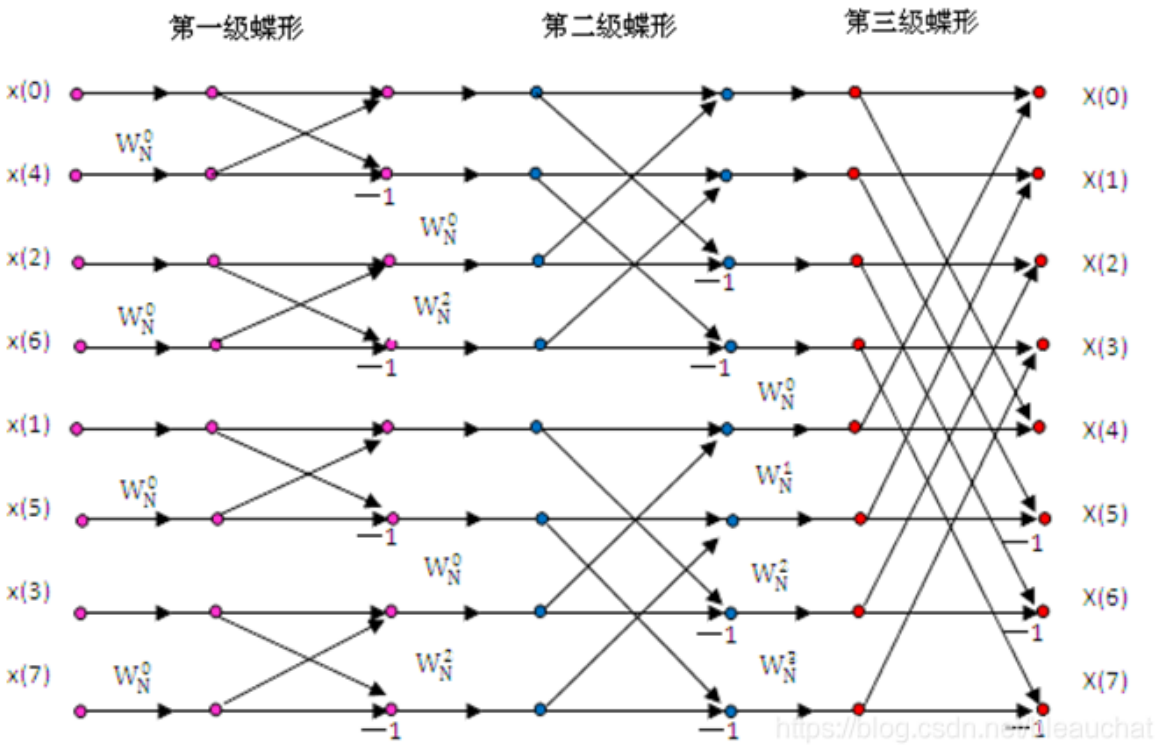
实验二报告

学号：PB20051061 姓名：牟真伟

1.FFT算法实现



算法参考以上蝶形图实现，先将输入的时域序列到位序，采用matlab内置函数bitrevorder()，由于matlab的向量运算较于循环较快，故将整个fft算法分成log2(len)，每层内使用向量化实现.用lay表示第几层，用len/2表示该层有几组蝶形运算，每组有2^lay个蝶形运算.每层按按行进行奇偶抽取后，则可以将蝶形运算中不乘旋转因子的一项和要乘旋转因子的一项分别抽取到矩阵x_even和x_odd，第lay层的x_even和x_odd的维度为(2 ^lay , len/2)，将每层的len/2个组蝶形运算分解到矩阵的len/2列上，每组2^lay个蝶形运算分解到矩阵的 2^lay 行上，每个蝶形运算的两个输入即是x_even和x_odd相同位置的两个数，需要将x_odd的每个元素乘上旋转因子，由于每列是一组蝶形运算，可用remat()函数将旋转因子转置，并复制为len/2列后再与x_odd相乘.

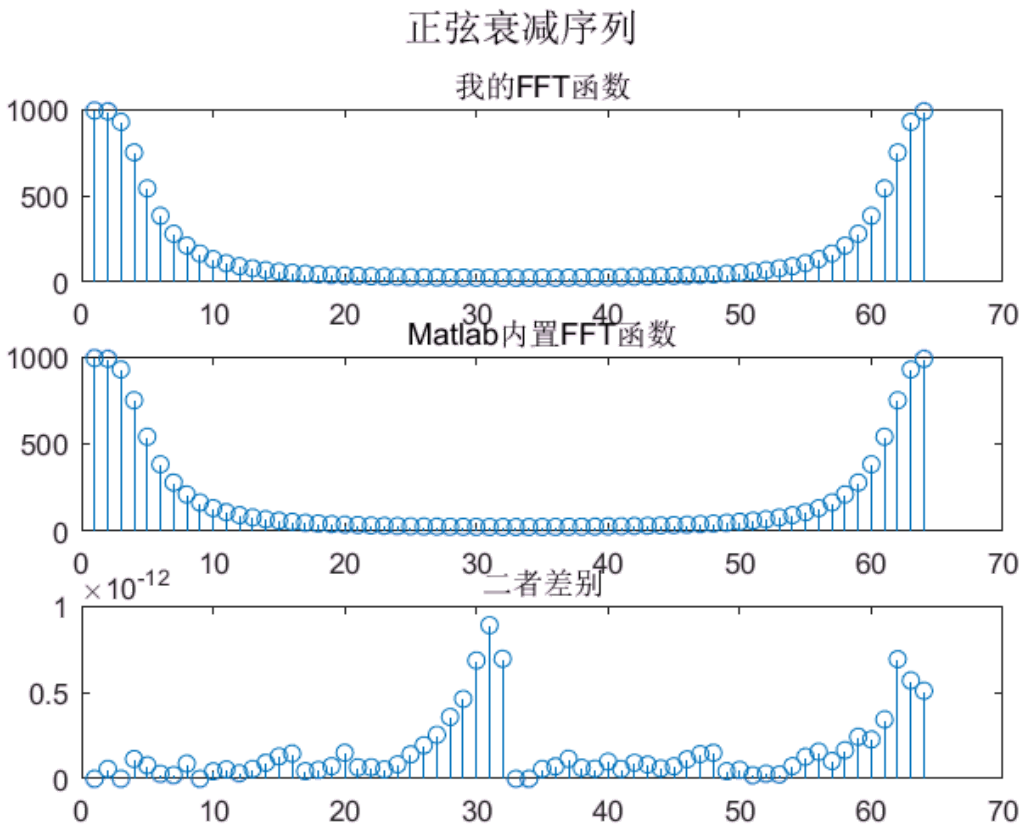
将蝶形运算的两个输出上下拼接，可让下一轮偶抽取为蝶形运算的第一个输入，奇抽取为蝶形运算的第二个输入.

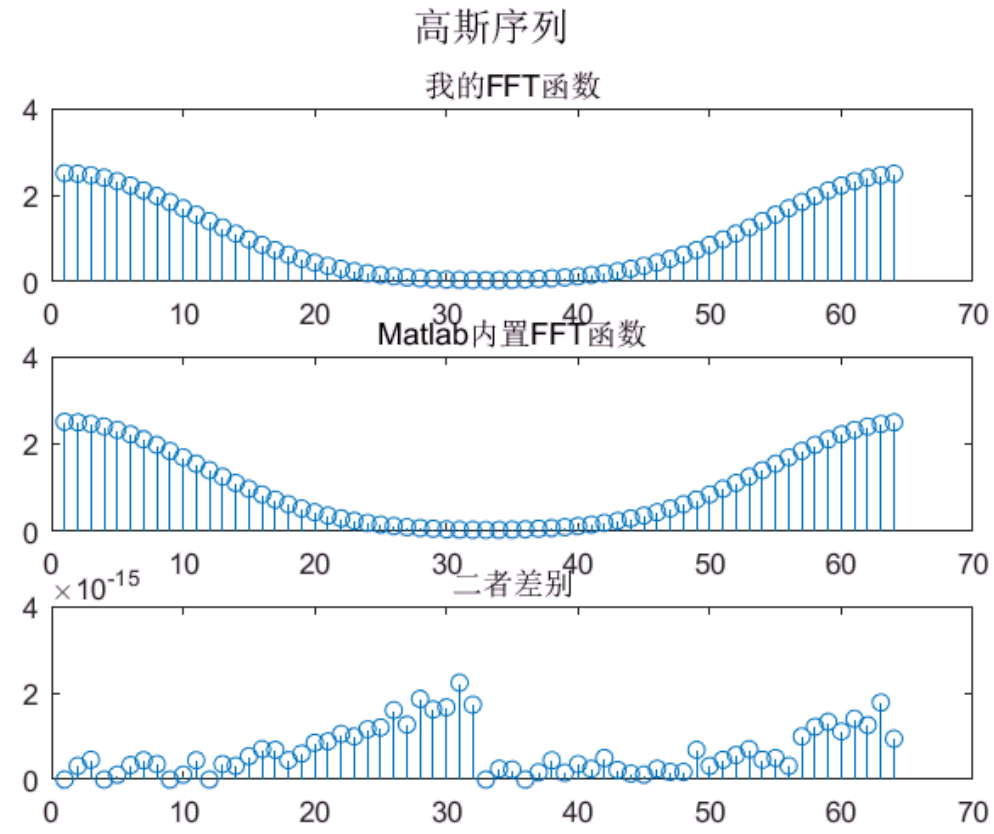
```
function [ xk ] = my_fft( xn )
%MY_FFT fft函数(输入为时域序列),输出为dft频域序列
len = length(xn);         % 序列长度
layer = log2(len);        % fft层数
xk = bitrevorder(xn);    % 输入倒位序
for lay = 1:layer
    x_even = xk(:,1:2:len);      % 按行偶抽取
    x_odd = xk(:,2:2:len);       % 按行奇抽取
```

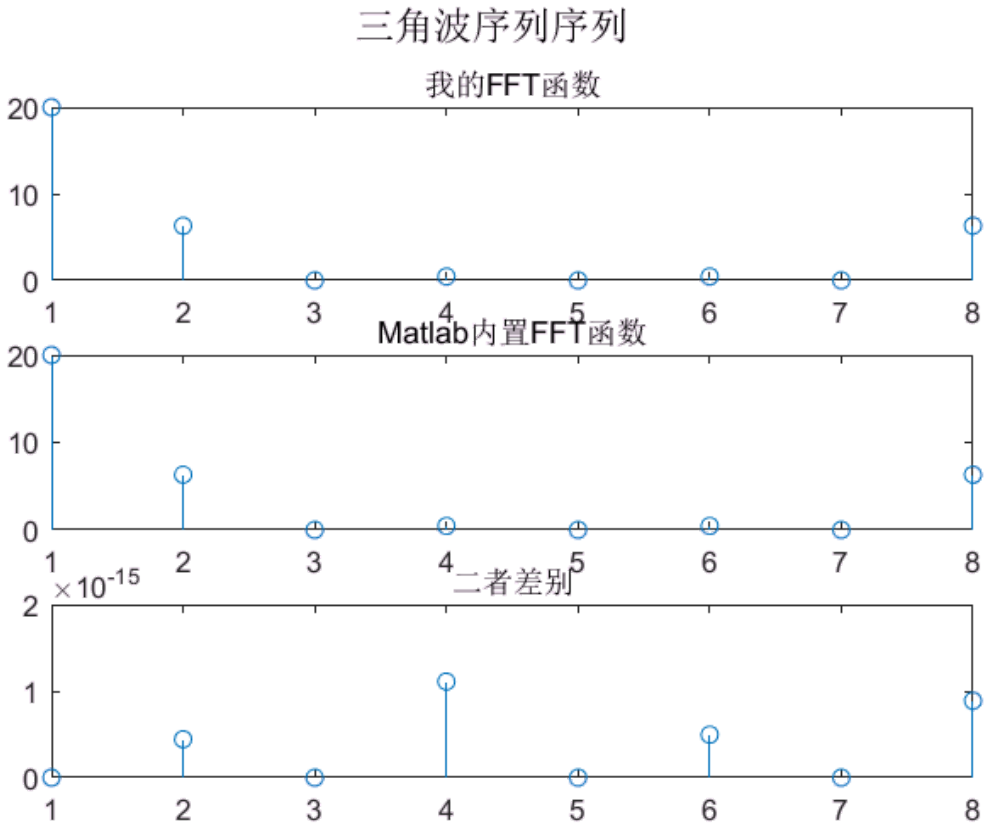
```
    % 第lay层 有len/2个组蝶形运算,每组有2^lay个蝶形运算
    % 将每层的len/2个组蝶形运算分解到矩阵的len/2列上
    % 将每组2^lay个蝶形运算分解到矩阵的2^lay行上

    N = 0:2^(lay-1)-1;
    Wn = exp((-1j*2*pi)/(2^lay));
    W = Wn.^N;
    % 将旋转因子转置,并复制为len/2列
    %W = repmat(W',1,len/2);
    % 给x_odd的每一行乘上相同的旋转因子
    x_odd = x_odd.*repmat(W',1,len/2);

    % x_odd和x_even矩阵对应位置即为一个蝶形运算的两个输入
    % 将蝶形运算的两个输出上下拼接,可让下一轮偶抽取为蝶形运算的第一个输入,奇抽取为蝶形运算的第二个输入
    xk = [x_even + x_odd; x_even - x_odd];
    len = len/2; % 更新len
end
xk = xk';
end
```

2.FFT算法验证

通过选取实验1中的典型信号序列做fft运算并于matlab内置fft进行对比,验证算法的有效性.







由上图可知,fft算法与matlab内置fft算法相差均为10的-10次方以上数量级,可以认为实现的fft算法有效.

3.FFT算法性能评估

- 我的FFT算法与Matlab的FFT算法比较

```
% 正弦衰减序列
clc;
A = 444.128;
alpha = 50*pi*sqrt(2);
omega0 = 50*pi*sqrt(2);
T = 1/1000;
n = 1:1024;
xa = A*exp(-alpha*T*n).*sin(omega0*T*n);

% 我的FFT算法性能测试
my_fft_time = 0;
for i=1:10000
tic;
my_fft(xa);
my_fft_time = my_fft_time + toc;
end
my_fft_time = my_fft_time/10000;

% Matlab的FFT算法性能测试
fft_time = 0;
for i=1:10000
tic;
fft(xa);
fft_time = fft_time + toc;
end
fft_time = fft_time/10000;
fprintf('%4.2f\n',my_fft_time/fft_time);
```

测试结果361.55

- 我的FFT算法与DTFT算法比较

```
% 正弦衰减序列
clc;
A = 444.128;
alpha = 50*pi*sqrt(2);
omega0 = 50*pi*sqrt(2);
T = 1/1000;
n = 1:1024;
xa = A*exp(-alpha*T*n).*sin(omega0*T*n);

% 我的FFT算法性能测试
my_fft_time = 0;
for i=1:100
tic;
my_fft(xa);
my_fft_time = my_fft_time + toc;
end
```

```
my_fft_time = my_fft_time/100;

% DTFT算法性能测试
dft_time = 0;
for i=1:100
tic;
dft(xa);
dft_time = dft_time + toc;
end
dft_time = dft_time/100;
fprintf('%4.2f\n',my_fft_time/dft_time);
```

测试结果0.2

FFT算法较DFT算法效率更高.

4.实验总结

fft算法较dft算法效率上有极大提升,在实现fft算法时,由于matlab的向量运算较于循环更快,可以将fft算法的实现尽量使用向量化,可以提高效率,实现fft算法时,可以利用一些matlab内置的函数,比如bitrevorder()实现倒位序,因为一般matlab内置实现比自己实现更快.