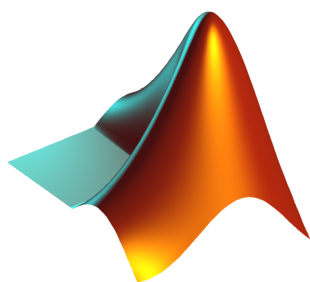


임베디드 신호처리 실습

DCT 실습 결과보고서

전자공학부 임베디드시스템

2014146004 김민섭

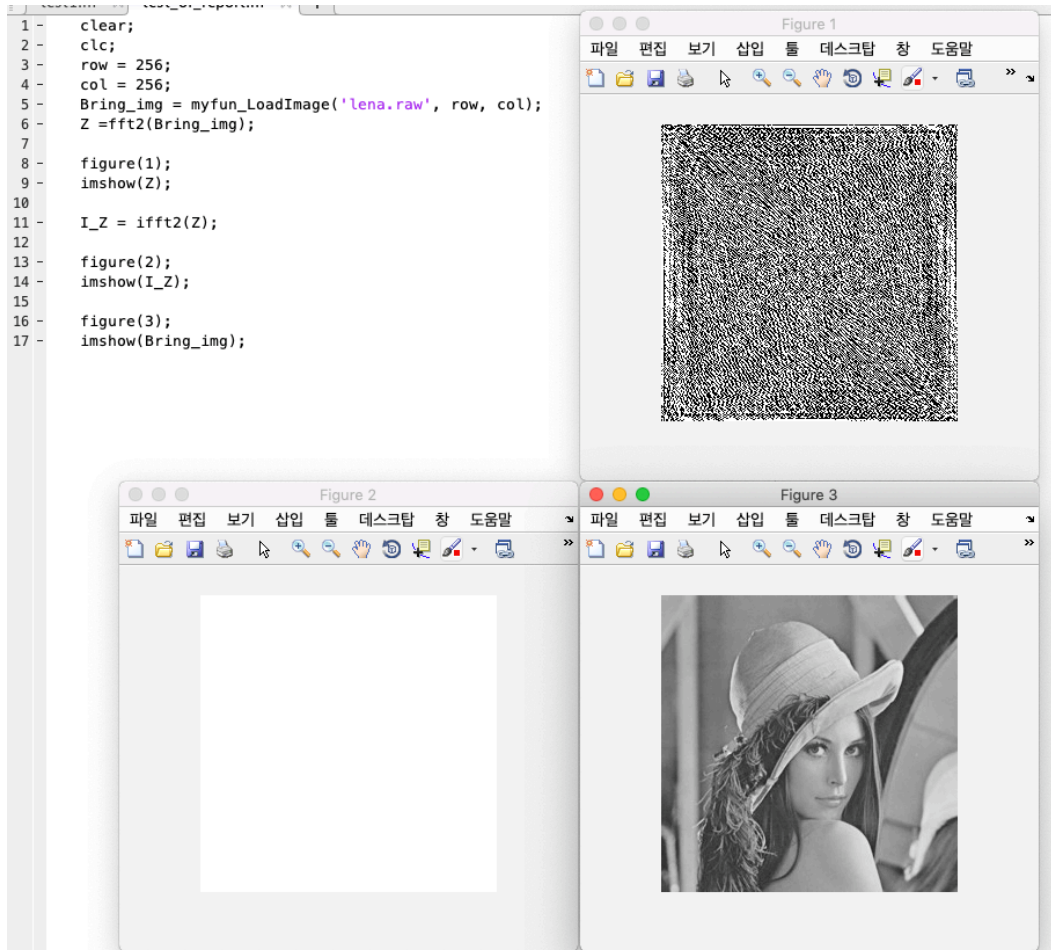


MATLAB

1. 2차원 DFT 변환을 통한 이미지 처리

이번 실습은 공간영역에 있는 이미지를 주파수 영역으로 이동시킨 이후에 다시 공간영역으로 되돌리는 실습이다. 지금까지의 실습은 1차원을 다루어 봤다면 이번에는 2차원을 다루게 되는 것이다.

본 실습에서는 MATLAB의 내장함수인 `fft2` 와 `ifft2` 를 이용할 예정이다. MATLAB 내장함수를 이용하니 이전의 실습과 달리 난이도가 쉬울 수 있지만 생각해야 할 것이 있다. 바로 정규화 과정이다. 그냥 단순히 `fft2` 와 `ifft2`를 하게 될 경우 주파수 영역과 복원되는 이미지 영역에서도 잘 보여지지 않는다. 이를 확인하기 위해 다음 그림들을 살펴보자.



위 코드와 결과값들을 보면 Figure1 는 `fft2`를 이용하여 변환한 주파수영역에서의 이미지, Figure2 는 Figure1 에 나온 값들을 `ifft2` 하여 복원한 이미지이다. 보이는 것과 같이 단순히 내장함수만을 사용해서는 주파수 영역에서 제대로 관찰 하기도 어렵고 복원도 되지 않는다. 이렇게 되는 이유는 0~255까지의 픽셀값을 표현해주는 `imshow`에 너무나 높은 값이 들어가기 때문이다. 다음은 위 식의 최대값과 최소값이다.

```
lowest =
    -1.2959 - 4.4115i

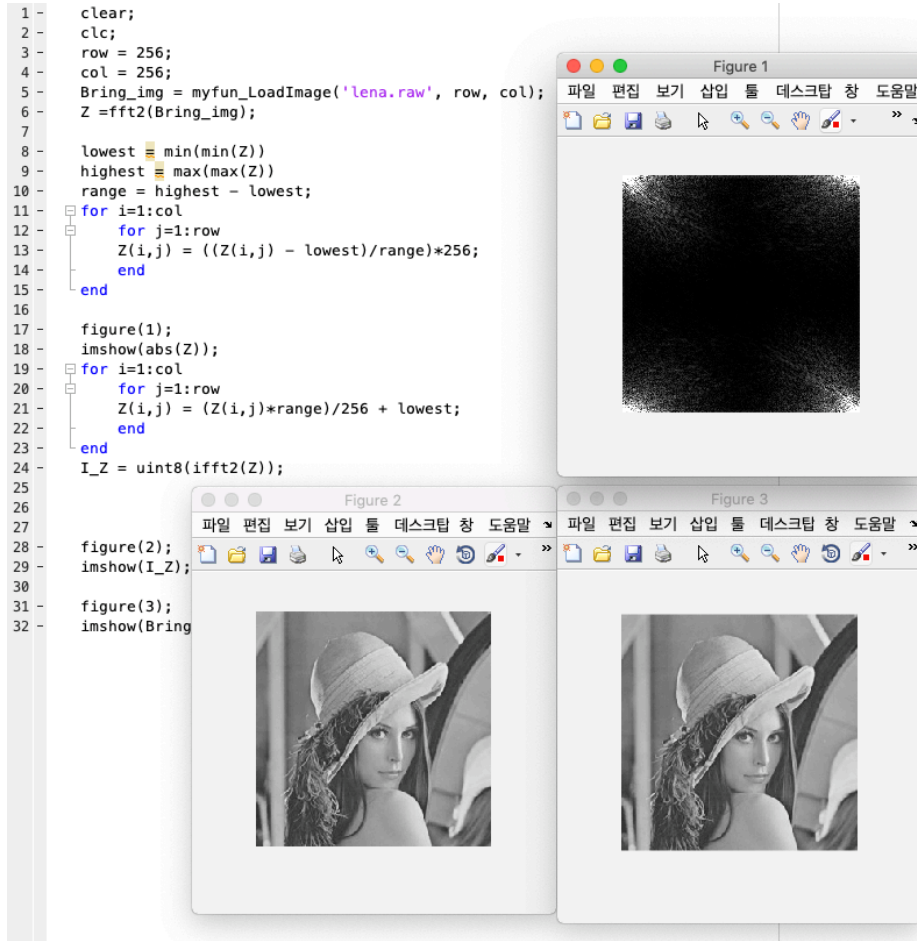
highest =
    8416162
```

결과를 살펴본 결과 최대값이 255보다 과도하게 크다는 것을 알 수 있다.

그렇다면 어떤 방식으로 정규화를 시켜주는 것이 좋을까? 여기서는 가장 간단한 정규화 방법인 최대 최소 정규화 방식을 사용하였다. 최소 최대 정규화 공식은 다음과 같다.

$$X_{new} = \frac{X - \min(X)}{\max(X) - \min(X)}$$

위 공식을 사용하면 원 식에 해당되는 값들이 0~1 사이의 값으로 나오게 된다. 이 값에다 255를 곱하여 우리가 원하는 범위 내에 있는 값을 찾을 수 있다. 구현된 코드와 데모는 다음과 같다.



위 코드와 그림을 보면 원본 사진이 제대로 주파수축으로 변환된 모습과 이를 다시 복원한 모습이 우리의 예상대로 잘 이루어진 것을 살펴볼 수

```

function [After_img] = myfun_munTuck(Befor_img, mun_Tuck)
Z=Befor_img;
k =find(abs(Z)< mun_Tuck);
Z(k) =0;
After_img=Z;

```

있다. 이처럼 어떤 신호나 이미지를 변환한 후 우리가 보기 편한 방식으로 관찰하고자 할때 그의 기준에 따른 정규화가 필요하다. 이 실습과정을 통하여 이미지를 주파수 영역에서 관찰할 수 있었다. 하지만 FFT를 통한 변환이라 실수부와 허수부가 존재하여 많은 양의 데이터를 처리하기에는 쉽지않아 보인다. 이를 좀 더 간략하게 하기 위한 DCT실습을 다음으로 진행해 보도록 하겠다.

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

실습 2를 통해서 DCT 변환 크기값을 잘 구할 수 있었다. 우리가 DCT 실습을 통해서 알고자 하는 것은 이미지 압축의 원리이다. 이번 실습에서는 이미지 압축의 원리 중 하나인 주파수축에서의 압축을 진행하는 것이라 생각된다. 문턱값 보다 작은 값들을 0으로 만들어 줌으로 인해서 0에 가까운 값들을 0으로 만들어 주는 것이다. 그리고 0으로 만들어준 값에 대해서는 저장하지 않음으로서 이미지의 용량을 줄이는 압축을 진행 할 수 있게 되는 것이다. 이미지가 처리되기전에 어떻게 이미지를 처리하는지 코드를 통해서 다시한번 살펴보도록 하자. 코드는 다음과 같다.

위의 코드처럼 문턱값보다 작은 Z값들의 인덱스 값을 찾은 이후에 이 값들을 0으로 만들어 버리면 된다. 굉장히 간단하다 위의 코드가 이해가 되지 않는다면 for문으로도 위와 같이 구현이 가능하다.

```
for i=1:col
    for j=1:row
        if abs(Z(i,j)) < mun_Tuck
            Z(i,j) = 0;
        end
    end
end
```

이해하기는 더 쉬워졌을지는 몰라도 코드가 다소 길어졌으며 물론 실행시간도 차이가 발생한다. 다음은 두 코드의 실행시간이다.

myfun_munTuck	6	0.004 s	0.004 s	
myfun_munTuck	6	0.060 s	0.060 s	■

위에 있는 문턱값이 find 문을 사용한 것이고 아래에 있는 것이 for 문을 사용한 것이다. 이처럼 for문이 다른 문법보다 실행시간이 더 걸린다는 것을 확인 할 수도 있었다. 코드에 대해 이해를 했으니 결과값을 예상해 보자. 당연한 예상이겠지만 문턱값이 커질수록 복원될 이미지는 원본보다 좋지 못할 것이라 예상된다. 이 예상이 맞는지 결과를 보면서 확인해 보도록 하자.

2. 2차원 DCT 변환을 통한 이미지 처리

문제 2에서는 DCT변환을 사용하여 이미지 변환을 한다. DCT변환이란 기존에 사용하던 FFT랑 DFT 변환과 비슷한 푸리에 관련 변환 이지만 크게 다른점은 DFT 변환의 결과값은 복소수인 점에 반해 DCT 변환의 결과값은 실수형이란 점이다.

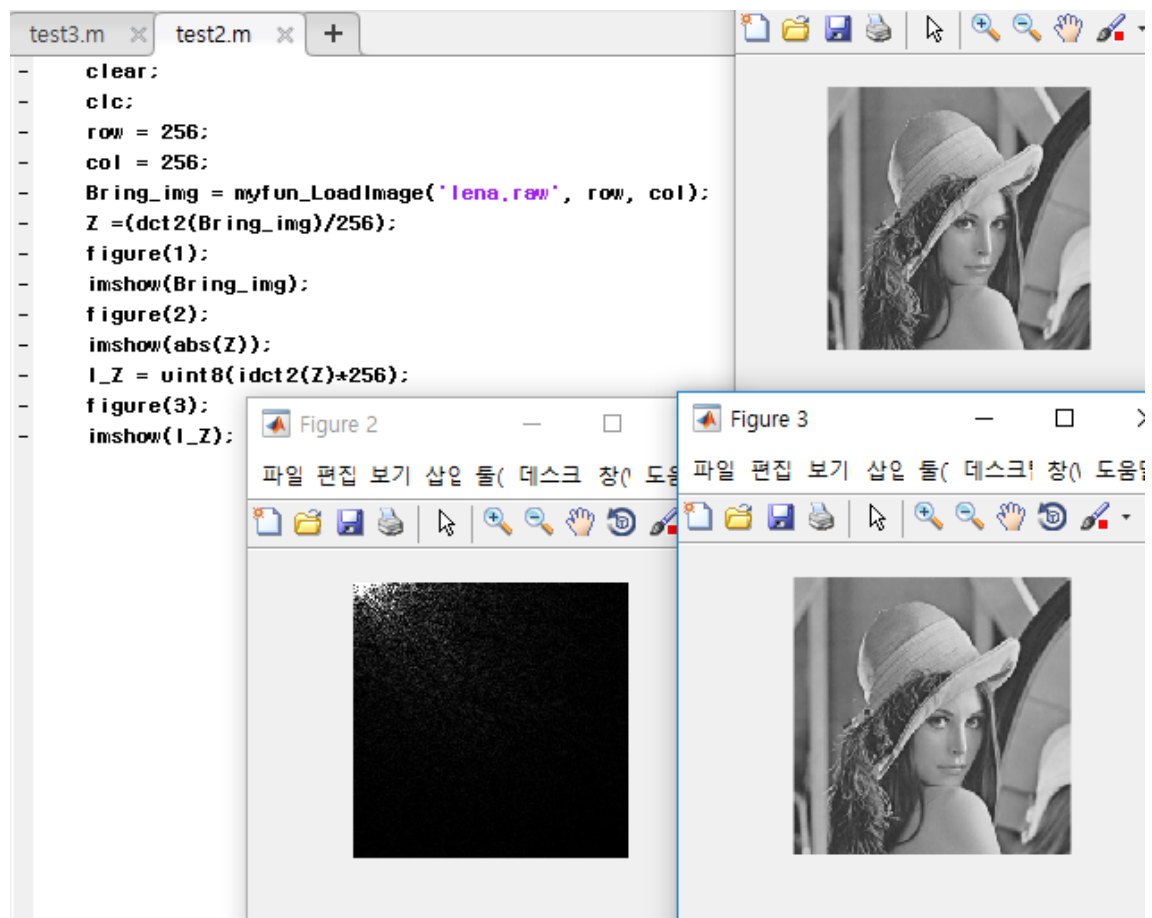
DCT변환에 대한 수식은 다음과 같이 나타난다.

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

수식상에서 기존 DFT와 다른점 이라면 $x(n)$ 에 곱해지는 항이 DFT의 경우 $e^{-j2\pi \frac{k}{N}n}$ 으로

오일러의 공식에 의하면 $\cos(2\pi \frac{k}{N}n) - j \cdot \sin(2\pi \frac{k}{N}n)$ 이 된다.

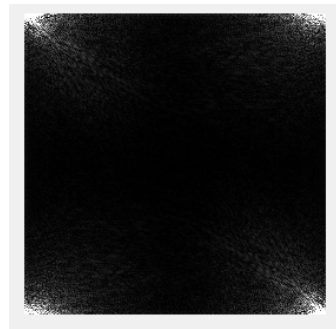
여기서 복소수 부분이 생기게 된다. 그에 반해 DCT변환의 경우 위 수식을 보게 되면 $x(n)$ 에 곱해지는 항이 \cos 함수 뿐이라서 결과값이 실수형으로 나오게 되는 것이다. 복소수에 비해 비교적 계산이 쉬운 실수형이 영상처리에서 주로 쓰인다고 한다. 다음은 DCT를 사용하여 변환한 이미지 결과이다.



우측 첫 번째 그림 figure(1)은 오리지널 이미지이고 figure(2)는 오리지널 이미지를 MATLAB 내부 함수 `dct2`를 사용한 2차원 DCT 변환을 한 그림이다. figure(2)는 시간영역의 이미지를 2차원 DCT변환을 하여 주파수영역으로 변환시킨 결과인데 대체적으로 검은색배경에 왼쪽 윗 부분만 흰색의 빛을 띤다. 여기서 검은색 부분의 오른쪽 아래가 고주파 영역, 흰색 부분이 저주파 영역이다. 더 나은 비교를 위해 아래의 오른쪽 DFT변환 시 그림을 보게되면, DFT 변환의 결과값은 복소수 값의 위상 때문에 네방향 모두 반복적으로 흰색부분의 저주파 영역이 나온다. 왼쪽의 DCT의 경우 DFT와는 달리 복소수 부분이 없기 때문에 한쪽 방향에만 저주파 영역이 나타난다.

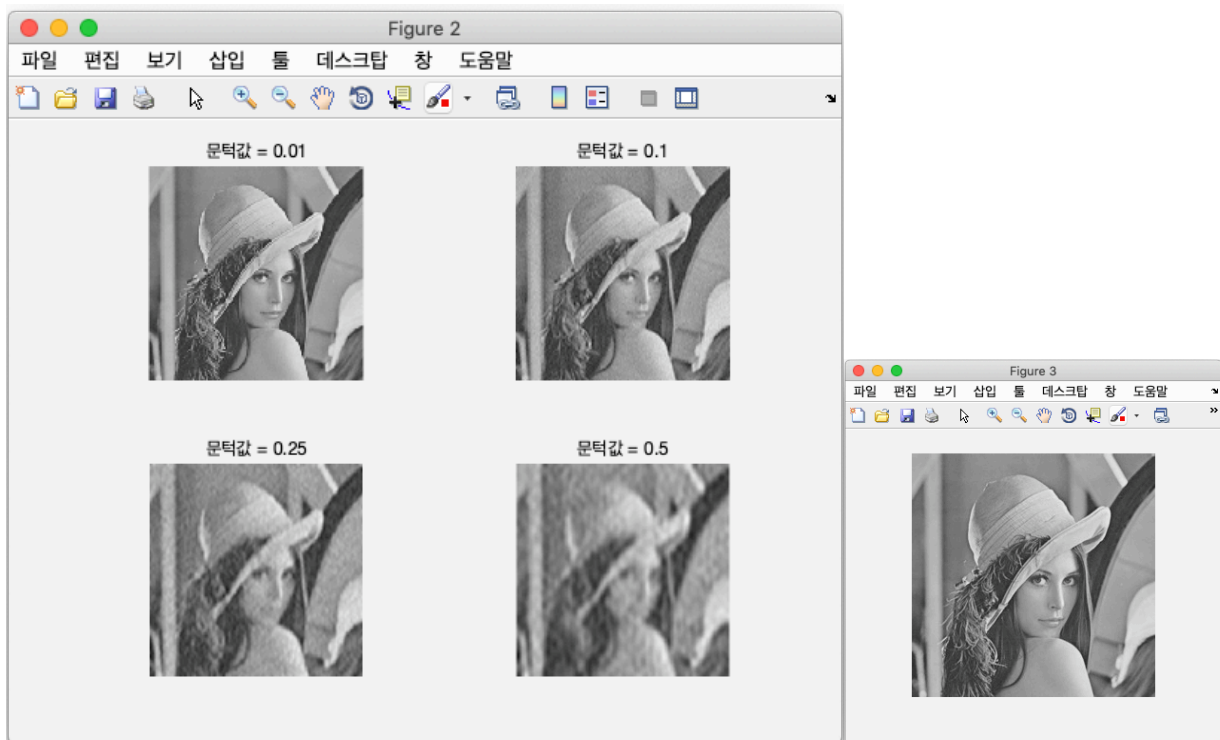


DCT변환 결과



DFT변환 결과

마지막 figure(3)은 한번 DCT변환 한 그림을 역DCT변환 한 그림이다. 원래 DCT변환을 한 후 압축하여 저장후 다시 복원하는 방식으로 JPEG를 사용하였는데 압축시 문턱값을 사용하여 문턱값 이하의 값들을 다 0으로 만들어 주어 압축을 한다. 하지만 지금 문제는 문턱값을 고려하지 않으므로 DCT변환 후 다시한번 역변환을 해주면 원본 이미지와 동일한 결과를 얻을 수 있다.



좌측에 있는 창이 문턱값을 적용시켜 이미지를 처리한 것이며 우측에 있는 것이 원본이다. 우리의 예상대로 문턱값이 커질수록 이미지 화질의 저하가 일어나는 것을 확인할 수 있다. 이러한 결과가 나오는 이유는 간단하다. 문턱값이 커질수록 더 많은 데이터를 0으로 만드는 것이니 말이다.

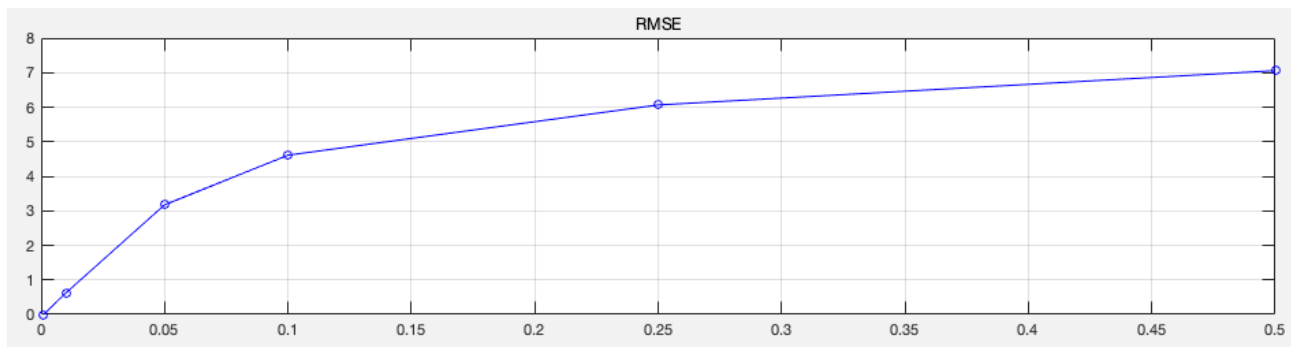
3.3 2차원 DCT 결과값에 대해서 문턱값 대비 정확도 관계를 구하시오 (x축 : 문턱값, y축 : RMSE 값)

- 문턱값 [0,001, 0,01, 0,05, 0,1, 0,25, 0,5] 에 대해서 RMSE를 구하시오.

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

위 실습을 진행하기에 앞에 RMSE 가 무엇인지 간략히 알아보고자 한다. RMSE의 한국어 번역은 ‘평균 제곱근 오차’이며 표준 편차를 일반화시킨 척도로서 실제값과 추정값의 차이가 얼마인지 알려주는데 사용된다고 한다. (출처 : 네이버 지식백과) 본 실습에서는 문턱값을 기준으로 이미지를 처리한 것과 원본 이미지 간의 차이를 알아보는 과정이라고 생각해 볼 수 있다.

위 실습도 마찬가지로 예측을 진행할 수 있다. 문턱값이 높아지면 그만큼 데이터를 많이 잃어버리게 될 확률이 높으므로 RMSE 값은 증가할 것이라 예상된다. 그럼 RMSE 결과를 살펴보도록 하자.



결과는 우리가 실험을 실시하기 전에 예측했던 것과 동일했다. 문턱값이 높아질수록 RMSE 값이 증가하는 모습을 살펴볼 수 있었다. RMSE를 구현한 코드는 다음과 같다.

```
function [RMSE_Value] = myfun_RMSE(First_img, Second_img, col,row)
RMSE_mid =[];
RMSE_mid =abs(First_img - Second_img).^2;
RMSE_MAX = sum(sum(RMSE_mid));
RMSE_Value =sqrt((1/(row*col))*RMSE_MAX);
```

이번 DCT 실습을 통해서 주파수 영역에서 이미지를 관찰하는 것과 주파수 영역에서의 이미지 압축하는 기본적인 원리에 대해서 살펴볼 수 있었다. 이미지 압축을 실시할적에 무리하게 문턱값을 높여서 압축을 진행하기 보다는 적절한 문턱값을 설정하여야 원본 이미지와의 차이를 크게 두지 않으면서 이미지 데이터를 줄일 수 있다는 것을 알게되었다.




PS. 실습을 통해 알게된 추가적인 사항

- for 문의 실행속도가 굉장히 좋지 않는다.

1번 실습 보고서에도 언급이 되었지만 for 문보다 다른 MATLAB 문법이 훨씬 더 빠르고 간편하다는 것을 알 수 있었다. 다음은 본 실습에서 진행되었던 문턱값, RMSE, 압축률에 대해서 for문을 사용했을 때와 다른 문법을 사용했을 때의 실행속도 비교 모습이다.

myfun_RMSE	6	0.001 s	0.001 s	
myfun_Rates_of_DCT	6	0.004 s	0.004 s	
myfun_munTuck	6	0.005 s	0.005 s	

< for 문을 사용하지 않은 실행시간 >

myfun_RMSE	6	0.609 s	0.609 s	
myfun_Rates_of_DCT	6	0.099 s	0.099 s	
myfun_munTuck	6	0.060 s	0.060 s	

< for 문을 사용한 실행시간 >

위 실행시간을 보면 확실히 for문을 사용하지 않은 곳에서 실행시간이 매우 짧아진 것을 볼 수 있다. 이를 통해 앞으로 MATLAB 코드를 작성할 시 특별한 상황이 아니라면 for문 외에 다른 알고리즘을 사용하는 습관이 들여야 겠다는 생각이 들었다.