实验五　离散傅立叶变换及谱分析

一、　实验目的

1. 熟悉离散傅里叶级数变换变换原理，并掌握离散傅里叶级数正变换和反变换的 MATLAB 程序实现方法；
2. 熟悉离散傅里叶变换原理，并掌握离散傅里叶正变换和反变换的 MATLAB 程序实现方法；
3. 熟悉快速离散傅里叶变换原理，并掌握快速离散傅里叶正变换和反变换的 MATLAB 程序实现方法；
4. 通过对比实验，直观体会 DFT 和 FFT 的运行时间差异；
5. 通过实验对比原信号与先进行 FFT 后再进行 IFFT 的重构信号，分析误差及其原因，以便正确应用 FFT。

二、　实验原理与方法

1. 离散傅里叶级数正变换和反变换

原理：离散周期信号的离散傅里叶级数（DFS）的频谱是周期性的，因为时域的离散对应于频率的周期。离散周期序列的 DFS 离散谱是离散时间序列傅里叶变换（DTFT）主值序列连续谱的离散抽样，即在（0，2π）的频域区间上取 N 个点。

时域：离散整型时间变量的周期函数；

频域：离散整型频率变量的周期函数。

离散傅里叶级数正变换和反变换数学表达式如下：

正变换： $$\tilde{X}(k)=DFS[\tilde{x}(n)]=\sum_{n=0}^{N-1}\tilde{x}(n)e^{-j\frac{2\pi}{N}kn}=\sum_{n=0}^{N-1}\tilde{x}(n)W_N^{nk} \quad （5\text{-}1）$$

反变换： $$\tilde{x}(n)=IDFS[\tilde{X}(k)]=\frac{1}{N}\sum_{k=0}^{N-1}\tilde{X}(k)e^{-j\frac{2\pi}{N}kn}=\frac{1}{N}\sum_{k=0}^{N-1}\tilde{X}(k)W_N^{-nk} \quad （5\text{-}2）$$

其中，$\tilde{x}(n)$ 和 $\tilde{X}(k)$ 都是周期为 N 的周期序列，$W_N=e^{-j\frac{2\pi}{N}}$。

方法：MATLAB 中没有提供离散傅里叶级数的正变换和反变换函数，需要自己编译。MATLAB 参考程序如下：

- **离散傅里叶级数正变换 DFS 代码**

```
function [Xk] = dfs(xn,N)
% 计算离散傅里叶级数(DFS)系数
n = [0:1:N-1];                    % n 的行向量
k = [0:1:N-1];                    % k 的行向量
WN = exp(-j*2*pi/N);              % Wn 因子
nk = n'*k;                        % 产生一个含 nk 值的 N 乘 N 维矩阵
WNnk = WN .^ nk;                  % DFS 矩阵
Xk = xn * WNnk;                   % DFS 系数的行向量
```

- **离散傅里叶级数反变换 IDFS 代码**

```
function [xn] = idfs(Xk,N)
% 计算逆离散傅里叶级数(IDFS)
n = [0:1:N-1];                    % n 的行向量
k = [0:1:N-1];                    % k 的行向量
WN = exp(-j*2*pi/N);              % Wn 因子
nk = n'*k;                        % 产生一个含 nk 值的 N 乘 N 维矩阵
WNnk = WN .^ (-nk);               % IDFS 矩阵
xn = (Xk * WNnk)/N;               % IDFS 值的行向量
```

2. 离散傅里叶正变换和反变换

原理：离散傅里叶变换（DFT），是傅里叶变换在时域和频域上都呈现离散的形式，将时域信号的采样变换为在离散时间傅里叶变换（DTFT）频域的采样。在形式上，变换两端（时域和频域上）的序列是有限长的，而实际上这两组序列都应当被认为是离散周期信号的主值序列。即使对有限长的离散信号作 DFT，也应当将其看作经过周期延拓成为周期信号再作变换。

时域：离散整型时间变量的非周期函数；

频域：离散整型频率变量的非周期函数。

离散傅里叶正变换和反变换数学表达式如下：

正变换： $$X(k) = DFT[x(n)] = \sum_{n=0}^{N-1} x(n)W_N^{kn}, \text{ k=0, 1, }\cdots\text{, N-1} \qquad (5\text{-}3)$$

反变换：$X(n)=IDFT[X(k)]=\frac{1}{N}\sum_{k=0}^{N-1}X(k)W_N^{-kn}$, n=0, 1, ⋯, N-1　　（5-4）

其中，$W_N=e^{-j\frac{2\pi}{N}}$。

方法：一般在实际应用中会用 FFT 来实现信号的离散傅里叶变换，因此 MATLAB 中并未提供原始的离散傅里叶正变换和反变换函数，需要自己编译。MATLAB 参考程序如下：

- **离散傅里叶正变换 DFT 代码**

```
function [Xk] = dft(xn,N)
% 计算离散傅里叶叶变换
n = [0:1:N-1];                 % n 的行向量
k = [0:1:N-1];                 % k 的行向量
WN = exp(-j*2*pi/N);           % Wn 因子
nk = n'*k;                     % 产生一个含 nk 值的 N 乘 N 维矩阵
WNnk = WN .^ nk;               % DFT 矩阵
Xk = xn * WNnk;                % DFT 系数的行向量
```

- **离散傅里叶反变换 IDFT 代码**

```
function [xn] = idft(Xk,N)
% 计算逆离散傅里叶变换
n = [0:1:N-1];                 % n 的行向量
k = [0:1:N-1];                 % k 的行向量
WN = exp(-j*2*pi/N);           % Wn 因子
nk = n'*k;                     % 产生一个含 nk 值的 N 乘 N 维矩阵
WNnk = WN .^ (-nk);            % IDFT 矩阵
xn = (Xk * WNnk)/N;            % IDFT 的行向量
```

3. 快速傅里叶正变换和反变换

原理：快速傅里叶正变换（FFT）的基本思想是把原始的 N 点序列，依次分解成一系列的短序列。充分利用 DFT 计算式中指数因子 所具有的对称性质和周期性质，进而求出这些短序列相应的 DFT 并进行适当组合，达到删除重复计算，减少乘法运算和简化结构的目的。

方法：计算离散傅里叶变换的快速方法，有按时间抽取的 FFT 算法和按频率抽取的 FFT 算法。前者是将时域信号序列按偶奇分排，后者是将频域信号序列按偶奇分排。它们都借助于的两个特点：一是周期性；二是对称性。

MATLAB 提供了快速傅里叶正变换函数 fft 和快速傅里叶反变换函数 ifft，其语句格式为

- Y = fft(X)　%对 X 做离散傅里叶正变换，输出为 Y。
- Y = fft(X,n)　%若 X 点的个数少于 n，则在后面加 0，凑到 n；若 X 的点的个数多于 n，则删除多余的数。
- X = ifft(Y)　%对 Y 做离散傅里叶反变换，输出为 X。
- X = ifft(Y,n)　%若 X 点的个数少于 n，则在后面加 0，凑到 n；若 X 的点的个数多于 n，则删除多余的数。

三、　实验内容

1. 离散傅里叶级数正变换和反变换

%离散傅里叶级数正变换和反变换。

代码：

```
xn=[1,2,3,4];
N=4;
xk=dfs(xn,N)   %  离散傅里叶级数正变换
xn1=idfs(xk,N) %离散傅里叶级数反变换
```

结果：

xk =   10.0000 + 0.0000i   -2.0000 + 2.0000i   -2.0000 - 0.0000i   -2.0000 - 2.0000i

xn1 =   1.0000 - 0.0000i   2.0000 - 0.0000i   3.0000 - 0.0000i   4.0000 + 0.0000i

2. 离散傅里叶正变换和反变换

%离散傅里叶变换正变换和反变换。

代码：

```
xn=[4,3,2,1];
N=4;
```

xk=dft(xn,N) % 离散傅里叶正变换

xn1=idft(xk,N) %离散傅里叶反变换

结果：

xk = 10.0000 + 0.0000i 2.0000 - 2.0000i 2.0000 - 0.0000i 2.0000 + 2.0000i

xn1 = 4.0000 - 0.0000i 3.0000 + 0.0000i 2.0000 + 0.0000i 1.0000 + 0.0000i

3. 快速傅里叶正变换和反变换

%快离散傅里叶变换正变换和反变换。

代码：

xn=[3,4,1,2];

N=4;

xk=fft(xn,N) % 离散傅里叶正变换

xn1=ifft(xk,N) %离散傅里叶反变换

结果：

xk = 10.0000 + 0.0000i 2.0000 - 2.0000i -2.0000 + 0.0000i 2.0000 + 2.0000i

xn1 = 3 4 1 2

4. DFT 和 FFT 的运行时间对比

% DFT 和 FFT 的运行时间对比。

代码：

N=16384; % 2^14 序列长度

M=500; % 非零值长度

x=[1:M,zeros(1,N-M)]; %非零值的其它部分补零

t=cputime; %记录时间

y1=dft(x,N); %离散傅里叶变换

Time_dft=cputime-t %离散傅里叶变换时间，不同的计算机上运行时间会不一样

t1=cputime;

```
y2=fft(x,N);
Time_fft=cputime-t1    %快速离散傅里叶变换时间，不同的计算机上运行时间会不一样
```

结果：

```
Time_dft =    416.6094
Time_fft =      0.0313
```

5. 原信号与先进行 FFT 后再进行 IFFT 的重构信号比较

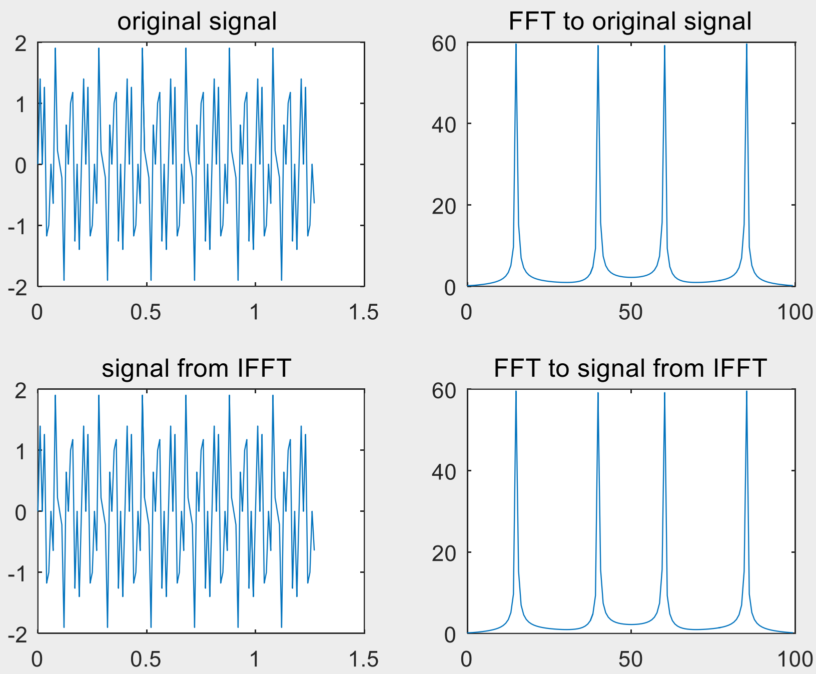
%无噪声信号与先进行 FFT 后再进行 IFFT 的重构信号比较。

代码：

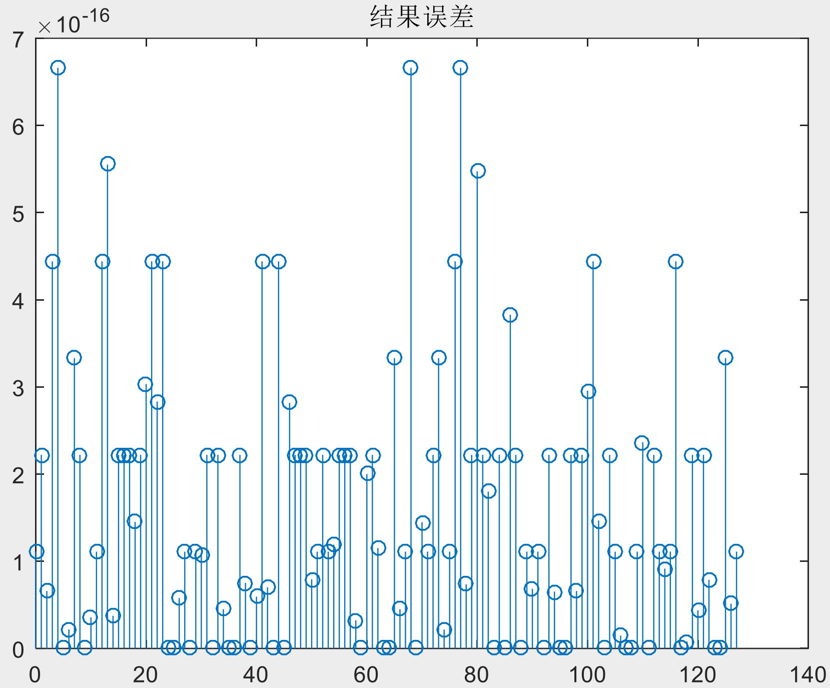
```
fs=100;
N=128;
n=0:N-1;
t=n/fs;
x=sin(2*pi*40*t)+sin(2*pi*15*t);   %无噪声信号
subplot(2,2,1)
plot(t,x)
title('original signal ')
y=fft(x,N);
mag=abs(y);
f=(0:length(y)-1)'*fs/length(y);
subplot(2,2,2)
plot(f,mag)
title('FFT to original signal')
xifft=ifft(y);
magx=real(xifft);
ti=[0:length(xifft)-1]/fs;
subplot(2,2,3)
plot(ti,magx);
title('signal from IFFT ')
```

```
yif=fft(xifft,N);
mag=abs(yif);
subplot(2,2,4)
plot(f,mag)
title('FFT to signal from IFFT')
error = xifft-x;%结果误差
figure;
stem(n,abs(error));%绘图，误差幅度
title('结果误差')
```

结果：



结果误差



四、 学生作业

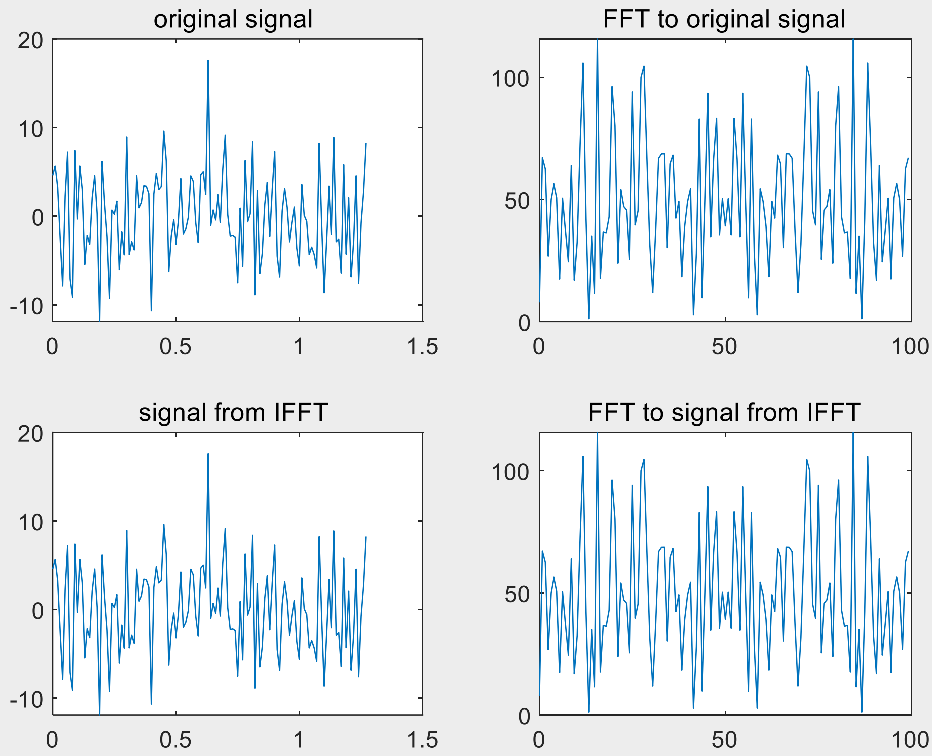
1. 将无噪声信号与先进行 FFT 后再进行 IFFT 的重构信号比较实验中的信号添加随机噪声：x=sin(2*pi*40*t)+sin(2*pi*15*t)+ 5*randn(1,N)，做对比实验，并绘制图形。

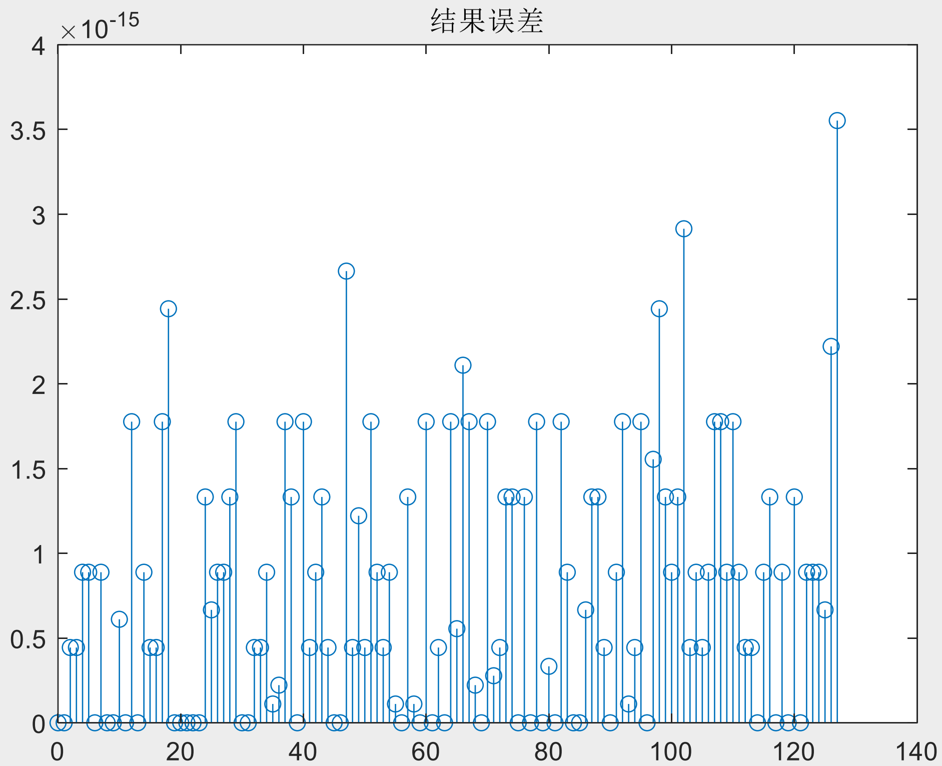
运行代码:

```
fs=100;
N=128;
n=0:N-1;
t=n/fs;
x=sin(2*pi*40*t)+sin(2*pi*15*t)+ 5*randn(1,N);
subplot(2,2,1)
plot(t,x)
title('original signal ')
y=fft(x,N);
mag=abs(y);
f=(0:length(y)-1)'*fs/length(y);
subplot(2,2,2)
plot(f,mag)
title('FFT to original signal')
xifft=ifft(y);
magx=real(xifft);
ti=[0:length(xifft)-1]/fs;
```

```
subplot(2,2,3)
plot(ti,magx);
title('signal from IFFT ')
yif=fft(xifft,N);
mag=abs(yif);
subplot(2,2,4)
plot(f,mag)
title('FFT to signal from IFFT')
error = xifft-x;%结果误差
figure;
stem(n,abs(error));%误差幅度
title('½á¹ûÎó²î')
```

运行截图：





2. 采用时间抽取的方法，通过 2 个 1024 点的 FFT 变换的组合实现 2048 点的 FFT 变换，并计算直接变换结果和组合变换结果之间的误差并绘制图形。

```
% 生成信号
N = 2048;
t = linspace(0, 1, N);
x = sin(2*pi*10*t) + sin(2*pi*20*t) + sin(2*pi*30*t);

% 将信号分成两部分
x1 = x(1:2:end);
x2 = x(2:2:end);

%  ·分别计算两个小的FFT变换
X1 = fft(x1);
X2 = fft(x2);

% 计算组合的FFT变换
X = zeros(1, N);
for k = 1:N/2
    X(k) = X1(k) + exp(-1i*2*pi*(k-1)/N)*X2(k);
    X(k+N/2) = X1(k) - exp(-1i*2*pi*(k-1)/N)*X2(k);
```

```
end

%计算直接变换的FFT
X_direct = fft(x);

%绘制结果
subplot(2,1,1);
plot(abs(X));
title('FFT via Time Decimation');

subplot(2,1,2);
plot(abs(X_direct));
title('Direct FFT');

% ¼ÆËãÎó²î²¢Êä³ö
MSE = mean(abs(X - X_direct).^2);
disp(['MSE between FFT via Time Decimation and Direct FFT: ' num2str(MSE)]);
```

运行截图:

```
>> Code2
MSE between FFT via Time Decimation and Direct FFT:  2.0185e-28
```