REPORT

DCT 결과 보고서

전자 - 임베디드 전공
4조
주성민

목차

1. 개요

- DCT 변환과 FFT변환의 스케일링 (정규화)
- DCT를 활용한 간단한 압축 알고리즘의 이해

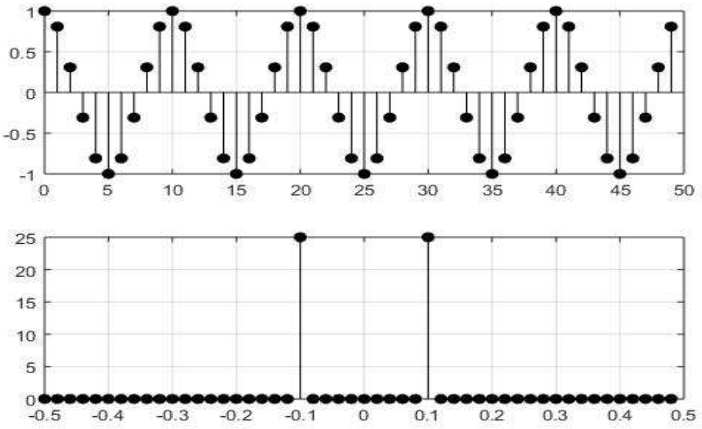
2. 본문

- DCT 변환과 FFT변환의 스케일링 (정규화)

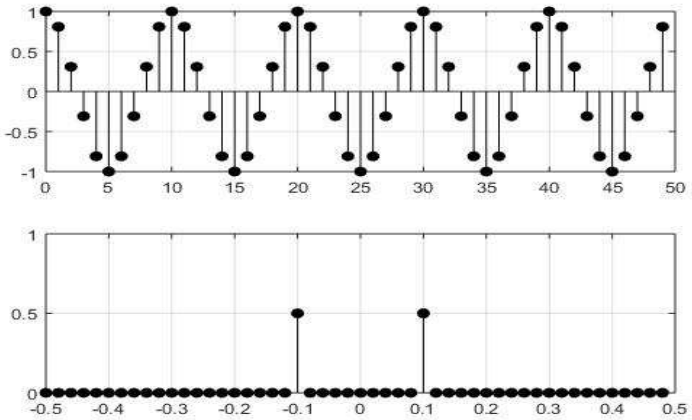
예비보고서를 작성하면서 DCT 변환과 FFT 변환의 공통점과 차이점 등을 자세히 공부했었다. 이론적인 공부 외에 실습을 진행하기 위해서 매트랩으로 DCT 변환과 FFT변환의 스펙트럼을 보기 위한 스케일링 알고리즘에 대한 공부가 추가적으로 필요했다. FFT 변환은 DFT변환 수식을 기반으로 하는 알고리즘이기 때문에 먼저 DFT 변환의 수식으로 스케일링을 알아보려고 한다.

$$X_k = \sum_{n=0}^{N-1} x[n] e^{-j2\pi \frac{k}{N} n}, \; k = 0, 1, 2, \cdots N-1$$

이전의 실습들을 통해 k가 증가할 때마다 N번의 곱셈연산이 일어난다는 것을 알고 있다. 이 말을 수학적으로 해석해보면 X[k]의 점 하나하나에 N개의 x[n]성분이 더해지기 때문에 N이 증가할수록 X[k]의 전체적인 평균값은 증가하게 된다.
다음은 이전 실습결과였던 cos함수의 dft변환이다.



이론상으로 파장의 크기가 '1'인 $\cos(2\pi f)$ 신호를 주파수 축에서 보면 크기가 파장의 크기에 $\frac{1}{2}$배가 된 $\pm f$성분이 나오게 된다. 하지만 $\cos(2\pi \hat{f} t)$, $\hat{f} = 0.1$ 신호를 50-point DFT 변환의 결과에서는 $\hat{f} = \pm 0.1$ 에서 크기값이 25인 것을 볼 수 있다. 여기서 공부한 스케일링방법을 적용해 보려고 한다. N-point가 증가하면 결과값들의 크기는 커지지만 비율은 그대로 유지되기 때문에 X[k]를 N으로 나눠주기만 하면 된다. 이렇게 간단한 스케일링 알고리즘을 거치면, X[k]의 전체적인 스펙트럼을 보기 수월해지고, N값이 변하더라도 동일한 스펙트럼 결과물을 반환하게 된다. 그럼 간단하게 스케일링을 거친 50-point DFT 변환의 결과를 보겠다.



이론상 결과값과 동일하게 크기가 0.5인 성분을 만들어내는 것을 성공했다.
이미지 처리를 위해서는 2차 DFT변환의 스케일링 작업이 필요하다. 수식을 보면 1차 DFT 변환처럼 쉽게 스케일링을 할 수 있다.

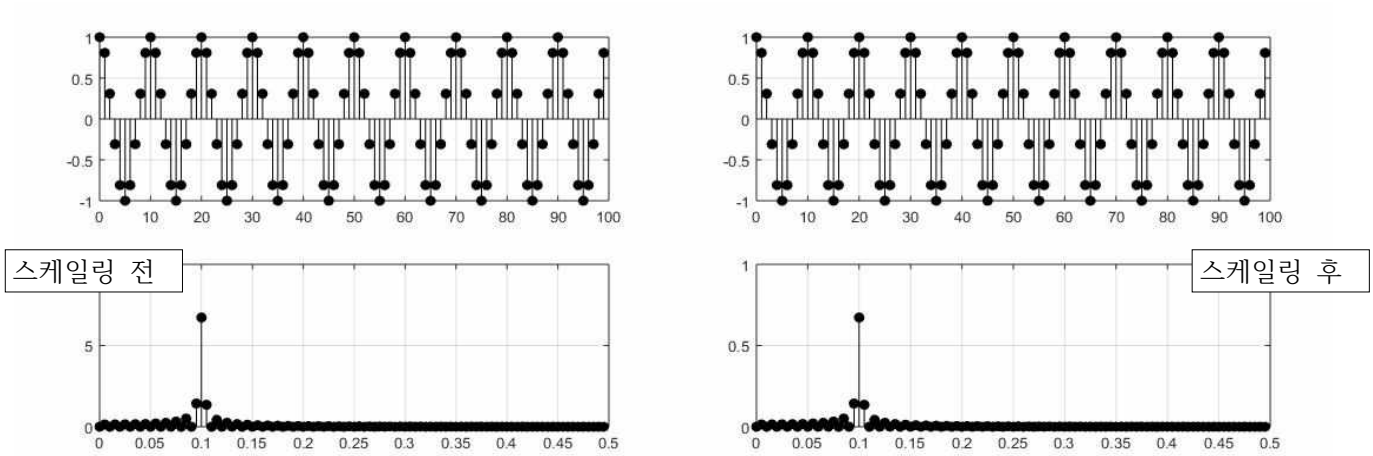
$$X(k_1,k_2)=\sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1}x(n_1,n_2)e^{-j2\pi\frac{n_1k_1}{N_1}}e^{-j2\pi\frac{n_2k_2}{N_2}}$$

수식을 보면 k1, k2 값이 변할 때마다 N^2 개의 성분이 더해진다. 이전의 결과에서 이해했던 것처럼, $X(k_1,k_2)$ 의 크기 값은 N이 증가할 때마다 크기 성분들의 비율은 유지되지만, 전체적인 크기는 N^2배 된다는 뜻이다. 스케일링 결과는 실습결과에서 확인할 수 있다.

DFT 변환의 스케일링을 이해하고 다음 DCT 변환을 수행하면서 DFT 변환의 스케일링과 동일하게 수행을 해줬다. 결과는 좋지 않았다. DCT 변환의 스케일링은 어떤 값으로 할지 확인하기 위해 수식을 살펴보려고 한다. 이번 실습(매트랩)에서 사용하는 DCT 변환은 Type-II DCT 변환이고 수식은 다음과 같다.

$$X(k)=\alpha(k)\sum_{n=0}^{N-1}x(n)\cos\left[\frac{\pi(2n+1)k}{2N}\right]\qquad \alpha(k)=\begin{cases}\sqrt{\frac{1}{N}}, & \text{if } k=0,\\ \sqrt{\frac{2}{N}}, & \text{if } 1\le k\le N-1.\end{cases}$$

DFT 변환에서 sin성분이 없어지고, 시그마 앞에 $a(k)$항이 추가된 것을 볼 수 있다. $a(k)$항이 추가되는 효과로 DCT 변환의 결과는 N이 증가할 때마다 $\sqrt{N}$ 배만큼만 커진다. DFT 변환만큼은 아니더라도 N이 증가할 때마다 값이 무수히 커지는 결과가 나오기 때문에 최종적으로 $\frac{1}{N}$배의 결과가 나오도록 스케일링 알고리즘을 구현해 보았다.



값을 충분히 줄이면서 비율을 유지하는 스케일링이 성공적으로 된 것을 볼 수 있다. 수식에 이미 $\frac{1}{\sqrt{N}}$이 곱해져있기 때문에 간단하게 $\frac{1}{\sqrt{N}}$ 로 나눠주었다. 이미지 처리를 위한 2차원 DCT 변환에서도 앞서 구현한 것처럼 쉽게 구현할 수 있다. k1에 해당하는 축과 k2에 해당하는 축을 고려하여 $\frac{1}{\sqrt{N}} \times \frac{1}{\sqrt{N}}$ 을 나눠주면 된다. 이밖에도 log를 취해 성분들의 비를 볼 수 있는 스케일링도 있지만 우리조가 구현한 방법이 demo 결과와 비슷하여 실습에서는 앞서 설명했던 방법을 사용하다.

- DCT를 활용한 간단한 압축 알고리즘의 이해

DCT 변환과 스케일링을 통해 스펙트럼 파형을 볼수 있게 되었다. DCT변환의 강점 중 하나는 압축하기에 용이한 변환이라는데, 이번 실습에서는 간단한 손실 알고리즘을 구현해보게 되었다.

손실 압축의 기본원리는 문턱값을 정해주면 스펙트럼에서 문턱값 보다 작은 값들을 0으로 바꿔주는 것이다. 처리하지 않아도 되는 픽셀(값)이 늘어나기 때문에 용량이 줄어들게 되는 이점이 있다. 설명은 간단하게 할 수 있지만, 실제 구현시 주의해야할 부분이 있다.

$$DCT(x) < Threshold$$ (DCT(x)는 이미 스케일링 됐다고 가정한다.)

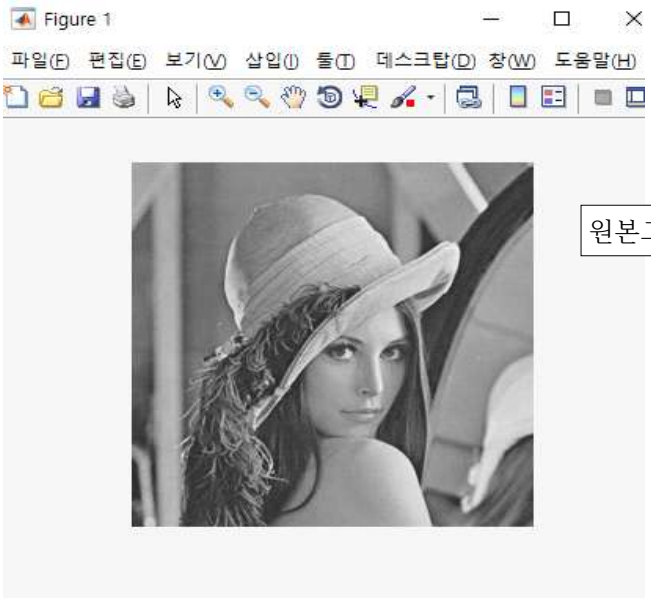
위와 같이 알고리즘을 구현한다면, 성분에서 모든 음수값이 0이 되어버리게 된다. DCT 변환 후 나오는 음수들은 상당히 중요한 정보를 가지고 있기 때문에 -threshold < DCT(x) < threshold 에 해당하지 않는 부분은 그대로 보존되어야 한다. 이러한 이유로 스케일링이 된 DCT 값에 절댓값으로 비교를 해준 뒤, 조건을 만족하는 성분만 0으로 만들어주는 압축을 구현해야 하는데 자세한 알고리즘 구현방법은 실습과정에서 설명하려고 한다.

3. 실습결과

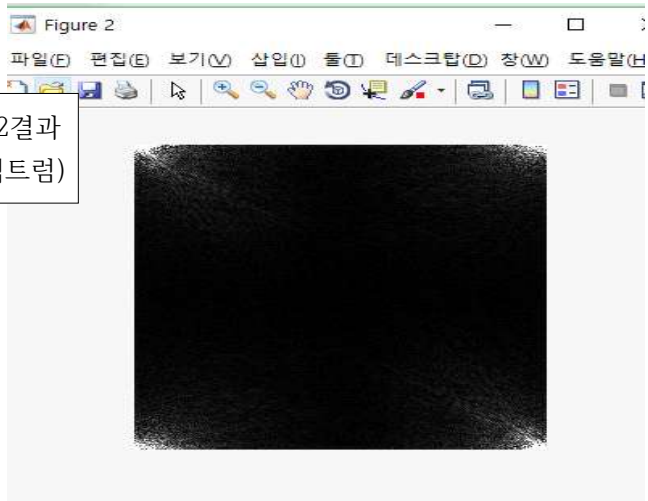
3.1 2차원 DFT (FFT) 변환을 통한 이미지 처리

- 실습 'lena.raw' 이미지의 2차원 DFT (FFT) 변환 크기값 (magnitude)을 출력하시오.
  (fft2(), myfun_LoadImage(), imshow() 함수 이용. 주의: fft 결과값이 0 ~ 255 범위를 가지도록 적절한 상수 인자로 정규화 (normalization) 해주어야 함.)
- 실습 위 2차원 DFT (FFT)된 이미지의 역변환 이미지를 출력하시오.
  (ifft2(), myfun_LoadImage(), imshow() 함수 이용)
- 실습 DEMO 위 실습 결과들을 그래프로 그리시오. (그림 3 참고)

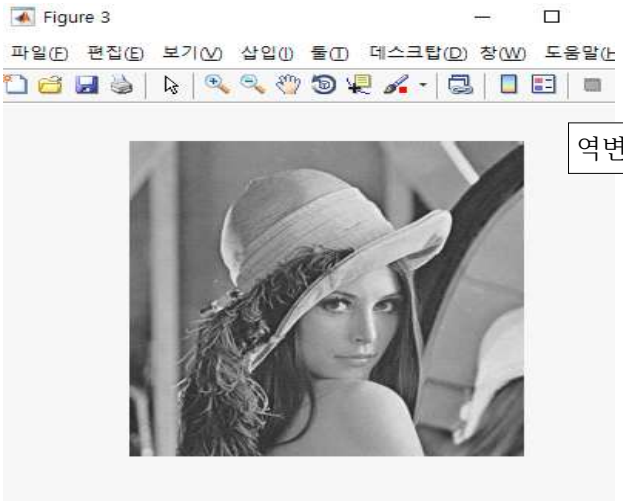
```
1 -  X = myfun_LoadImage('lena.raw', 256, 256);
2 -  figure(1);
3 -  imshow(X);
4    %실습 3.1
5 -  Z = (1/(256*256))*fft2(X);
6 -  figure(2)
7 -  imshow(abs(Z))
8
9 -  Z_re = (256*256)*ifft2(Z);
10 - Z_re = cast(Z_re,'uint8');
11 - figure(3)
12 - imshow(Z_re);
```



원본그림



FFT2결과
(스펙트럼)



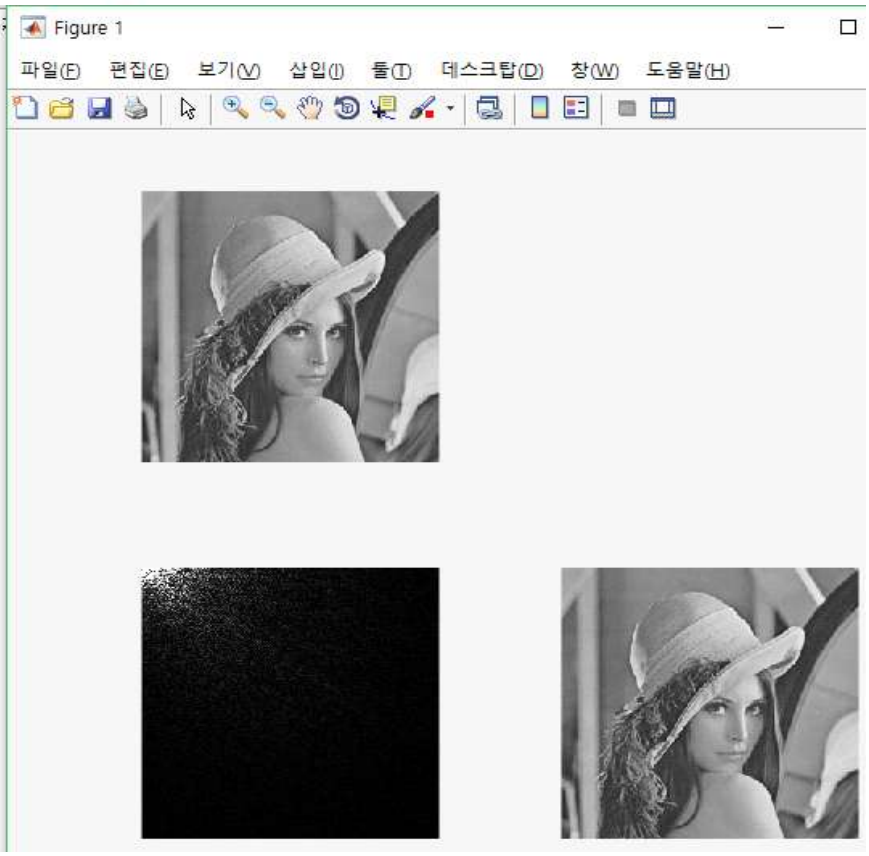
역변환

앞서 설명했던 N^2 으로 나눠주면서 FFT의 스케일링을 구현했습니다. 역변환을 구현할 때 IFFT2(FFT2(X))와같은 표현은 실습의 의미가 없다고 생각하여 스케일링한 사진을 N^2배 하면서 역 스케일링(정규화) 과정을 구현했습니다. (2번에 추가 설명 같이 했습니다.)

3.2 2차원 DCT 변환을 통한 이미지 처리

- 실습 'lena.raw' 이미지의 2차원 DCT 변환 크기값 (magnitude)을 출력하시오.
  (dct2(), myfun_LoadImage(), imshow() 함수 이용. 주의: dct 결과값이 0 ~ 255 범위를 가지도록 적절한 상수 인자로 정규화 (normalization) 해주어야 함.)
- 실습 위 2차원 DCT된 이미지의 역변환 이미지를 출력하시오.
  (dct2(), myfun_LoadImage(), imshow() 함수 이용)
- 실습 DEMO 위 실습 3.2 결과들을 그래프로 그리고, 실습 3.1 DFT 결과와 비교하여 설명하시오. (그림 4 참고)

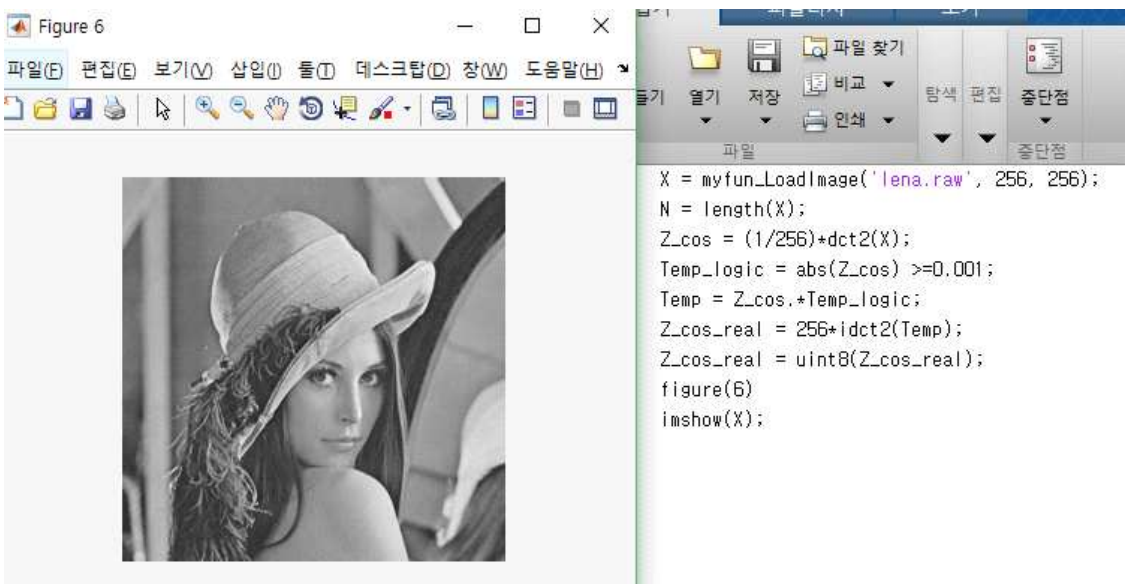
```
1 -  X = myfun_LoadImage('lena.raw', 256, 256);
2 -  figure(1);
3 -  subplot(2,2,1)
4 -  imshow(X);
5
6    %실습 3.2
7 -  Z_cos = (1/256)*dct2(X);
8 -  subplot(2,2,3)
9 -  imshow(abs(Z_cos))
10
11
12 -  Z_cos_re = 256*idct2(Z_cos);
13 -  Z_cos_re = cast(Z_cos_re,'uint8');
14 -  subplot(2,2,4)
15 -  imshow(Z_cos_re);
16
17
18
19
20
21
22
23
24
25
26
27
28
29
```



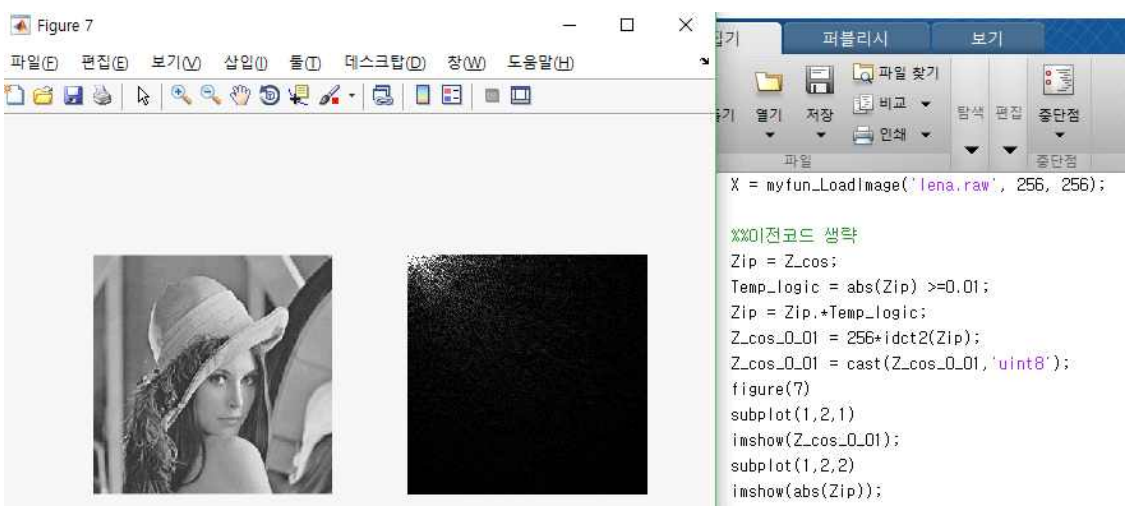
3.1번 문제처럼 idct2(dct2(X)가 아닌 스케일과 역스케일로 구현하였습니다. 3.1번과 다른점은 주파수축으로 이동하는 과정에서 정규화해주는 인자가 달라진 것을 볼 수 있습니다. 앞의 이론 정리 부분에서 언급했던 것처럼 수식의 차이 때문에 FFT2 변환에 $\frac{1}{N_1}, \frac{1}{N_2}$, DCT 변환에 $\frac{1}{\sqrt{N_1}}, \frac{1}{\sqrt{N_2}}$ 로 정규화를 하고 실습을 진행 하였습니다. 주파수 영역에서 abs()함수를 사용하여 imshow() 함수로 스펙트럼을 확인 후 역변환해준 뒤 재 정규화를 합니다. 재 정규화해주지 않으면 lena이미지의 주파수 성분들이 원래 크기 값보다 줄어든 상태에서 역변환을 실행하게 됩니다. 줄어든 주파수 성분의 영향을 받아 역변환 결과의 원소들은 크기 분포는 원본과 동일하지만 크기가 확 줄어든 흑색 화면이 나오게 됩니다. 원소들의 크기 비율은 원본과 동일하게 유지되기 때문에 각각 N_1, N_2 , $\sqrt{N_1}, \sqrt{N_2}$ 씩 곱해주면 원본과 같은 그림이 나오게 됩니다.

3.3 DCT 변환을 이용한 이미지 압축

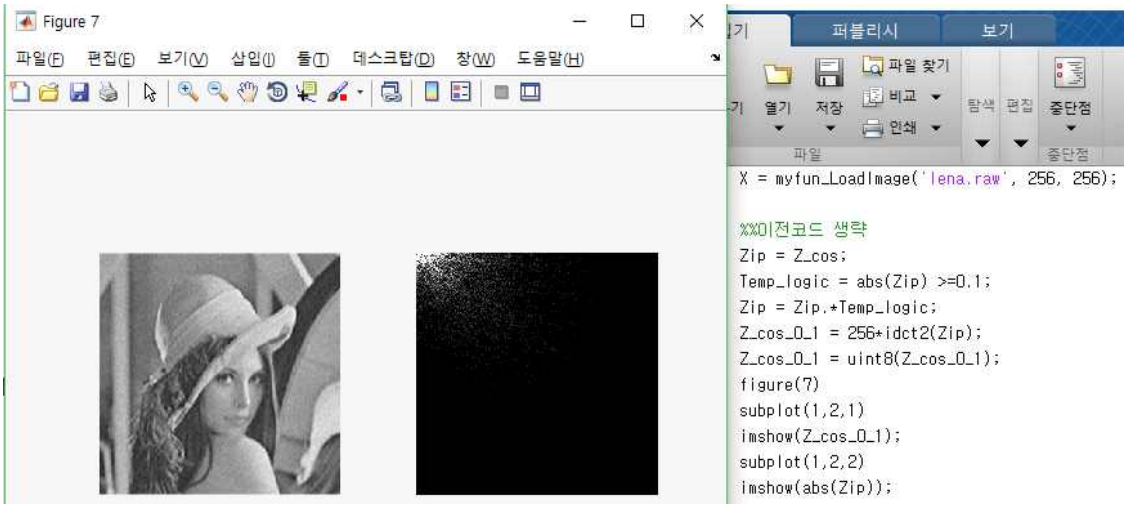
(1) 실습 DCT 변환 크기값 (0 ~ 255 으로 정규화)에 대해서 문턱값 (threshold) [0.01, 0.1, 0.25, 0.5] 보다 작은 값은 0으로 처리하는 압축을 수행하고, 압축처리된 DCT 결과값을 IDCT 역변환을 수행해서 복원한 이미지를 출력하여 문턱값에 따라 비교하시오.



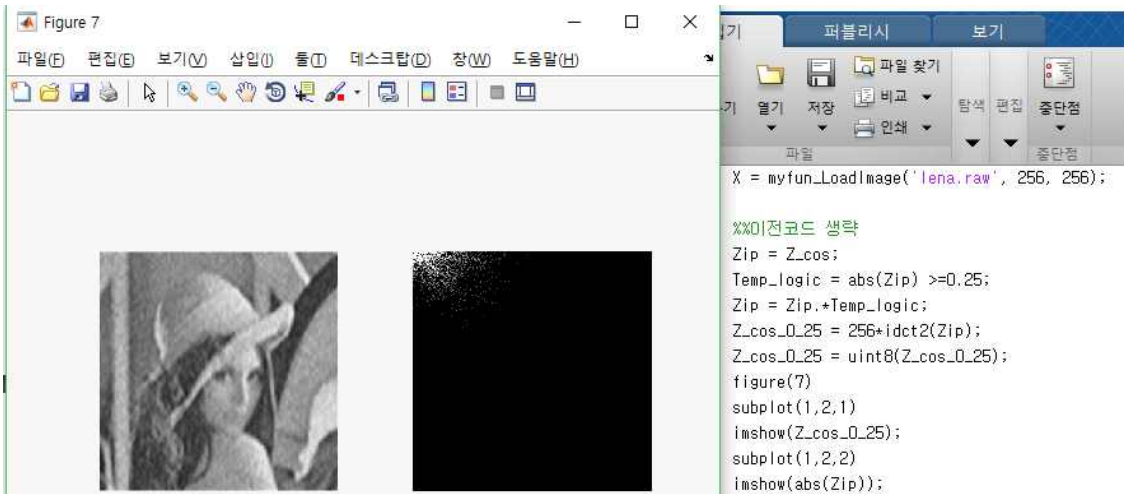
먼저 원본이미지 자체를 변수에 담아줍니다. 다음 문제에서 원본 이미지를 문턱값 0.001로 가정해주기 때문에 3.3 문제 전체에서 원본은 위의 코드를 사용합니다.



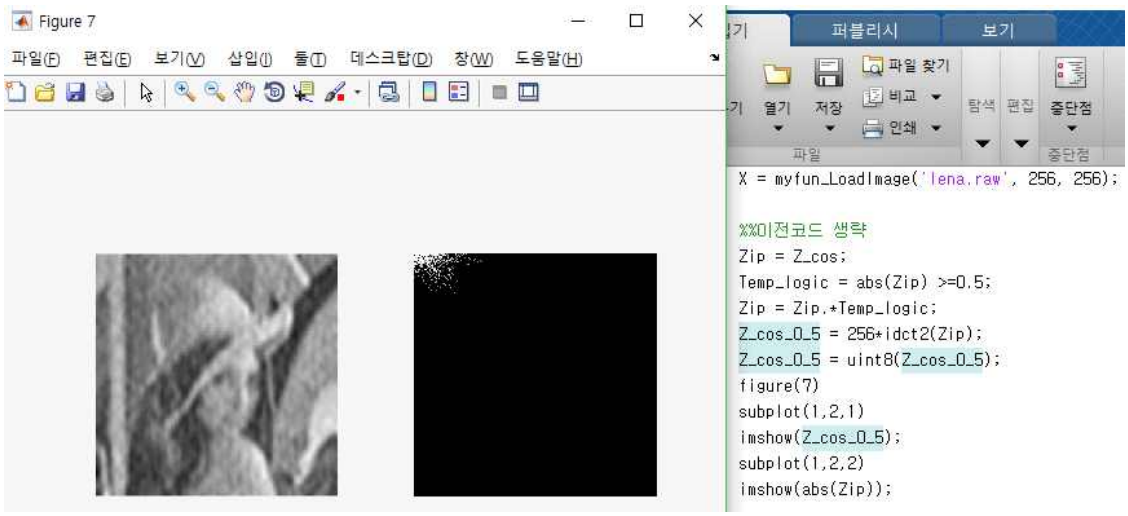
문턱값이 0.01일 때입니다. Zip이라는 변수에 이전에 만들었던 Z_cos을 넣어주고 압축을 진행합니다. 압축 알고리즘에서는 logic값들을 사용하였습니다. 먼저 Zip 변수에 abs()함수를 사용하고 크기 비교를 수행 뒤 Temp_logic이라는 변수에 담아둡니다. 변수명에서 알 수 있듯이 조건을 만족하면 해당 인덱스에 1을 만족하지 못하면 0을 반환하게 됩니다. 이후 logic값과 Zip변수를 곱해주게 되는데, 주의할 점은 비교에만 abs함수를 사용하고 크기 조건을 만족하는 음수값들을 유지하는것 입니다. 주파수 성분을 스펙트럼으로볼 때 크기 그래프로 보기 때문에 실수로 음수값 자체를 무시해 버릴수도 있는데, 그렇게 되면 사진에 필요한 중요한 주파수 성분이 없어지면서 원본 사진이 많이 손상됩니다. 이러한 주의점을 고려하고 알고리즘을 구현하여 좌측과 같은 스펙트럼과 압축 이미지를 얻었습니다. 문턱값이 0.01일때는 원본과 많이 차이가 나지 않는 것을 볼 수 있습니다.



문턱값이 0.1일 때 결과입니다. 좌측에 결과를 보면 스펙트럼은 거의 변하지않았지만 lena 사진에 아주 살짝 자글자글한 부분이 있는 것을 볼 수 있습니다. 스펙트럼이 일치할 정도로 크기가 겨우 0.1 미만인 성분만 제거했는데도 사진에 변화가 생기기 시작했습니다. 아무리 문턱값이 0.1 이여도 스케일링 전에는 생각보다 큰 주파수 성분이여서 그랬을 것 같습니다.



이어서 문턱값이 0.25일 때 결과입니다. 원본사진보다 많이 손상이 됐으며 스펙트럼에서도 눈으로 감지할 수 있을 정도의 변화가 일어났습니다.



마지막으로 문턱값이 0.5일 때 사진입니다. 압축률이 높아서 스펙트럼과 원본 이미지가 많이 변한 것을 확인할 수 있습니다. 정규화한 스펙트럼 그래프에서 0.1~0.5 정도의 작은 성분도 원본 이미지에 큰 영향을 주고 있습니다. 자연계의 대부분 신호들은 저주파수 사진이라고 하지만, 작은 성분의 고주파 성분도 상당히 중요한 비중이 있다는 것을 알 수 있는 실습이 되었습니다.

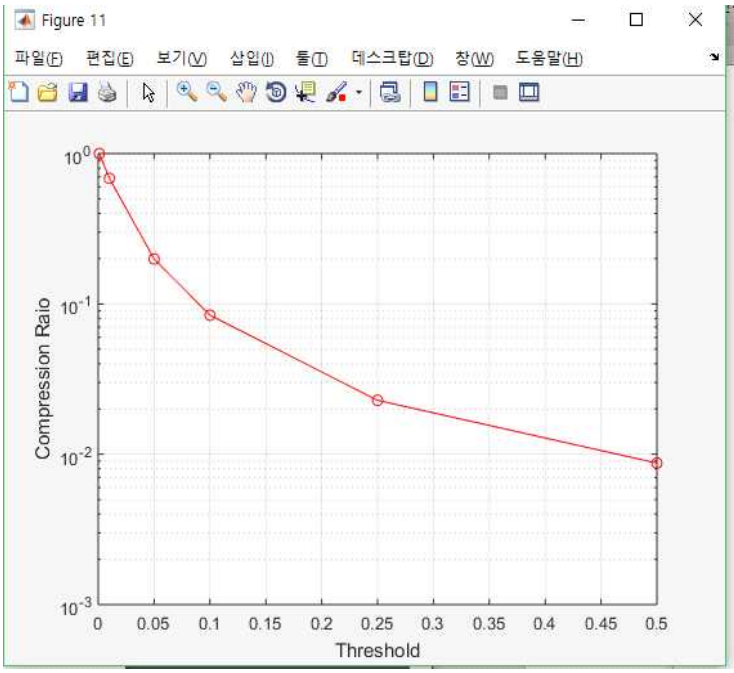
(2) 실습 2차원 DCT 결과값에 대해서 문턱값 (threshold) 대비 압축률 관계를 구하시오. (x축: 문턱값, y축: 압축률)

- 문턱값 [0.001, 0.01, 0.05, 0.1, 0.25, 0.5] 에 대해서 압축률을 구하시오.
- 압축률은 다음으로 정의하시오.

$$\text{압축률} = \frac{\text{압축 처리 후 이미지에서의 DCT 값이 0이 아닌 픽셀의 개수}}{\text{원본 이미지에서의 DCT 값이 0이 아닌 픽셀의 개수}},$$

단, 원본 이미지에서의 DCT 값은 0.001 미만이면 0으로 간주한다.

- 압축률 값은 log 스케일로 그리시오.




```
clc;
clear;
X = myfun_LoadImage('lena.raw', 256, 256);
N = 256; %lena의 col, row 값이 동일
Z_cos = (1/256)*dct2(X);
k_real = find(abs(Z_cos) < 0.001);
k = [0.001,0.01,0.05,0.1,0.25,0.5];
Zip = [];
for i = 1:6
    Temp_logic = abs(Z_cos) >= k(i);
    Z_cos = Z_cos.*Temp_logic;
    count_0 = find(Z_cos == 0);  %압축된 Temp에서 0의 개수를 찾음
    %전체 N^2에서 0이아닌 개수를 뺀다.
    %압축후의 count_0를 뺀 값에서 원본 k_real을 뺀 값을 나눠준다.
    Zip = [Zip , ((N^2)-length(count_0))/((N^2)-length(k_real))];
end
figure(11)
semilogy(k,Zip,'r-o')   %로그 그래프
grid on
xlabel('Threshold')
ylabel('Compression Raio')
```

문제에 주어진 압축률의 정의를 보면, 원본 이미지의 DCT변환 중 0이아닌 픽셀의 개수가 분모에 들어가 있기때문에, 실제 압축이 많이 될수록 오히려 결과값이 작아지게 됩니다. 문턱값을 배열로 만든 후 for문 이 시작되고, 해당 문턱값이 적용된 스펙트럼에서 0이아닌 개수를 찾게 됩니다. k_real은 원본 이미지의 스펙트럼에서 0이 아닌 픽셀의 인덱스이고 count_0은 문턱값에 따른 압축이 적용된 스펙트럼에서 0이 아닌 픽셀의 인덱스입니다. 총 픽셀 수에서 0이 아닌 픽셀 수를 빼주면서 압축률을 구현하게 되는 알고리즘입니다.

결과를 해석해보면 스펙트럼에서 문턱값이 0.001일 때 0이 아닌 픽셀의 수보다 문턱값이 0.5일 때 0이 아닌 픽셀의 수가 100배 줄어든 것을 볼 수 있습니다.

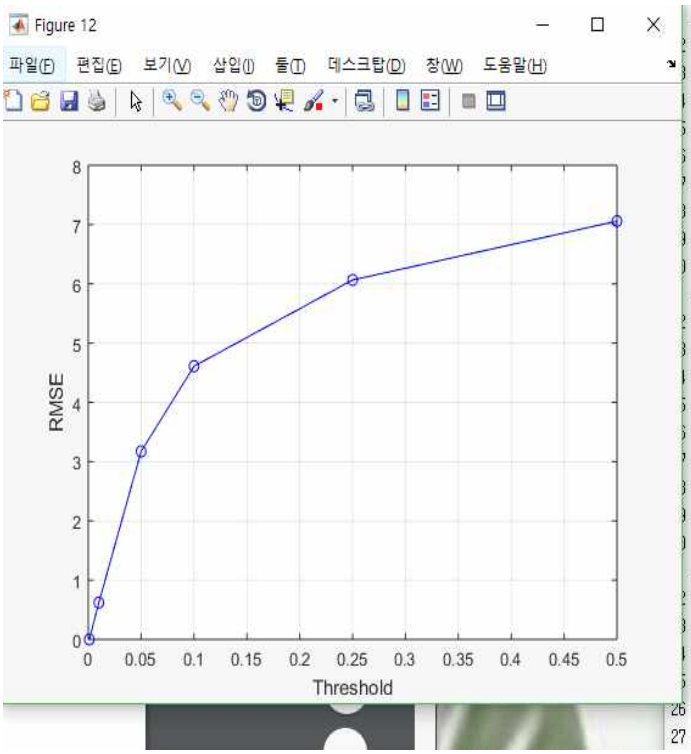
(3) 실습 2차원 DCT 결과값에 대해서 문턱값 (threshold) 대비 정확도 관계를 구하시오. (x축: 문턱값, y축: RMSE 값)

- 문턱값 [0.001, 0.01, 0.05, 0.1, 0.25, 0.5] 에 대해서 압축률을 구하시오.
- 정확도는 RMSE (root mean square error)로 나타내시오.

$$\mathrm{RMSE} = \sqrt{\frac{1}{N_1 N_2} \sum_{n_1=1}^{N_1} \sum_{n_2=1}^{N_2} |x(n_1, n_2) - \tilde{x}(n_1, n_2)|^2},$$

여기서 $x(n_1, n_2)$는 원본 이미지의 (n_1, n_2) 픽셀에 해당하 gray-scale 값 $(0 \sim 255)$, $\tilde{x}(n_1, n_2)$는 압축 복원된 이미지의 (n_1, n_2) 픽셀에 해당하는 gray-scale 값$(0 \sim 255)$이다.

(4) 실습 DEMO 위 실습 결과들을 그래프로 나타내고 문턱값 (threshold)와 압축률 및 정확도와의 관계에 대해서 논하시오. (그림 5, 6, 7, 8 참고)




```
X = myfun_LoadImage('lena.raw', 256, 256);
N = 256; %lena의 col, row 값이 동일
Z_cos = (1/256)*dct2(X);
Temp_logic = abs(Z_cos) >=0.001;
Z_cos = Z_cos.*Temp_logic;
Z_cos_real = 256*idct2(Z_cos);
Z_cos_real = uint8(Z_cos_real);
k = [0.001,0.01,0.05,0.1,0.25,0.5];
for i = 1:6
    Temp = Z_cos;
    Temp_logic = abs(Temp) >= k(i);
    Temp = Temp.*Temp_logic;
    Z_cos_team(:,:,i) = uint8(256*idct2(Temp));
end
RMSE = [];
Div_N = 1/(N^2);    %N값 고정
for i = 1:6
   Z_sub = Z_cos_real - Z_cos_team(:,:,i);
   Z_sub = Z_sub.*Z_sub;  %뺀 값을 제곱한다.
   %sum을 두번사용한 이유는 2차원이기 때문에(한번에 한 줄씩)
   RMSE = [RMSE , sqrt(Div_N*(sum(sum((Z_sub)))))]; %루트에 상수 Div_N을 곱하고 제곱
end
figure(12)
plot(k, RMSE,'b-o');
grid on
xlabel('Threshold')
ylabel('RMSE')
```

- 뒷장에 결과 해석이 있습니다.

- **(3)에 대한 설명**

RMSE 값을 구하기위해 먼저 문턱값 배열을 만들어주고 각 문턱값에 해당하는 압축 이미지를 Z_cos_team이라는 3차원 배열에 저장합니다. 압축 알고리즘은 이전 문제들에 사용하던 알고리즘과 동일합니다. RMSE 공식을 해석해보면 원본 그림과 압축된 그림에서 각 화소의 차이를 제곱평균하고 루트를 씌운 값입니다. for문 안에서 두 그림의 차이를 구하면서 RMSE 공식을 해석한 그대로 계산을 진행하는 알고리즘을 구현하였습니다. 특이한 점은 sum을 두 번 쓰게 되는데, 매트랩에서 2차원 배열에 sum함수를 쓰게 되면, 모든 열만 합쳐서 1행으로 만들어주기 때문에 한번 더 sum 함수를 사용하여 남았던 1행도 하나의 값으로 만들게 됩니다. RMSE 값이 크다는 것은 원본 그림과의 차이가 많이 난다는 뜻이고, 압축에 사용하는 문턱값이 증가할 때마다 RMSE값도 증가하는 결과를 볼 수 있습니다.

- **(4)문턱값과 압축률 및 정확도와의 관계**

문턱값 (threshold) 는 압축률에서 화소의 값이 0으로 변하게 되는 범위라고 해석할 수 있습니다. 임계값이 0.001이면 정규화한 스펙트럼에서 크기가 0.001 미만의 화소들은 모두 0이 되게 됩니다. 정확히 0의 값을 같게 되는 주파수의 범위는 예시로 든 임계값을 기준으로 -0.001 ~ 0.001 이 됩니다. 문턱값이 높다는 것은 0의 값을 갖게되는 화소가 많아진다는 것을 뜻하고, 상대적으로 작은 값인 고주파수 성분들이 점점 없어진다는 것을 의미합니다. 실습 결과를 보면 문턱값이 높을수록 스펙트럼 그래프는 우측하단에서 좌측상단 방향으로 검정색 부분이 늘어나게 됩니다. 이 말은 해당 주파수 성분의 값이 0이 됐다는 뜻입니다. 그림에서 고주파수 성분의 역할은 변화가 심한 부분에 몰려있고 선명도와 연관이 있는데, 실습 결과물에서 고주파수 성분이 점점 없어질수록 그림이 뭉개지고 선명도가 많이 떨어지는 것을 볼 수 있습니다.

압축률은 압축 처리 후 이미지에서의 DCT값이 0이 아닌 화소의 개수를 원본 이미지에서의 DCT값이 0이 아닌 화소의 개수로 나눠준 값입니다. 압축을 적게 할수록(문턱값을 낮게 잡을수록) 값을 0으로 대체하는 크기의 범위가 좁아져서 0이되는 주파수가 줄어들게 되고, 문턱값보다 크다면 아무리 크기가 작은 값을 갖는 주파수 성분도 0이 아닌 자신의 값을 갖고 있게 됩니다. 결과적으로 문턱값이 증가할수록 분자의 값이(0이 아닌 화소 수가) 줄어들기 때문에 실습에 사용된 압축률 수치는 낮아지게 됩니다.

RMSE는 정확도를 나타내는 수치이며 원본 이미지의 화소와 압축된 이미지의 화소들의 차이를 제곱평균 해주고 루트를 씌우는 방법으로 산출하게 됩니다. 원본 그림에서 원본 그림을 빼주면 0이 나오고, 오차의 평균이 0 이라는 것은 두 그림이 일치한다는 뜻입니다. 문턱값이 증가하면 RMSE값이 증가하게 되는데, 두 그림의 차이가 크기 때문에 많이 훼손됐다고 해석할 수 있습니다.

결과적으로 문턱값이 높을수록 압축은 잘 되지만(실습에서 쓰인 압축률 수치는 낮아지게 되지만), RMSE값은 커지기 때문에 원본 그림과 비교했을 때 정확도는 떨어진다고 볼 수 있습니다.

(+) DEMO에 있는 RMSE그래프는 uint8형 뺄샘으로 결과값이 나왔습니다. 압축한 사진의 화소값이 원본보다 '3' 높을 경우에 오차는 3 이라는 값이 나와야 하지만, uint8의 부호없는 성질로 '0'이라는 값을 반환하게 되어 전체적으로 RMSE값이 낮게 형성됩니다. 다행히도 두 그래프 모두 전체적으로 보면 압축을 할수록 RMSE값이 커지고, 오차가 커진다는 동일한 결과가 나왔습니다. 아래는 화소값들을 뺄셈 직전에만 double형으로 변환하여 RMSE 알고리즘을 실행한 후 비교해 본 결과입니다.