2020-2학기 임베디드신호처리실습

결과보고서

Lab4. FFT

2016146041 정성균

2014146012 박동훈

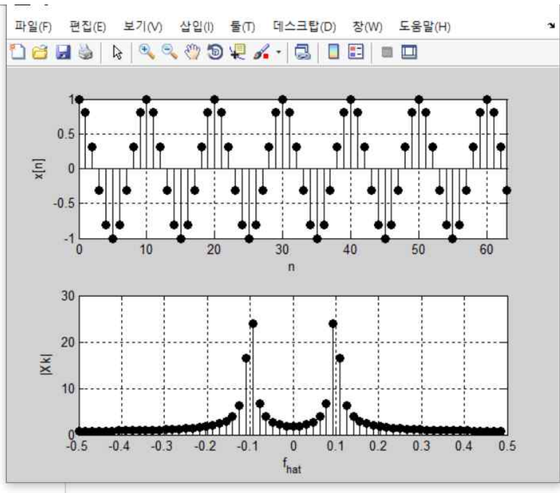
- 실습 1.2.3절에 설명한 시분할 알고리즘을 이용해 N-point FFT 함수를 작성하라. (m-file function으로 구현할 것)
  - 입력
    - x : 이산신호 $x[n]$, 신호의 길이는 N, $n = 0, 1, \cdots, N-1$
  - 출력
    - f_hat : 이산주파수 $\hat{f}$, $\hat{f} = 0, \frac{1}{N}, \frac{2}{N}, \cdots, \frac{N-1}{N}$
    - Xk : 스펙트럼 X_k, 복소수이며 길이는 N
    - N_mult : 곱셈 연산의 횟수

```
function [f_hat,X_k,N_mult] = my_fft(x,N)
lognum=log2(N); % log2에 입력신호의 길이를 넣어 lognumber 생성,
f_hat = zeros(1,N); % 1부터 N까지 zero padding해서 f_hat에 저장 (변환된 신호를 담기 위한 그릇)
N_mult = 0; % 연산 횟수를 구하기 위한 변수
X_k = zeros(N,lognum+1); % N x lognumber 행렬만큼 (초기배열)
% 신호 길이 N이 2의 거듭제곱수 꼴이 아니면 여기서 오류가 난다.(lognum이 정수가 아니게 되서)
W = zeros(N/2,lognum); % 회전인자의 초기배열. 대칭성을 가져서 N/2만큼

%Setting the array
lognumber1=floor(log2(N)); % 배열 설정
f=zeros(N,lognumber1); % 배열
f(:,lognumber1)=transpose(0:N-1);  % f(:log) = 입력 신호 x[n]으로보면됨
n = 0:N-1;
n_f = bin2dec(fliplr(dec2bin(n))); % 역비트순 구현
%10진수를 2진수로 변환 후 그걸 뒤집어서 다시 2진수를 10진수로 변환하면 역비트순을 구현할 수 있음

for k=1:N % FFT를 수행하기 위하여 역비트순을 한 것을 대입
    X_k(k,1)=x(n_f(k)+1); % 총 N개가 1열에 들어간다. x값을 각각 넣어줌
end

% 회전인자 생성
for n=1:lognum
    temp=N/(2^(lognum-n)); % sample number n번째
    for k=1:temp/2 % 단계에 따라서 곱해줄 회전인자를 각각 만들어줌.
        W(k,n)=exp(-1j*2*pi/temp*(k-1)); % 1개, 2개, 4개 ...
    end
end

% 버터플라이 알고리즘 구현
for n=1:lognum  % 1단계, 2단계, ... , lognum단계로 진입시키기 위한 변수 n
    temp=N/(2^(lognum-n)); % n단계의 샘플 수(temp만큼 묶어서 샘플링).점차 증가함
    max=2^(lognum-n); % n번째 단계에서 진행하는 DFT연산의 횟수
    for j=1:max  % j값을 증가시켜서 값을 증가시킴.
        offset=(j-1)*temp;  % offset는 행의 위치를 지정한다. offset이 증가할수록 행의 시작 위치가 증가함.
        for k=1:temp/2 % k는 연산을 할 때 행의 시작 위치를 가리킨다. n단계에서 k는 2^(n-1)번 있음
            % k와 offset
            % offset+k 번째라면
            X_k(offset+k,n+1)=X_k(offset+k,n)+W(k,n)*X_k(offset+temp/2+k,n);
            % offset+k+temp/2 번째를 같이 구할 수 있다. (원점대칭하면 - 부호만 붙여주면 되기 때문)
            X_k(offset+temp/2+k,n+1)=X_k(offset+k,n)-W(k,n)*X_k(offset+temp/2+k,n);
            N_mult = N_mult+1;
        end
    end
end

for i = 0:N-1
    f_hat(i+1)= i/N;
end


f_hat = f_hat -0.5; % -0.5만큼 평행 이동
X_k = X_k';
X1 = X_k(lognum+1 ,N/2+1:N);
X2 = X_k(lognum+1 ,1:N/2);
X_k(lognum+1,:) = [X1,X2];
```

- 실습 DEMO 위에서 구현한 N-point FFT를 이용해 다음 이산신호 $x[n]$의 크기 스펙트럼을 구하고 그래프에 표시하라. (단, 이산주파수 $\hat{f} \in [-\frac{1}{2}, \frac{1}{2}]$에 대해 그려라.) (그림 7 참고)
  - $x[n] = \cos(2\pi \hat{f}_0 n),\ n = 0, 1, 2, \cdots, N-1$
  - $\hat{f}_0 = 0.1,\ N = 64$

- 실습 위 신호를 FFT하는데 필요한 곱셈 연산의 횟수는 얼마인가? 신호 $x[n]$을 DFT했을 때 필요한 곱셈 횟수와 비교하라.

```
clear; clc; close all;
N=64;
n=0:N-1;
lognumber=log2(N); % number of samples in the order of 2
x=cos(2*pi*0.1*n);
[f_hat,X_k,N_mult] = my_fft(x,N);
[f_halt,X_k1,N_mult1] = dft(x);

subplot(2,1,1);
stem(n,x,'k','MarkerFacecolor','k');
grid on;
axis([0,63,-1,1])
xlabel('n');ylabel('x[n]');

subplot(2,1,2);
stem(f_hat,abs(X_k(lognumber+1,:)),'k','MarkerFacecolor','k')
grid on;
xlabel('f_h_a_t');ylabel('|Xk|');

N_num = N_mult;
```



N_mult	192
N_mult1	4096

주어진 조건을 대입하고 my_fft에 신호를 대입하여 그려준 결과입니다.
DFT와의 연산횟수를 비교하기 위해 dft 함수도 넣어주었습니다. 결과를 확인해보면 FFT는 $\frac{N}{2} * \log_2 N$ 공식에 N=64를 대입하면 정확히 192가 나오고
DFT는 N^2에 N=64를 대입하면 4096이 나오게 됩니다.
DFT의 연산량이 FFT에 비해 20배이상 높은 것을 알수 있습니다.

3.2 DFT와의 비교 – 스펙트럼

- 실습 DEMO 다음과 같은 신호 $x[n]$을 N-point DFT와 N-point FFT를 이용해 크기 스펙트럼을 구하라. (그림 8 ∼ 11 참고)

$$x[n] = 0.3\cos(2\pi \hat{f}_1 n) + 0.8\sin(2\pi \hat{f}_2 n)$$

  – $n = 0, 1, 2, \cdots, N-1$
  – $\hat{f}_1 = 0.1,\ \hat{f}_2 = 0.3$
  – $N = 16,\ 32,\ 64,\ 128$

(1) N = 16

```
clear;
clc;
N=16;
n=0:N-1;
lognumber=log2(N);
x=0.3*cos(2*pi*0.1*n)+0.8*sin(2*pi*0.3*n);
[f_hat,X_k,N_mult] = my_fft(x,N);
[f_hat1,X_k1,N_mult1] = my_dft(x);

figure(1)

subplot(2,1,1);
stem(n,x,'k','MarkerFacecolor','k');
grid on;
axis([0,15,-2,2])
xlabel('n');ylabel('x[n]');

subplot(2,1,2);
plot(f_hat1,abs(X_k1),'-ok')
```


```
clear;
clc;
N=16;
n=0:N-1;
lognumber=log2(N);
x=0.3*cos(2*pi*0.1*n)+0.8*sin(2*pi*0.3*n);
[f_hat,X_k,N_mult] = my_fft(x,N);
[f_hat1,X_k1,N_mult1] = my_dft(x);

figure(1)

subplot(2,1,1);
stem(n,x,'k','MarkerFacecolor','k');
grid on;
axis([0,15,-2,2])
xlabel('n');ylabel('x[n]');

subplot(2,1,2);
plot(f_hat1,abs(X_k1),'-ok')
```

(2) N = 32

```
clear;
clc;
N=32;
n=0:N-1;
lognumber=log2(N);
x=0.3*cos(2*pi*0.1*n)+0.8*sin(2*pi*0.3*n);
[f_hat,X_k,N_mult] = my_fft(x,N);
[f_hat1,X_k1,N_mult1] = my_dft(x);

figure(2)

subplot(2,1,1);
stem(n,x,'k','MarkerFacecolor','k');
grid on;
axis([0,31,-2,2])
xlabel('n');ylabel('x[n]');

subplot(2,1,2);
plot(f_hat1,abs(X_k1),':ok')
grid on;
xlabel('f_h_a_t');ylabel('|Xk|');
DFT_N = N_mult1;

hold on;
plot(f_hat,abs(X_k(lognumber+1,:)),':xr')
grid on;
xlabel('f_h_a_t');ylabel('|Xk|');
FFT_num = N_mult;

legend('DFT','FFT');
```

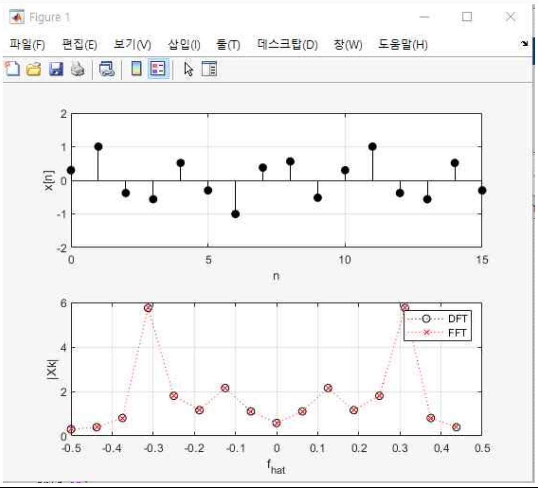
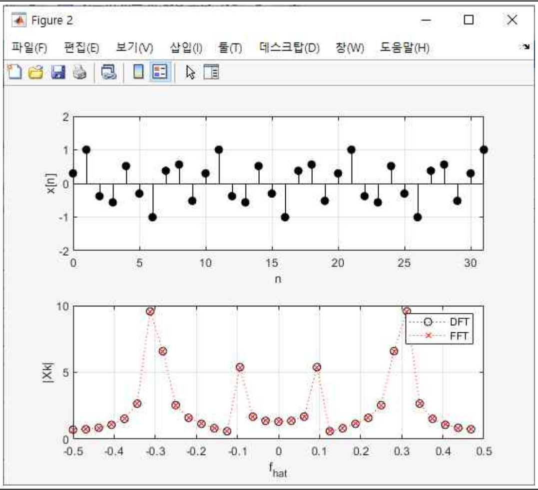
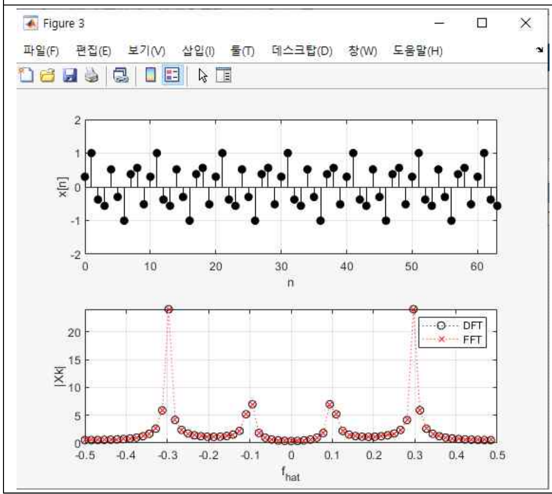
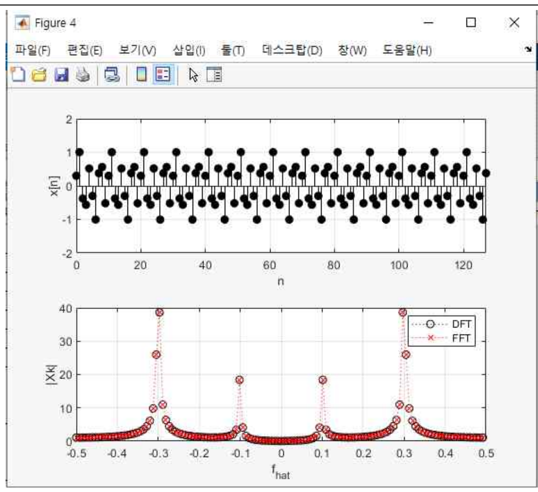
(3) N = 64

```
clear;
clc;
N=64;
n=0:N-1;
lognumber=log2(N);
x=0.3*cos(2*pi*0.1*n)+0.8*sin(2*pi*0.3*n);
[f_hat,X_k,N_mult] = my_fft(x,N);
[f_hat1,X_k1,N_mult1] = my_dft(x);

figure(3)

subplot(2,1,1);
stem(n,x,'k','MarkerFacecolor','k');
grid on;
axis([0,63,-2,2])
xlabel('n');ylabel('x[n]');

subplot(2,1,2);
plot(f_hat1,abs(X_k1),'-ok')
```


```
clear;
clc;
N=64;
n=0:N-1;
lognumber=log2(N);
x=0.3*cos(2*pi*0.1*n)+0.8*sin(2*pi*0.3*n);
[f_hat,X_k,N_mult] = my_fft(x,N);
[f_hat1,X_k1,N_mult1] = my_dft(x);

figure(3)

subplot(2,1,1);
stem(n,x,'k','MarkerFacecolor','k');
grid on;
axis([0,63,-2,2])
xlabel('n');ylabel('x[n]');

subplot(2,1,2);
plot(f_hat1,abs(X_k1),'-ok')
```

(4) N = 128

```

clear;
clc;
N=128;
n=0:N-1;
lognumber=log2(N);
x=0.3*cos(2*pi*0.1*n)+0.8*sin(2*pi*0.3*n);
[f_hat,X_k,N_mult] = my_fft(x,N);
[f_hat1,X_k1,N_mult1] = my_dft(x);


figure(4)

subplot(2,1,1);
stem(n,x,'k','MarkerFacecolor','k');
grid on;
axis([0,127,-2,2])
xlabel('n');ylabel('x[n]');

subplot(2,1,2);
plot(f_hat1,abs(X_k1),':ok')
grid on;
xlabel('f_h_a_t');ylabel('|Xk|');
DFT_N = N_mult1;

hold on;
plot(f_hat,abs(X_k(lognumber+1,:)),':xr')
grid on;
xlabel('f_h_a_t');ylabel('|Xk|');
FFT_num = N_mult;

legend('DFT','FFT');
```

<table>
<tr><th>N = 16</th><th>N = 32</th></tr>
<tr><td>
</td><td>
</td></tr>
<tr><th>N = 64</th><th>N = 128</th></tr>
<tr><td>
</td><td>
</td></tr>
</table>

네 개의 그래프 모두 DFT와 FFT의 결과가 같은 것을 볼 수 있습니다. N이 증가할수록 [-0.5 ~ 0.5] 범위 안에 많은 점이 찍히기 때문에 DTFT의 결과와 비슷해집니다. 또한 두 알고리즘 모두 N이 증가할수록 더해지는 횟수가 많아져서 결과값들의 전체적인 크기가 증가하는 것을 발견했습니다. DFT와 FFT는 알고리즘의 차이는 있지만, 결과 자체는 똑같다는 것을 확인하였습니다.

3.3 DFT와의 비교 – 연산복잡도

- 실습 **DEMO** 다음과 같은 신호 $x[n]$을 N-point DFT와 N-point FFT를 이용해 스펙트럼을 구하고 N에 따른 곱셈 연산횟수, FFT와 DFT의 곱셈 연산횟수의 비율을 측정해 그래프에 표시하라. (그림 12 참고)

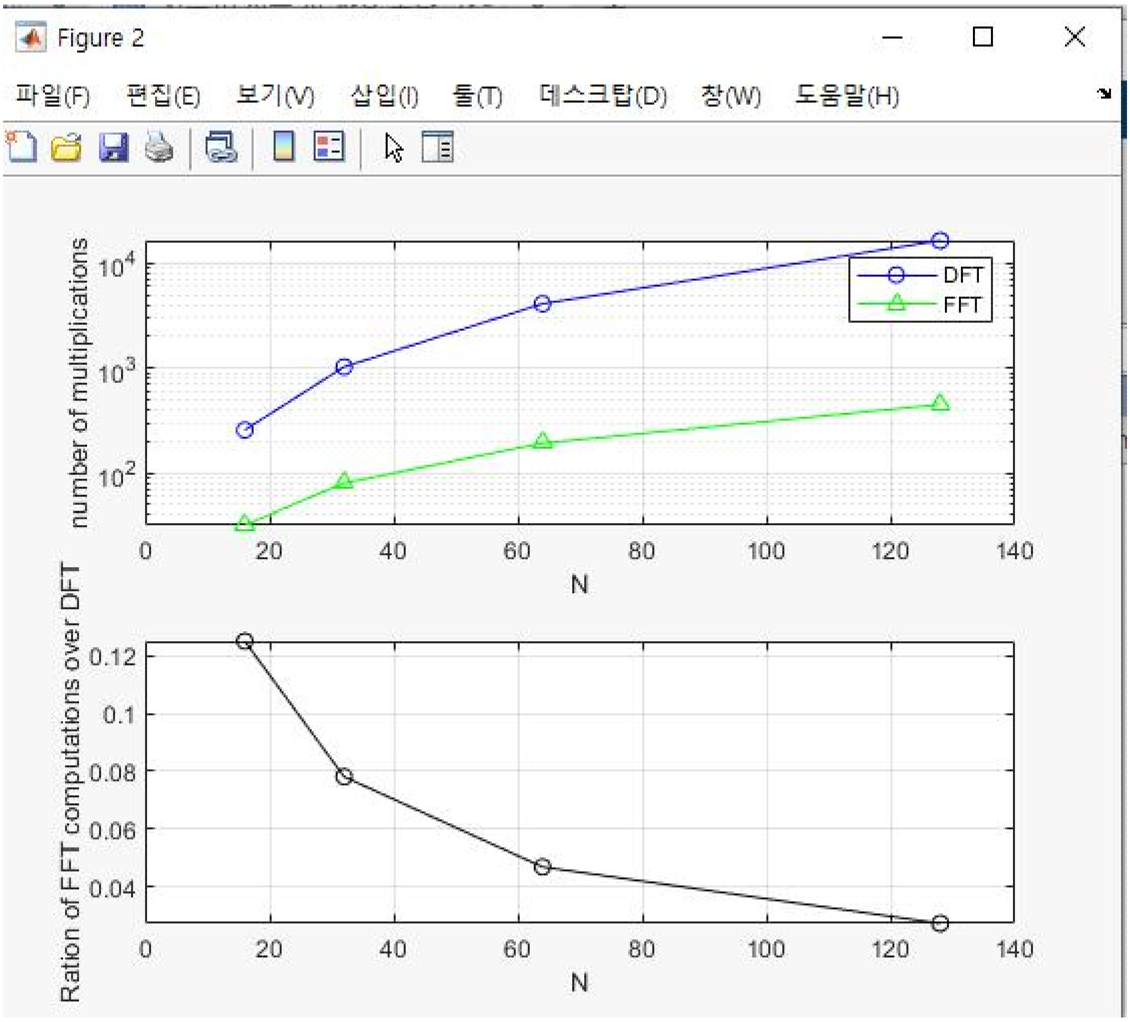
$$x[n] = 0.3\cos(2\pi \hat{f}_1 n) + 0.8\sin(2\pi \hat{f}_2 n)$$

  - $n = 0, 1, 2, \cdots, N-1$
  - $\hat{f}_1 = 0.1,\ \hat{f}_2 = 0.3$
  - $N = 16,\ 32,\ 64,\ 128$

```
clear;
clc;
N1=16;
n=0:N1-1;
x=0.3*cos(2*pi*0.1*n)+0.8*sin(2*pi*0.3*n);
[f_hat,X_k,N_mult1] = my_fft(x,N1);
[f_hat1,X_k1,N_mult2] = my_dft(x);

FFT_num1 = N_mult1;
DFT_N1 = N_mult2;
%------------------------------------------------------------------
N2=32;
n=0:N2-1;
x=0.3*cos(2*pi*0.1*n)+0.8*sin(2*pi*0.3*n);
[f_hat2,X_k2,N_mult3] = my_fft(x,N2);
[f_hat3,X_k3,N_mult4] = my_dft(x);

FFT_num2 = N_mult3;
```

```
DFT_N2 = N_mult4;
%------------------------------------------------------------------
N3=64;
n=0:N3-1;
x=0.3*cos(2*pi*0.1*n)+0.8*sin(2*pi*0.3*n);
[f_hat4,X_k4,N_mult5] = my_fft(x,N3);
[f_hat5,X_k5,N_mult6] = my_dft(x);

FFT_num3 = N_mult5;
DFT_N3 = N_mult6;
%------------------------------------------------------------------
N4=128;
n=0:N4-1;
x=0.3*cos(2*pi*0.1*n)+0.8*sin(2*pi*0.3*n);
[f_hat6,X_k6,N_mult7] = my_fft(x,N4);
[f_hat7,X_k7,N_mult8] = my_dft(x);

FFT_num4 = N_mult7;
```

```
DFT_N4 = N_mult8;
%------------------------------------------------------------------
DFT = [DFT_N1,DFT_N2,DFT_N3,DFT_N4];
FFT = [FFT_num1,FFT_num2,FFT_num3,FFT_num4];
n =[N1,N2,N3,N4];
figure(2)
subplot(2,1,1)
semilogy(n,DFT,'-ob',n,FFT,'-^g')
grid on;
xlabel('N');ylabel('number of multiplications');
legend('DFT','FFT');

subplot(2,1,2)
Ratio = FFT./DFT;
plot (n,Ratio,'-ok')
grid on;
xlabel('N');ylabel('Ration of FFT computations over DFT');
```



좌측 그래프는 FFT와 DFT의 곱셈 연산량을 보여주는 그래프이고 우측 그래프는 FFT연산량을 DFT연산량으로 나눠준 값을 보여주는 그래프입니다. 좌측 그래프에서 N값이 커질수록 그래프 간격이 더 벌어지는데, N이 증가할 때마다 FFT의 효율이 DFT보다 훨씬 좋아진다는 것을 알 수 있습니다. 우측의 그래프는 N이 증가함에 따라 분모에 있는 DFT의 곱셈 연산량이 분자에 있는 FFT 곱셈 연산량보다 증가량이 훨씬 커져서, 분수값이 점점 작아지는 것을 볼 수 있습니다.