

2021秋数字信号处理第二次实验报告——快速傅里叶变换FFT算法实现

PB19071509 王瑞哲

一、实验目的

1. 加深对快速傅里叶变换的理解;
2. 掌握 FFT 算法及其程序的编写;
3. 掌握算法性能评测的方法。

二、实验原理

• 离散傅里叶变换(DFT)

在各种信号序列中,有限长序列在数字信号处理中占有很重要的地位。无限长的序列也往往可以用有限长序列来逼近。对于有限长的序列我们可以使用**离散傅立叶变换**(DFT),这一变换可以很好地反应序列的频域特性,并且容易利用快速算法在计算机上实现。当序列的长度是 N 时,我们定义离散傅立叶变换为

$$X(k) = DFT[x(n)] = \sum_{n=0}^{N-1} x(n)W_N^{kn}$$

其中旋转因子 $W_N = e^{-j\frac{2\pi}{N}}$ 。若令 $z = W_N^{-k}$, 则有:

$$X(z)|_{z=W_N^{-k}} = \sum_{n=0}^{N-1} x(n)W_N^{kn} = DFT[x(n)]$$

由此可知, $X(k)$ 是 z 变换在单位圆上的等距采样,或者说是序列傅立叶变换的等距采样。时域采样在满足 Nyquist 定理时,就不会发生频谱混淆;同样地,在频率域进行采样的时候,只要采样间隔足够小,也不会发生时域序列的混淆。

• 快速傅里叶变换(FFT)

快速傅立叶变换 FFT 并不是与 DFT 不相同的另一种变换,而是**为了减少 DFT 运算次数的一种快速算法**。它是对DFT变换式进行一二次的分解,使其成为若干小点数 DFT 的组合,从而减小运算量。常用的 FFT 是以 2 为基数,其长度 $N = 2^M$ 。它的**运算效率高,程序比较简单,使用也十分地方方便**。

对于 N 点有限长序列,如果直接采用DFT计算其频谱,则需要计 N^2 次复数乘法, $N(N-1)$ 次复数加法。而采用基-2FFT算法时,若 N 满足 $N = 2^m$ 的形式,运算次数则会下降为 $\frac{N}{2} \log_2^N$ 次复数乘法和 $N \log_2^N$ 次复数加法,大大加速了计算机处理数据的速度。

当需要进行变换的序列的长度不是 2 的整数次方的时候,为了使用以 2 为基的 FFT,可以用末尾补零的方法,使其长度延长至 2 的整数次方。IFFT一般可以通过 FFT 程序来完成,只要对 $X(k)$ 取共轭,进行 FFT 运算,然后再取共轭,并乘以因子 $1/N$,就可以完成 IFFT。

• 按时间抽取基-2快速傅里叶算法原理简述

在DFT中,运算复杂度主要来源于旋转因子矩阵中涉及到的大量复数乘法。但由于 $W_N = e^{-j\frac{2\pi}{N}}$ 旋转因子式本身的特殊性,很多乘法可以进行合并化简。例如,若 $(k_1 - k_2) = N/2$, 则 $W_N^{k_1 n} x(n) = W_N^{k_2 n} x(n)$ 。因此,如果我们关注 $X(k)$ 和 $X(k + N/2)$, 则有:

$$X(k) = \sum_{n=0}^{N-1} x(n)W_N^{nk}$$

$$X(k + \frac{N}{2}) = \sum_{n=0}^{N-1} x(n)W_N^{n(k + \frac{N}{2})}$$

而在 $n = 2r$ 的部分 $X(k)$ 和 $X(k + N/2)$ 重复计算，所以基于时域 n 的取值分离 $n = 2r, n = 2r + 1$:

$$X(k) = \sum_{n=0}^{\frac{N}{2}-1} x(2r)W_N^{2rk} + \sum_{n=0}^{\frac{N}{2}-1} x(2r+1)W_N^{(2r+1)k}$$

$$X(k + \frac{N}{2}) = \sum_{n=0}^{\frac{N}{2}-1} x(2r)W_N^{2r(k + \frac{N}{2})} + \sum_{n=0}^{\frac{N}{2}-1} x(2r+1)W_N^{(2r+1)(k + \frac{N}{2})}$$

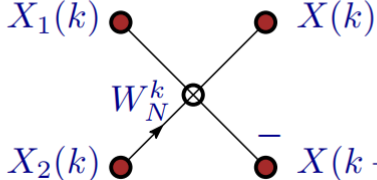
由旋转因子的对称性，我们可知：

$$W_N^{2rk} = W_N^{2r(k + \frac{N}{2})} = W_N^{rk} \quad W_N^{(2r+1)k} = W_N^k W_N^{rk} \quad W_N^{(2r+1)(k + \frac{N}{2})} = -W_N^k W_N^{rk}$$

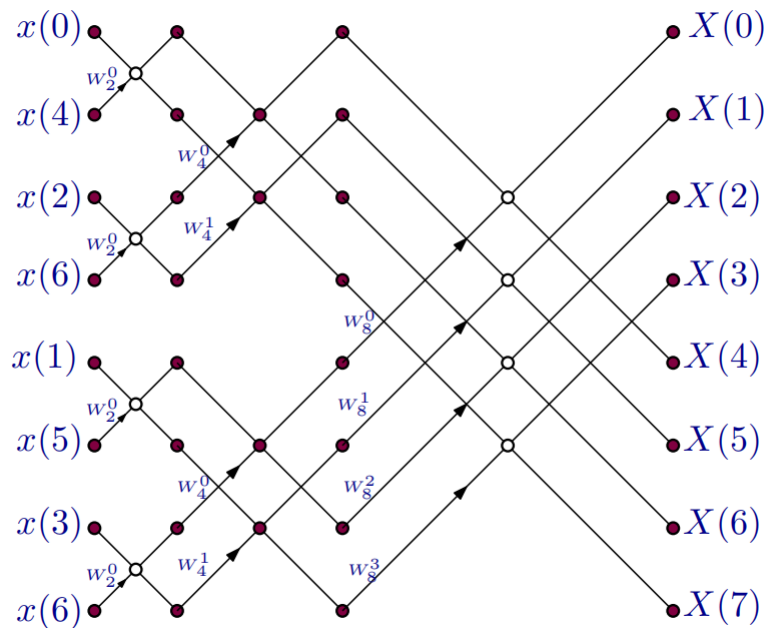
因此令 $X_1(k) = \sum_{r=0}^{N/2-1} X(2r)W_{N/2}^{rk}$, $X_2(k) = \sum_{r=0}^{N/2-1} X(2r+1)W_{N/2}^{rk}$ ，则可以得到下列的蝶形运算图：

$$X(k) = X_1(k) + X_2(k)W_N^k \quad X(k + \frac{N}{2}) = X_1(k) - W_N^k X_2(k)$$

$$X_1(k) = \sum_{r=0}^{N/2-1} X(2r)W_{N/2}^{rk}$$

$$X_2(k) = \sum_{r=0}^{N/2-1} X(2r+1)W_{N/2}^{rk}$$


因此，可以将 $x(n)$ 分为偶下标序列 $x(2r)$ 和奇下标序列 $x(2r + 1)$ ，分别得到对应的DFT结果 $X_1(k)$ 和 $X_2(k)$ ，再由 $X_1(k)$ 和 $X_2(k)$ 合成 $X(k)$ 和 $X(k + N/2)$ ，其中 $0 \leq k \leq N/2 - 1$ 。如果 $N = 2^m$ ，则还可以继续进行这样的分解和合并操作。以8点有限序列为例：



对于 N 点有限长序列，如果直接采用DFT计算其频谱，则需要计 N^2 次复数乘法， $N(N-1)$ 次复数加法。而采用基-2FFT算法时，若 N 满足 $N = 2^m$ 的形式，运算次数则会下降为 $\frac{N}{2} \log_2^N$ 次复数乘法和 $N \log_2^N$ 次复数加法，大大加速了计算机处理数据的速度。

三、实验内容

1. 编制自己的 FFT 算法

备注：实验报告要求中的第1条：

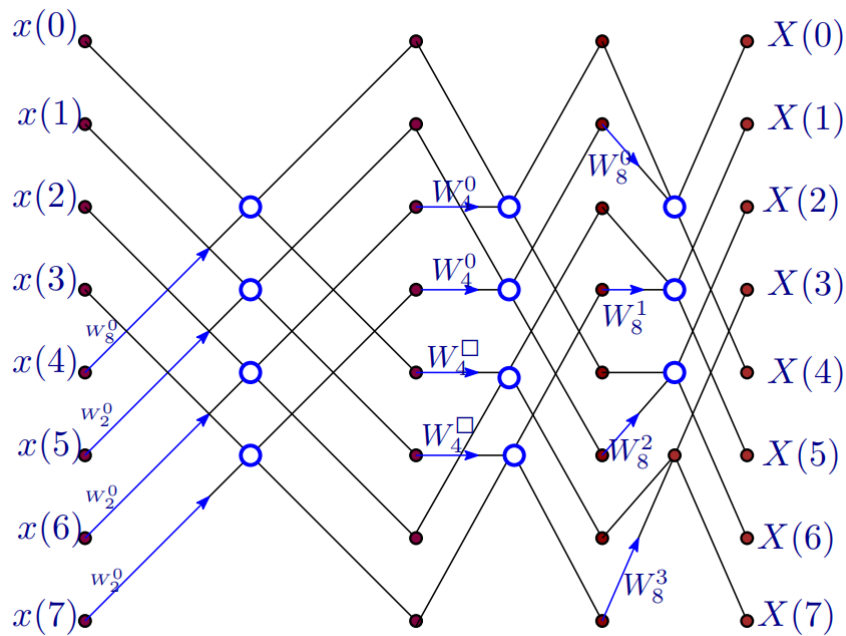
1、 总结自己实现 FFT 算法时候采用了哪些方法减小了运算量
在此部分给出

本次实验借助matlab软件，利用按时间抽取[DIT]的基-2快速傅里叶算法，自行编写了FFT算法程序并进行了运行验证。给出matlab实现代码如下：

```
function X=myfft(x)
% Myfft fuction by 王瑞哲 PB19071509
% 基本思路：基2-FFT算法，采用matlab矩阵并行运算以减少for循环所带来的运算量
M=nextpow2(length(x)); % 找比length(x)长的最小的2的幂，M为幂指数，也即是要做的
2点FFT次数
N=2^M;
if length(x)<=N
    x=[x,zeros(1,N-length(x))];
end %补全至2的幂次方

% 查阅资料得知，matlab中对循环运算耗费的时间很多，因此需要尽量避免循环运算
% 采用输入输出均为自然时序的基2-FFT算法
for i=1:M % 循环做2点FFT，一共做M=log2(N)次
    % 求n=当前列数
    if(i==1)
        n=N/2;
    else
        n=n/2;
    end
    % 利用matlab语法x(m:n,j:k)将x的第m到n行，j到k列挑出来
    x1=x(:,1:n); % 前一半序列
    x2=x(:,n+1:n*2); % 后一半序列
    w=exp(-1i*2*pi/(2^i)); % 计算当前旋转因子——对应点数为2^i
    col=(0:2^(i-1)-1)'; % 当前行数x1的一个纵向递增序列，作为w的指数
    w=w.^col; % 构造旋转因子列向量
    for j=1:n
        x2(:,j)=x2(:,j).*w; % 后一半序列每列按从0到n的指数乘旋转因子
    end
    x=[x1+x2;x1-x2]; % 将x的前一半和后一半纵向叠合并行计算
    % 循环结束后，原x列数减半M次最终至1，行数加倍M次最终至N=2^M
end
X=x'; % 按照上面一层一层并行的算法，结果应为N×1矩阵，转置即得最终的X(k)序列
end
```

算法分析：一般时域抽取FFT实现中，输入并不是按照自然顺序，而是按照二进制序列反转对应顺序。在此基础上，FFT关于其运算顺序还有很多变体，如输入自然时序输出反序，输入输出均为自然时序等等。本次实验考虑到matlab无论是对输入还是输出序列做二进制序列反序，都需要遍历序列，这将会消耗大量时间，因此本程序采用的是输入输出均为自然时序的FFT结构：



除此之外，由于matlab矩阵运算快的特性，本次实验中编写FFT算法**尽量避免使用循环尤其是嵌套循环，而转为使用向量计算方法**。例如在上流程图中，第一次对序列前后拆分并计算完毕DFT后，并未采用for循环，分别对前半序列和后半序列进行下一次拆分和DFT计算，而是把这两半序列合成为一个矩阵再进行计算，因为它们所要乘的旋转因子是具有关联的，则完全可以**借助matlab矩阵运算**（本实验中用到点积运算.*）——这样每次进行完一次拆分，原矩阵行数加倍，列数减半，再点乘上与之位数相同的旋转因子矩阵（旋转因子矩阵的构造由当前行数、列数共同决定），最终经过M次循环，原先的时域序列（认为是1×M的矩阵）得到M×1的频域序列，再转置即可得到输出。

2. 选取实验 1 中的典型信号序列验证算法的有效性

选取实验一中的理想采样信号序列： $x_a(t) = Ae^{-\alpha t} \sin(\Omega_0 nT)$, $0 \leq n < 1000$ ，其中 A 为幅度因子， α 是衰减因子， Ω_0 是频率， T 为采样周期。令

$A = 444.128$, $\alpha = 50\sqrt{2}\pi$, $\Omega_0 = 50\sqrt{2}\pi$, $T = 0.001$ 。对序列**分别调用FFT库函数和自编写FFT**求得其频谱，利用图像和数值对比两者结果。程序编写如下：

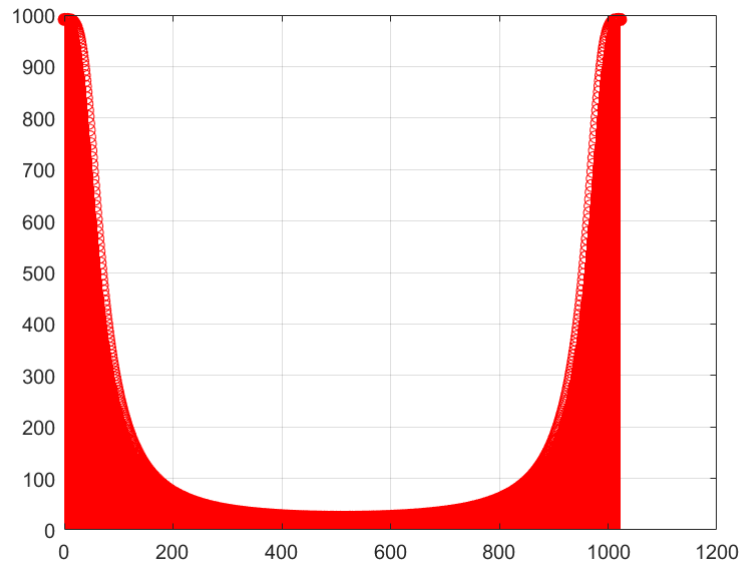
```
clear;
n=0:999;A=444.128;a=50*sqrt(2.0)*pi;T=0.001;w0=50*sqrt(2.0)*pi;
x=A*exp(-a*n*T).*sin(w0*n*T);    % 测试信号

x_fft=fft(x,1024);                % matlab自带的fft函数
x_myfft=myfft(x);                  % 自己编写的fft算法

figure(1)                           % 图片对比结果是否一致
stem((1:length(x_myfft))-1,abs(x_myfft),'b');hold on;grid on;
stem((1:length(x_fft))-1,abs(x_fft),'r');

sum=0;                               % 数值对比结果是否一致
for i=1:1000
    sum=sum+abs(abs(x_fft(i))-abs(x_myfft(i)));
end
```

运行结果为：



图像结果表示蓝色序列点（系统FFT）和红色序列点（自编FFT）完全重合，说明两者运行结果一致。而二者的绝对值误差之和计算结果为：

`sum=1.5029e-09`

% 在系统误差范围内，说明结果一致

选取更多的信号序列进行验证，如：

- 高斯序列 $x(n) = e^{-\frac{(n-p)^2}{q}}$, $0 \leq n < 1000$
- 衰减正弦序列 $x(n) = e^{-\alpha n} \sin 2\pi f n$, $0 \leq n < 1000$
- 矩形序列 $x(n) = R_N(n) = \begin{cases} 1, & 0 \leq n < N \\ 0, & \text{else} \end{cases}$, 其中 $N = 100$

编写代码如下：

```
clear;
n=0:999;
a=0.005;f=0.002;xa=exp(-a*n).*sin(2*pi*f*n);    % 衰减正弦序列
p=500;q=5000;xb=exp(-(n-p).^2/q);                % 高斯序列
N=100;xc=sign(sign(N-n)+1);                       % 矩形窗序列

subplot(3,3,1);stem(xa);title('衰减正弦序列');
subplot(3,3,2);stem(xb);title('高斯序列');
subplot(3,3,3);stem(xc);title('矩形窗序列');
fa=fft(xa,1024);subplot(3,3,4);stem((1:1024),abs(fa),'b');title('系统FFT幅度谱');
fb=fft(xb,1024);subplot(3,3,5);stem((1:1024),abs(fb),'b');title('系统FFT幅度谱');
fc=fft(xc,1024);subplot(3,3,6);stem((1:1024),abs(fc),'b');title('系统FFT幅度谱');
mfa=myfft(xa);subplot(3,3,7);stem((1:1024),abs(mfa),'r');title('自编FFT幅度谱');
mfb=myfft(xb);subplot(3,3,8);stem((1:1024),abs(mfb),'r');title('自编FFT幅度谱');
mfc=myfft(xc);subplot(3,3,9);stem((1:1024),abs(mfc),'r');title('自编FFT幅度谱');

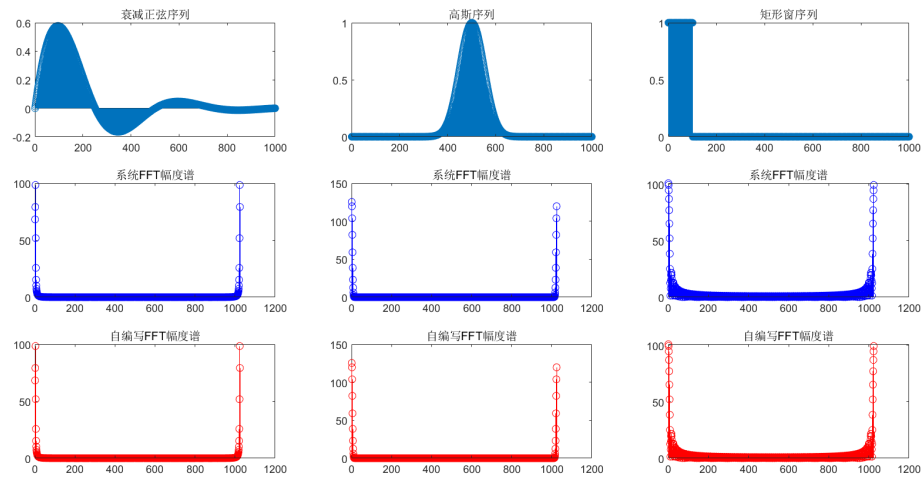
sum1=0;sum2=0;sum3=0;
for i=1:1000
    sum1=sum1+abs(abs(mfa(i))-abs(fa(i)));
end
```

```

sum2=sum2+abs(abs(mfb(i))-abs(fb(i)));
sum3=sum3+abs(abs(mfc(i))-abs(fc(i)));
end
disp(sum1);disp(sum2);disp(sum3);

```

作图验证得到结果：



幅值绝对值差sum输出结果为：

```

sum1=3.3914e-15
sum2=9.2879e-12
sum3=3.7443e-12

```

% 在系统误差范围内，说明结果一致

由此，经过多个典型的信号验证，可以说明所编写的FFT算法的正确性。

3. 对所编制 FFT 算法进行性能评估

备注：实验报告要求中的2、3两条：

2、 给出自己的 FFT 算法与实验 1 中自己的 DFT 算法的性能比较结果

3、 给出自己的 FFT 算法与 Matlab 中 FFT 算法的性能比较结果

在此部分给出

为了评估所编写FFT算法的性能，利用matlab中的 tic 和 toc 计数器，采用循环200次求平均运行时间的方法，比较自己编写的 FFT、matlab自带 FFT 和自己编写的 DFT 算法对于相同时域序列的分析情况。实现代码如下：

```

clear;
n=0:999;
A=444.128;a=50*sqrt(2.0)*pi;T=0.001;w0=50*sqrt(2.0)*pi;
x=A*exp(-a*n*T).*sin(w0*n*T); % 测试信号

loop=200; % 循环200次求平均运行时间
t1=zeros(1,loop);
t2=zeros(1,loop);
t3=zeros(1,loop);

for j=1:loop
    tic;
    x_fft=fft(x,1024); % matlab自带的fft函数
    t1(j)=toc;

```

```

end

for j=1:loop
    tic;
    x_myfft=myfft(x);          % 自己编写的fft算法
    t2(j)=toc;
end

for j=1:loop
    tic;
    x_mydft=mydft(x);          % dft算法
    t3(j)=toc;
end

figure(1)                      % 对比三种方法算出的结果是否一致
stem((1:length(x_myfft))-1,abs(x_myfft),'b');hold on;grid on;
stem((1:length(x_fft))-1,abs(x_fft),'r');
stem((1:length(x_mydft))-1,abs(x_mydft),'g');hold off;

Et1=mean(t1);                  % 对比三种方法运行花费的时间
Et2=mean(t2);
Et3=mean(t3);
disp('自带fft: '),disp(Et1);
disp('我的fft: '),disp(Et2);
disp('我的dft: '),disp(Et3);

```

其中，dft算法为：

```

function x=mydft(x)
M=nextpow2(length(x));
N=2^M;
if length(x)<=N
    x=[x,zeros(1,N-length(x))];
end%

n=0:N-1;
k=0:N-1;
X=x*(exp(-2i*pi/N)).^(n'*k);

```

其中 i 为复数单位， $W = \exp(-2i * \pi / N)$ 为旋转因子。**理解该式的关键在于理解 \wedge 算符。**matlab中， $\wedge 2$ 是矩阵中的每个元素都求平方， $\wedge 2$ 是求矩阵的平方或两个相同的矩阵相乘。这里先运算 $W.\wedge(n'*k)$ ——即为旋转因子 W 的 $n' * k$ 乘方，其中 n 和 k 都是0到49的序列，或说为维数 1×50 的矩阵。 n' 表示对 n 做转置，则 $n' * k$ 相当于 50×1 的矩阵和 1×50 的矩阵相乘，结果为 50×50 的矩阵，该矩阵通过 \wedge 作用于旋转因子上，相当于这个矩阵的每个元素作为指数，底数全变成 W ，便得到了我们熟悉的 50×50 的旋转因子的矩阵。

则再计算原矩阵 x （维数 1×50 ）矩阵乘（*）该旋转因子矩阵（维数 50×50 ），便可得到维数 1×50 的DFT结果序列 $X(k)$ 。其中 $X(k)$ 的每一个元素对应 k 不变时， n 从0到49积分，恰好对于旋转因子矩阵的每一列，所以前面的计算中必须是 $n' * k$ ，让变量 k 位于列的位置，而不能变换顺序成 $k' * n$ 。

根据上述对DFT算法分析可知，DFT分析 N 点序列要做 N^2 次复数乘法，仅在这一方面，相对于FFT算法所需要的 $\frac{N}{2} \log_2^N$ 次复数乘法就相差甚远，尤其是在 N 较大的情况下。运行上述代码，得到的结果为：

自带fft: 6.4030e-06
我的fft: 3.3805e-04
我的dft: 0.2884

可粗略看出，自编写的fft算法比自编写的dft算法快了约1000倍。而matlab自带的fft算法则比自行编写的fft又快了约50倍。**进一步地对比**，通过对不同序列点数的原序列分别进行matlab自带fft、自编写fft和dft分析，对比它们的运算时间随N的变化，并观察变化情况。给出此部分的代码实现如下：

```
N=12; % 循环次数
A=444.128;a=50*sqrt(2.0)*pi;T=0.001;w0=50*sqrt(2.0)*pi; % 测试信号的常数参数
T1=zeros(1,N);T2=zeros(1,N);T3=zeros(1,N);
for i=1:N % 循环N次
    n=1:(2^i); % 每次序列长为2^i
    x=A*exp(-a*n*T).*sin(w0*n*T); % 测试信号

    loop=50; % 循环50次求平均运行时间
    t1=zeros(1,loop);t2=zeros(1,loop);t3=zeros(1,loop);

    for j=1:loop % matlab自带的fft函数
        tic; x_fft=fft(x); t1(j)=toc;
    end

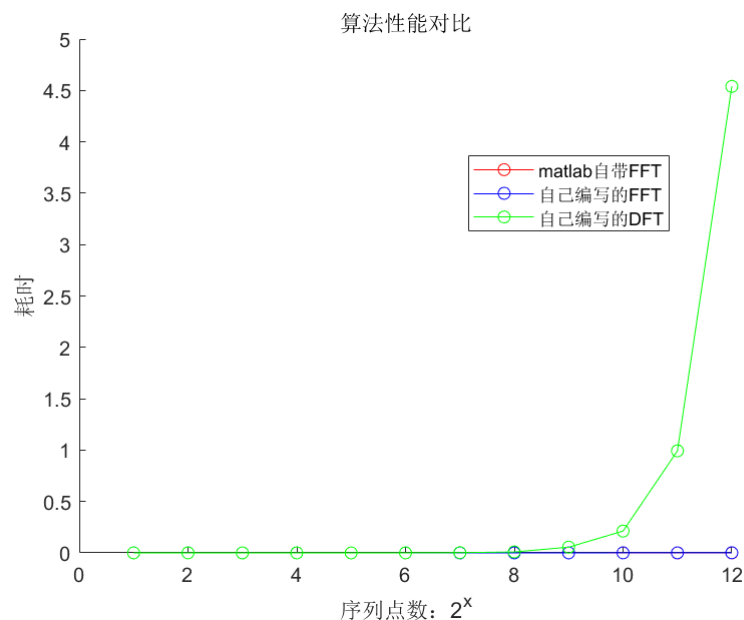
    for j=1:loop % 自己编写的fft算法
        tic; x_myfft=myfft(x); t2(j)=toc;
    end

    for j=1:loop % 自己编写的dft算法
        tic; x_dft=mydft(x); t3(j)=toc;
    end

    Et1=mean(t1); T1(i)=Et1;
    Et2=mean(t2); T2(i)=Et2;
    Et3=mean(t3); T3(i)=Et3;
end

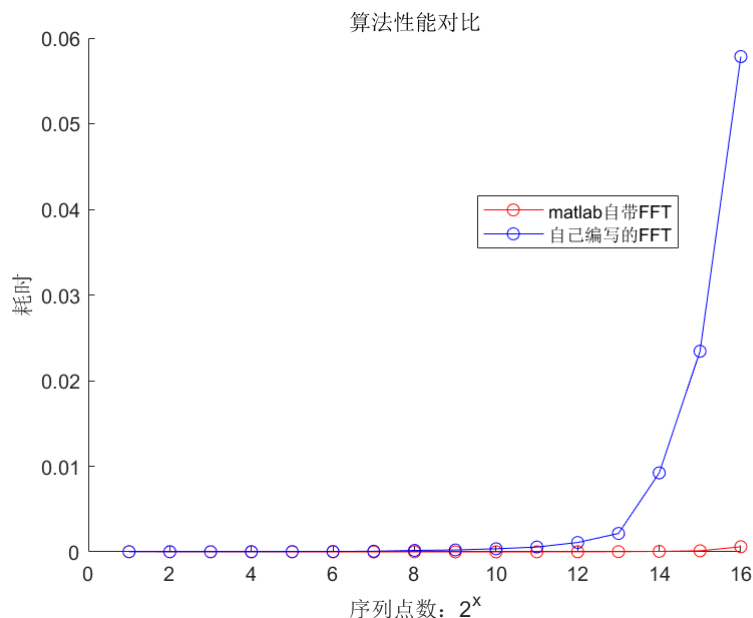
figure; hold on
title('算法性能对比'); ylabel('耗时'); xlabel('序列点数: 2^x');
plot(T1,'r-o'); plot(T2,'b-o'); plot(T3,'g-o');
legend('matlab自带FFT','自己编写的FFT','自己编写的DFT','Location','Best');
```

运行结果如下图所示：



可见自编写DFT随序列点数的上升，其所耗时间迅速增加，这是因为它所做的复数乘法是和 N^2 成正比的。与此同时，自编写FFT和matlab库FFT相对于DFT的耗时则大大降低，这与理论分析一致。

如果单独对比matlab库FFT和自编写FFT，则如下图：



由图像可见，两者在N增大的过程中，运行时间都有增长，且整体来说自编写FFT算法要比matlab自带FFT算法耗时更长；但至少在 $2^{12} = 4096$ 点序列以内，两者的差别还未相差很大；若 N 进一步增大，则自编写FFT的性能则会受到影响，但即使是在 $2^{16} = 65536$ 点如此庞大的序列下，仍然可在0.1秒内完成所有运算。相比之下matlab库fft函数则更为出色，在这种情况下仍可以控制在0.001秒内完成运算。

四、分析总结

- 1、总结自己实现 FFT 算法时候采用了哪些方法减小了运算量。
- 2、给出自己的 FFT 算法与实验 1 中自己的 DFT 算法的性能比较结果。
- 3、给出自己的 FFT 算法与 Matlab 中 FFT 算法的性能比较结果。

(1、2、3点已在第三部分实验内容中完成)

- 4、总结实验中根据实验现象得到的其他个人结论。

在本次实验中，除了理解快速傅里叶变换基本原理、完成基本FFT算法编写、比对各算法性能之外，我还得到了其他结论与收获：

- matlab函数调用方法

在本次实验中，三种不同的算法都是采用调用函数的方法来实现的。其中在编写自己的FFT和DFT算法中，采用的matlab函数编写语法为：

```
function [outputArg1,outputArg2] = untitled(inputArg1,inputArg2)
%UNTITLED 此处显示有关此函数的摘要
% 此处显示详细说明
outputArg1 = inputArg1;
outputArg2 = inputArg2;
end
```

本次实验中，所用到的函数myfft具有一个输入参数（原序列 x ）和一个输出参数（频域序列 X ）。上代码中的untitled填写为函数名（myfft）。

- matlab定时器tic、toc用法

在评测函数性能的时候，不可避免地会使用到计时功能。本次实验中，利用到了matlab自带计数器 `tic`、`toc`，具体用法如下：

```
tic;
% 待测试语句段
t=toc;
```

通过观察变量 t 既可以观察到函数运行所耗费的时间。

- 性能对比结果的展示

本次实验中用到了多种方法以展示不同算法之间的正确性和性能差异：

- 作出不同算法所计算的频谱，通过肉眼观察它们之间的区别。这种方法可以较为直观地看出各算法之间是否出错，但不能做出比较精细的判断；
- 计算绝对误差总和。若总误差在允许范围之内，则可以确认算法的正确性；
- 利用上面的tic、toc函数计算运行时间，对比算法性能。为了减小随机误差，本次实验还采取了多次运算求平均的方法，但代价是代码运行时间更长了；
- 利用matlab便捷的绘图功能，通过改变序列长度，绘图展示不同算法之间的性能差异，如本报告第三部分所述。