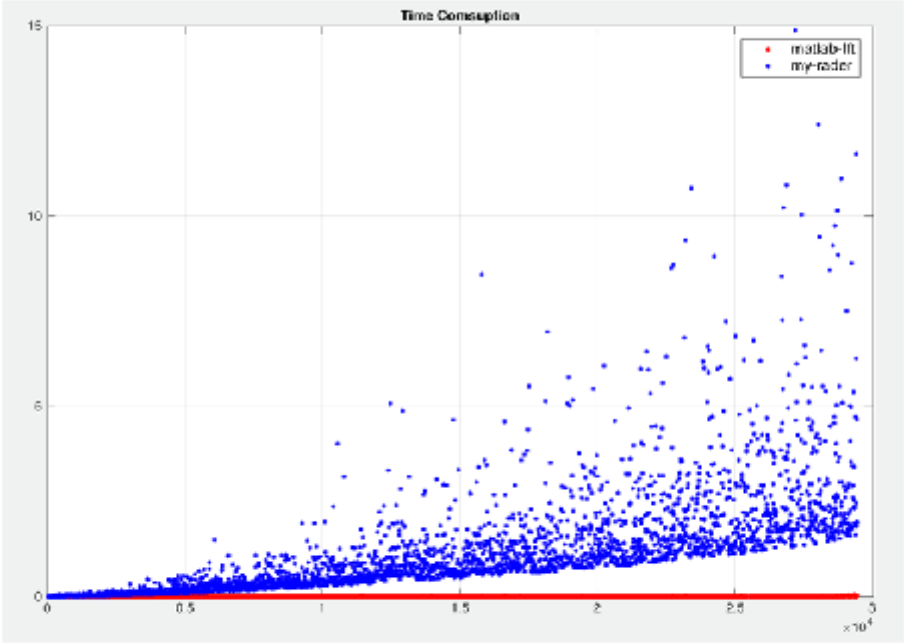
```
Created file 'F:\Acadamic_17_4_22\Digital_Image_Processing\project1\dft0.m'.
```

In [37]:
```
dft_rader = imread('pic1.png');
imshow(dft_rader)
```



上图中可以看到, Rader's算法确实比起暴力DFT具有不在同一数量级上的优势. 而我们把上图局部放大得到它与MATLAB内置的FFT对比, 理论上都是$O(nLog(n))$可以发现我们的算法还是比起它来差了不少, 说明MATLAB的自带函数还是优化的非常棒的.

In [38]:
```
fft_rader = imread('pic2.png');
imshow(fft_rader)
```



5.3 实际应用

在实际的应用中, 面对一个长为质数(或者因子比较少的合数), 如果我们单纯采用补零的方法来应用指基2 FFT, 这相当于对频域做了插值. 虽然理论上计算结果正确, 但是频域值的长度也被改变了. 在一些规定了数据样本傅里叶变换结果长度的情形下这种方法是不适用的.

> eg. 长度 5 的样本点如果补零到长为 8 进行指基2 快速傅里叶, 只能得到8个点的频谱, 而不是和原数据一一对应的5个频谱.

而像本文中的Raders这样的可用于质数的傅里叶变换, 保证了频谱长度不变的同时, 又没有损失可用于合数的FFT算法的$O(nlog(n))$复杂度, 因此在实践中还有一定的应用, 如傅里叶变换开源库FFTW在它的通用FFT实现中就采用了这一算法来计算比较大的质因子[2]. 经实际验证表明比其他算法能快2个常数倍.

而在它的实际实现中, 注意到上式中的第二项

$$\widetilde{H} - h_0 = DFT^{-1}[DFT[\tilde{h}] \bullet DFT[\widetilde{w}_p]]$$

完全与实际数据无关, 完全可以提前打表预计算, 因此原作者指出这个算法通过这个trick还可以至少被加速[3]$\frac{1}{3}$.

而总结我们的实现, 因为难度问题, 我们在求生成元等子步骤中并没有实现目前最新的, 基于数论研究的最快算法, 也没有做像原作者那样的优化, 但我们成功的实现了该算法的基本框架, 比起暴力DFT依旧取得了相当不错的性能优化.

因此我们的本次复现实验可以说取得了成功.

谢谢大家.

Reference

[1] 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.

[2] Frigo, M., & Johnson, S. G. (2005). *The design and implementation of FFTW3*. Proceedings of the IEEE, 93(2), 216-231

[3] Rader, C. M. (1968). *Discrete Fourier transforms when the number of data samples is prime*. Proceedings of the IEEE, 56(6), 1107-1108.