实验 2　FFT 算法实现

夏厚　PB18051031

2.1 实验目的

I、加深对快速傅里叶变换的理解。
II、掌握 FFT 算法及其程序的编写。
III、掌握算法性能评测的方法。
IV、熟悉 MatLab 编程。

2.2 实验原理

一个连续信号 Xa(t)的频谱可以用它的傅里叶变换表示为:

$$X_a(j\Omega)=\int_{-\infty}^{+\infty}x_a(t)e^{-j\Omega t}dt$$

如果对该信号进行理想采样，可以得到采样序列:

$$x(n)=x_a(nT)$$

同样可以对该序列进行 z 变换，其中 T 为采样周期:

$$X(z)=\sum_{-\infty}^{+\infty}x(n)z^{-n}$$

当 z=$e^{j\omega}$的时候，我们就得到了序列的傅里叶变换:

$$X(e^{j\omega})=\sum_{-\infty}^{+\infty}x(n)e^{j\omega n}$$

其中 w 称为数字频率，它和模拟域频域的关系为:

$$\omega=\Omega T=\Omega/f_s$$

其中 fs 是采样频率。上式说明数字频率是模拟频率对采样率 fs 的归一化。同模拟域的情况相似，数字频率代表了序列值变化的速率，而序列的傅立叶变换称为序列的频谱。序列的傅立叶变换和对应的采样信号频谱具有下式的对应关系。

$$X(e^{j\omega})=\frac{1}{T}\sum_{-\infty}^{+\infty}X_a\left(j\frac{\omega-2\pi n}{T}\right)$$

即序列的频谱是采样信号频谱的周期延拓。从上式可以看出，只要分析采样序列的频谱，就可以得到相应的连续信号的频谱。

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(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:

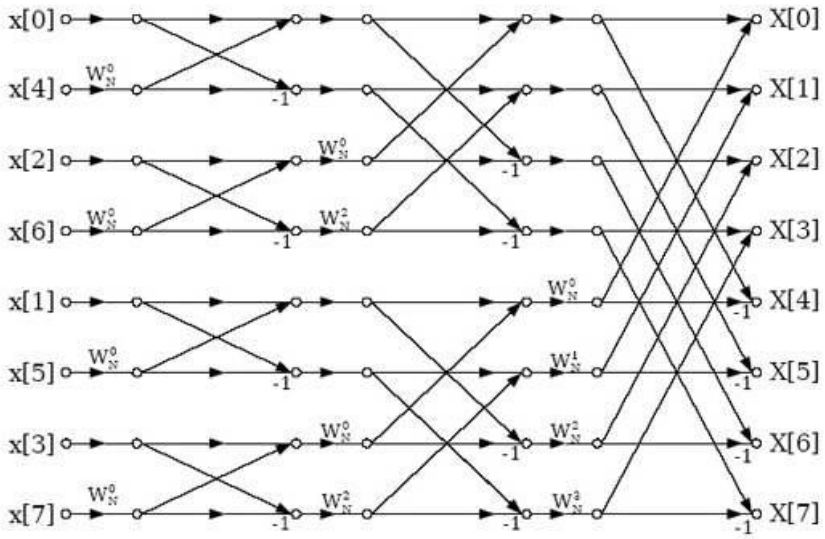
$$X(k) = DFT[x(n)] = \sum_{n=0}^{N-1} x(n) W_N^{kn}$$

其中W_N=$e^{-2\Pi j/N}$，它的反变换定义为：

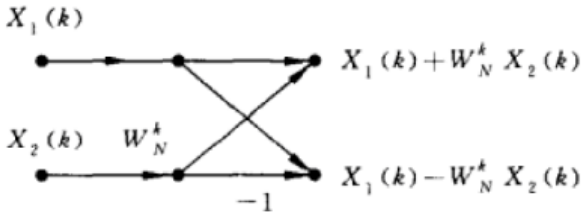
$$x(n) = IDFT[X(k)] = \frac{1}{N} \sum_{k=0}^{N-1} X(k) W_N^{-kn}$$

若直接计算 DFT 变换，整个 DFT 运算需要$4N^2$次实数相乘和 2N(2N-1)次实数相加。所以直接计算乘法次数与加法次数都和N^2成正比。例如 N=10 点的 DFT，需要 100 次复数相乘，而 N=1024 时则需要 1,048,576 即一百多万次复数乘法运算。这对于实时性要求很强的信号处理来说，必将对计算速度有十分严苛的要求。为此，FFT 作为对 DFT 的改进诞生。

快速傅里叶变换 FFT 并不是与 DFT 不相同的另一种变换，而是在 DFT 计算规律上建立的一种减少运算次数的快速算法。常用 FFT 是以基-2 的，长度 N=2^M。运算效率高，程序简单，使用方便。本实验就使用以 2 为基实现 FFT。算法流程图可以用蝶形算法来表示，以 8 点的基 2-FFT 算法为例：



每个蝶形运算为：



可以看到，每个蝶形运算都可原位运算。

当需要进行变换的序列长度不是 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。

2.3 实验内容

1、算法实现

以对高斯序列进行 FFT 代码为例：

```
function [ ]=dofft(M);%M为fft序列长度
m1=log2(M);%基2-fft层数
n=1:M;
x=zeros(1,M);
%%%%%%%%%
%高斯序列
%%%%%%%%%
i=0:63;
p=8;q=8;x=exp(-1*(i-p).^2/q);
%x是用来进行fft计算的序列
x1=zeros(1,M);
x1=x(bitrevorder(n-1)+1);%x下标二进制逆序调整，使x1顺序为满足fft输入的顺序
n=1:m1;
w=exp(-j*2*pi./2.^n);%第n层的旋转因子
for n=1:m1;%n表示在做第n层蝶形运算，一共有m1层
    for k=1:M/(2^n);%表示第n层分为k部分
        for m=1:2^(n-1);%m表示k部分的第m个蝶形运算
            p=x1((k-1)*2^n+m);
            q=x1((k-1)*2^n+m+2^(n-1));
            x1((k-1)*2^n+m)=p+q*(w(n)^(m-1));%蝶形运算
            x1((k-1)*2^n+m+2^(n-1))=p-q*(w(n)^(m-1));%蝶形运算
        end
    end
end
y=fft(x);
subplot(3,1,1);stem(abs(y));title('高斯序列MatLab的FFT结果：');
subplot(3,1,2);stem(abs(x1));title('编写的FFT结果：');
subplot(3,1,3);stem(abs(y-x1));title('两者相差：');
end
```

①、使用 bitrevorder()函数对待变换信号顺序进行调整，存入 x1 中。例如 M=8 时
原顺序：000(0)，001(1)，010(2)，011(3)，100(4)，101(5)，110(6)，111(7)
对二进制翻转之后为：
新顺序：000(0)，100(4)，010(2)，110(6)，001(1)，101(5)，011(3)，111(7)
对比实验原理中 M=8 的 FFT 输入序列发现两者顺序一致。

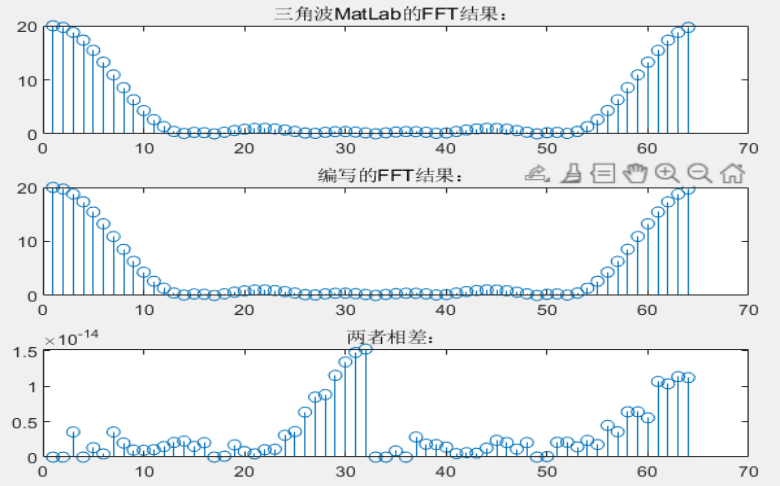
②、n=1:m1 表示一共有 m1 层运算，比如 M=8 时有 3 层。
k=1:M/(2^n)表示第 n 层分为 k 部分，比如 M=8 的第一层分为 4 个单独蝶形运算，第二层分为 2 个两两蝶形运算，第三部分为 1 个四四蝶形运算。
m 表示第 k 部分的第 m 个蝶形运算。

③、因为蝶形运算是原位运算，就不需要另外开辟空间，计算结果仍然存在原位置。

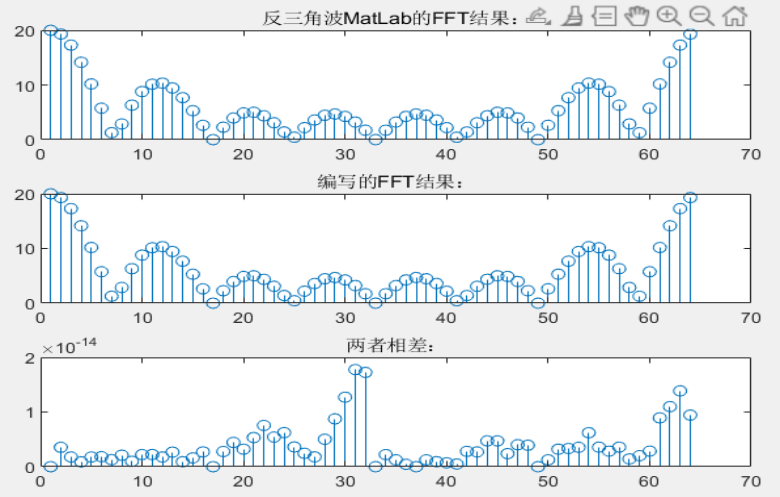
④、绘出编写的 FFT 计算结果与 MATLAB 的 FFT 计算结果，以及两者的差。

2、选取实验 1 中的典型信号序列验证算法的有效性

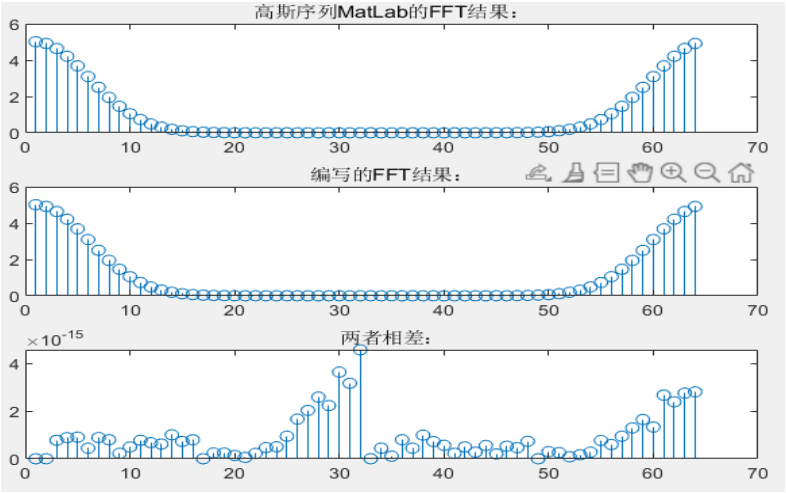
Ⅰ、三角波



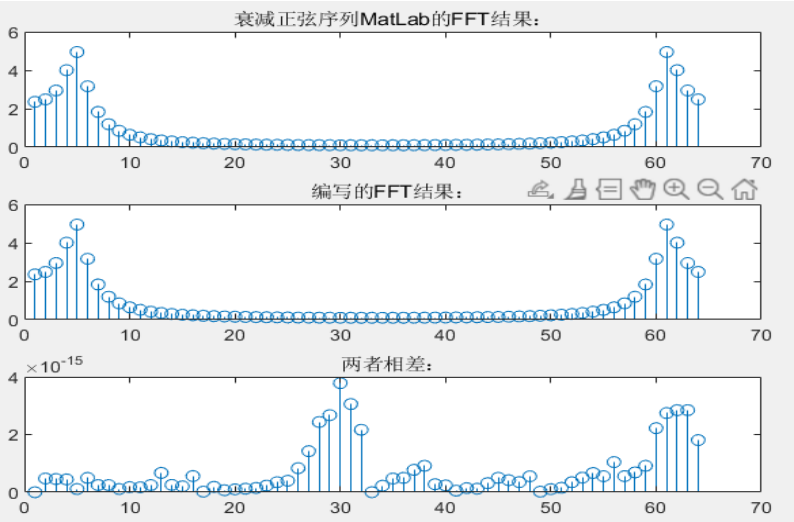
Ⅱ、反三角波



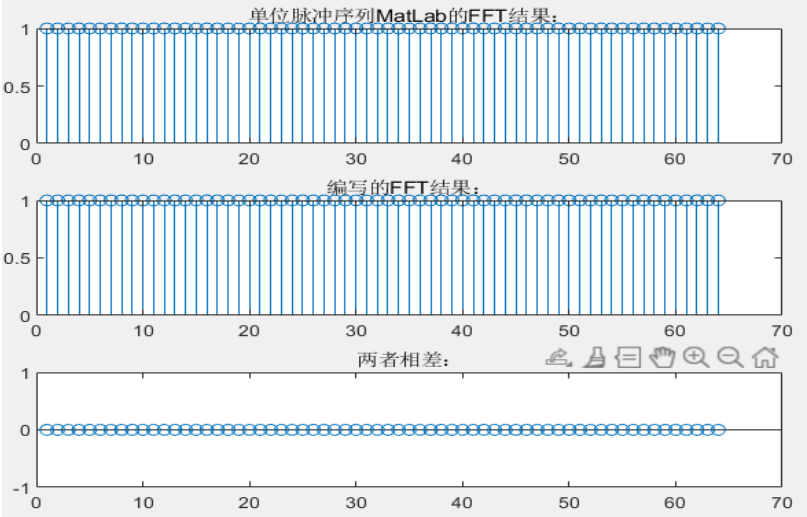
Ⅲ、高斯序列



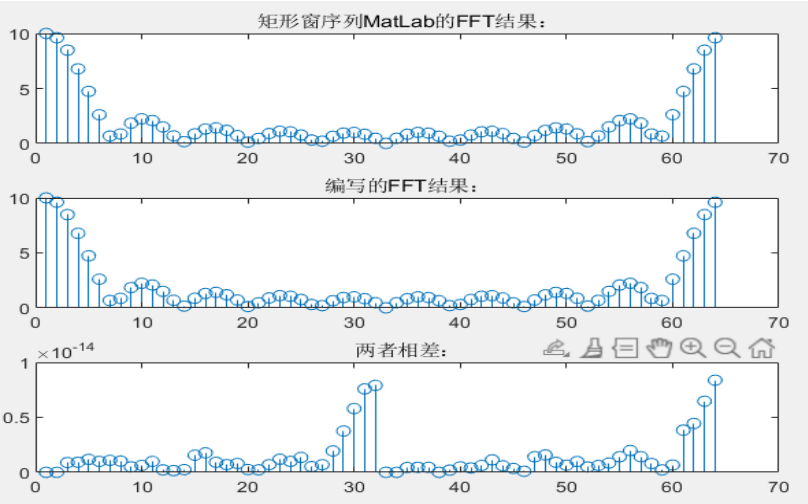
Ⅳ、衰减正弦序列



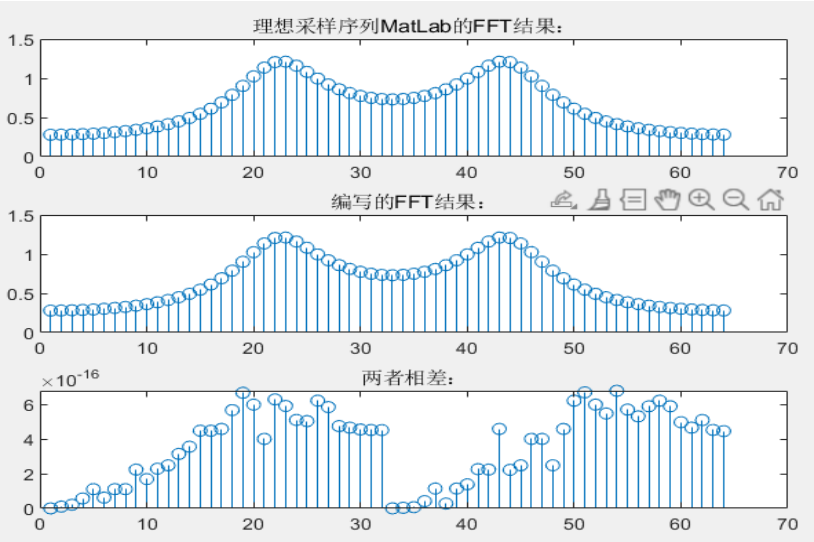
Ⅴ、单位脉冲序列



Ⅵ、矩形窗序列



VII、理想采样序列



注意每个序列自己编写的 FFT 计算结果和 MATLAB 的 FFT 计算结果，两者相差数量级在10^{-16}到10^{-14}。说明编写的 FFT 算法正确性没有问题。

3、对所编制的 FFT 算法进行性能评估

算法的评估首先是其正确性，是否能够完成预期功能决定了该算法是否有意义，上一部分已经通过典型信号序列验证了编写的 FFT 算法的有效性。

这里接下来主要从时间复杂度和空间复杂度两方面来进行评价。空间复杂度是指该算法运行过程需要占用多少内存空间，随着半导体产业的发展，内存空间变得越来越廉价，空间复杂度对算法性能的影响也越来越小。人们往往根据时间复杂度来评价一个算法的性能。而时间复杂度主要依赖于算法的计算次数。已知基 2-FFT 算法的复数乘法次数为$\frac{1}{2}Nlog_2N$。相比于加法，乘法在运算中要复杂得多，占用资源也更多，所以时间复杂度主要依赖乘法次数。从理论复杂度来看，FFT 显然比 DFT 的N^2更优。在 MARLAB 中查看程序运算时间有以下办法：

①、tic 和 toc 命令组合

```
tic;
operation;
toc;
```

tic 用来保存当前时间，也就是 operation 开始运行时间，toc 用来记录程序完成时间。MATLAB 会自动计算时间差并显示（以秒为单位但能精确到小数点后 6 位，即 us）。

②、etime(t1,t2)和 clock 配合

```
t1=clock;
operation;
t2=clock;
etime(t1,t2);
```

通过调用 windows 系统时钟进行时间差计算得到运行时间，t1 和 t2 之间的时间差。

③cputime 函数

```
t1=cputime;
operation;
t2=cputime-t1;
```

使用 cpu 主频计算运行时间差，得到程序运行时间。

tic/toc 是 MATLAB 自身计数器，精度要高于后两者。而且，如②调用系统时钟计算时间差，这段时间中系统可能还有其他后台程序。

MATLAB 官方推荐使用 tic/toc 组合，When timing the duration of an event,use the tic and toc functions instead of clock or etime.所以接下来的评估程序运行时间，本实验使用 tic/toc 命令。此外，程序运行时间和计算机本身的计算能力有着直接关系，以下数据都是在个人笔记本电脑测得。由于电脑属于商务本，计算能力很有限，时间相对会稍长一些。

以上主要说明了 FFT 算法评估方法和侧重点，具体评估数据在下面的 dofft 与 DFT、dofft 与 MATLAB-FFT 的性能比较中给出。

2.4 实验报告要求

1、总结自己实现的 FFT 算法时采用了哪些方法减少了运算量。

1）使用 matlab 的 bitrevorder()函数实现二进制翻转，由于 matlab 的函数是基于更底层的的 c 语言编写的，有很专业的优化，执行速度肯定更快。
2）尽量使用小循环套大循环，因为执行的跳转原因，大循环单次执行时间优于小循环。
3）利用蝶形运算的原位性，使用同一个地址空间存储变换前序列和变换后序列。
4）每相邻计算的蝶形运算数据在地址上尽量连续，减少寻址时间。

2、给出自己的 FFT 算法与实验 1 中的 DFT 算法性能比较结果。

为避免运算时间过短不利于记录，使用 20 次循环。dofft 程序见 2.3，其余程序如下

DFT 程序:

```
N=512;
n=1:N;
p=8;q=8;x=exp(-1*(n-p).^2/q);
k=1:N;
X=x*(exp(-j*pi/256)).^(n'*k);
```

dofft 测试程序:

```
clear all;
tic;
for f=1:20;
    clear all;
    dofft_3(512);
    clear all;
end
toc;
```

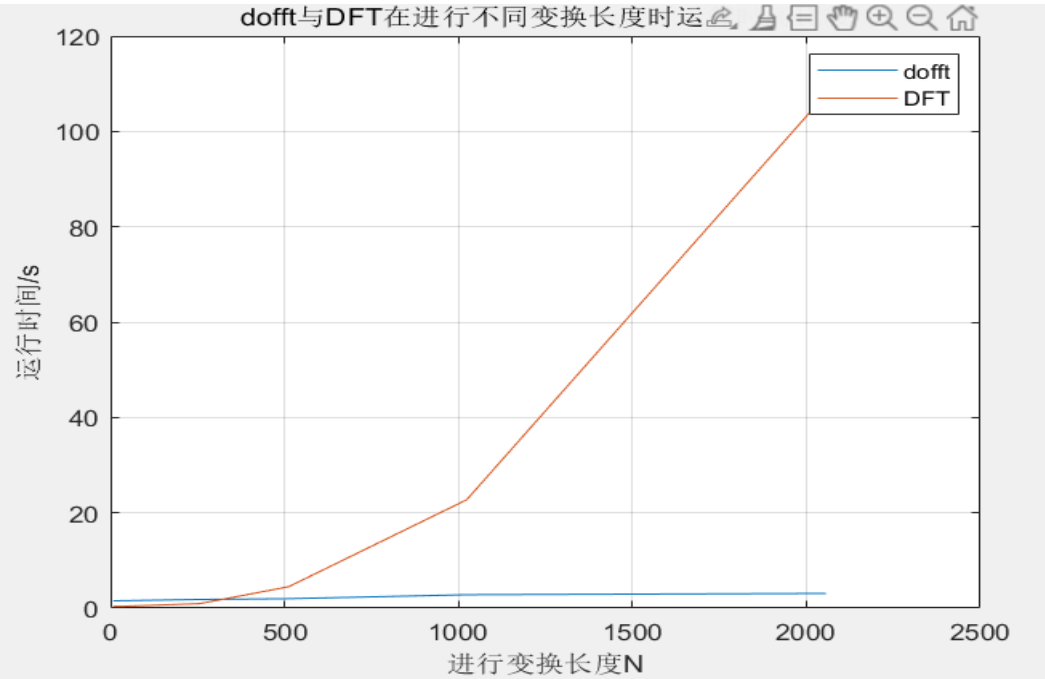
DFT 测试程序:

```
clear all;
tic;
for f=1:20;
    clear all;
    dft;
    clear all;
end
toc;
```

运行时间记录如下：

算法 \ N	8	32	256	512	1024	2048
dofft	1.524102	1.563602	1.820989	1.984634	2.814433	3.088558
DFT	0.367555	0.375953	0.945036	4.491770	22.746014	107.771205

作图：



测试序列为理想采样序列。为避免循环运行时，MATLAB 程序在前一循环已在内存中开辟空间和留有数据，测试程序中使用了 clear all 命令来清除内存中的数据。这样测得的时间更加准确。从记录的运行时间中可以看到，当 N 比较小的时候，DFT 运行时间比 dofft 时间更小，这是因为 DFT 算法使用的是向量运算，而 dofft 中使用了循环。MATLAB 本身对于向量计算的速度快于循环的计算速度。所以如果进一步优化 dofft 算法，可以改用向量运算，避免循环。当 N 趋于更大时,DFT 运行时间迅速上升，很快和 dofft 运行时间不在一个数量级。这和 DFT、FFT 两算法的理论时间复杂度一致。

3、给出自己的 FFT 算法和 MATLAB 中 fft 算法性能比较结果。

采用与 2 中相同的测试方法，同样使用 20 次循环。

fft 程序：

```
N=262144;
n=1:N;
p=8;q=8;x=exp(-1*(n-p).^2/q);
fft(x);
```

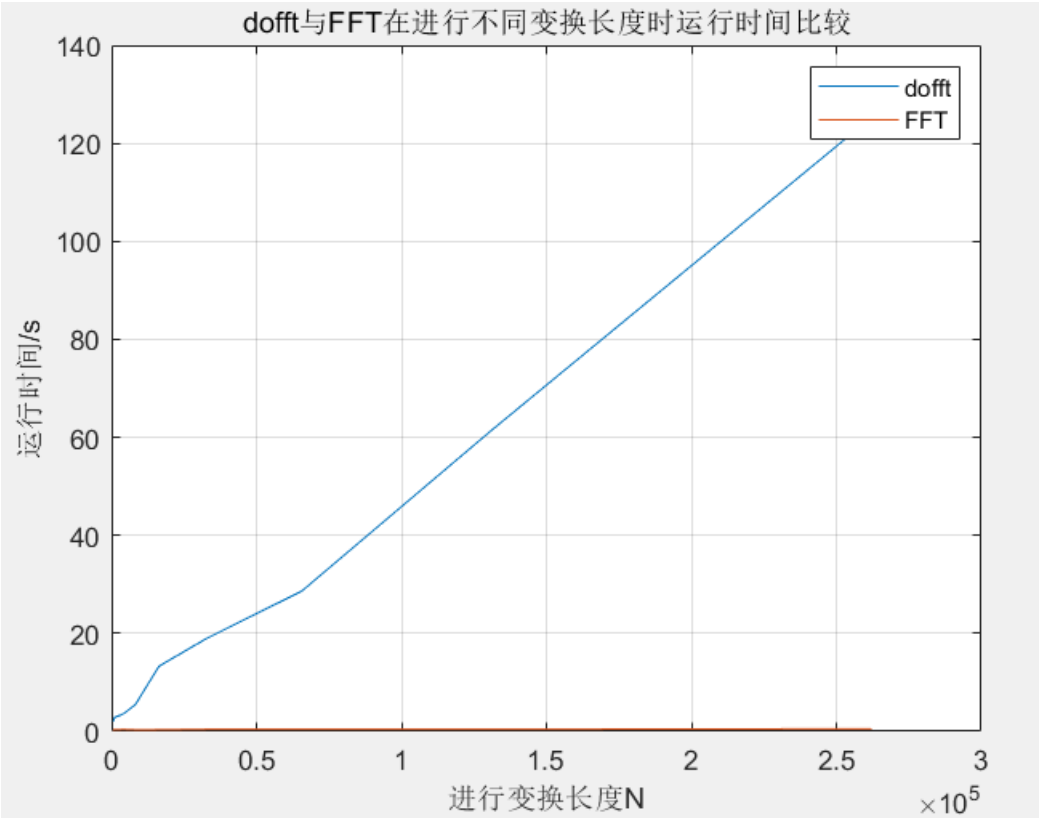
fft 测试程序：

```
clear all;
tic;
for f=1:20;
    clear all;
    matlab_fft;
    clear all;
end
toc;
```

运行时间记录如下：

算法 \ N	512	1024	2048	4096	8192
dofft	1.984634	2.814433	3.088558	3.579388	5.438975
FFT	0.349001	0.369433	0.365632	0.403413	0.334651
算法 \ N	16384	32768	65536	131072	262144
dofft	13.307104	18.961649	28.585727	61.441558	125.271445
FFT	0.350225	0.431561	0.430717	0.445077	0.530932

作图：



自己编写的 dofft 在进行变换长度的横坐标下近似线性增长。而 MATLAB 本身的 FFT 基本上随着 N 的增大，运行时间基本上没有变化。显然 dofft 性能比 fft 性能差。想必 MATLAB 中的 fft 函数进行了更多技巧和优化。

4、总结实验中根据实验现象得到的其他结论。

①实验中测运行时间，当前后两个程序运行时间相差不大时，可能时间大小有波动。比如 MATLAB 中的 fft 函数在做 2048 点计算时所用时间比 1024 点计算时间稍小，这与 MATLAB 当前占 CPU 和电脑状态相关。也会出现同一个程序在不同时刻测运行时间大小稍有差异，多测几次时间取平均时间长。
②DFT 在 N 较小时运行时间小于 dofft，说明 MATLAB 更优于计算向量。所以编写 MATLAB 程序时应尽量把 for 循环改为矩阵运算，尽量向量化。
③同样的算法，基于不同的编程，在运行速度上仍然会有很大的不同，比如在 MATLAB 中使用 for 循环编写，MATLAB 本身的 FFT，和用 C 语言编写的 FFT 在运行速度上都会有很大不同。所以提高编程技巧，了解程序具体流水线、地址开辟、循环嵌套等等如何对优化程序有很大意义。
④MATLAB 带有众多功能强大，高优化水平的函数，在编写程序时，尽可能查询 MATLAB 有无相关函数，充分利用，以提高编写的程序的执行效率，比如 dofft 中的 bitreorder。
⑤测量运行时间的时候要把绘图的函数注释掉，否者会占用程序大量运行时间。