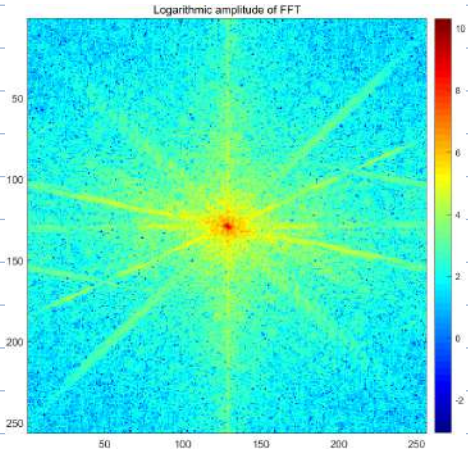
EXPERIMENT 1S

Demonstrator: Michalis Lazarou

Part 1: Image transformation

1. Fast Fourier Transform

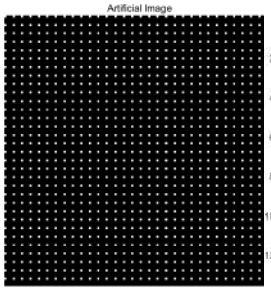


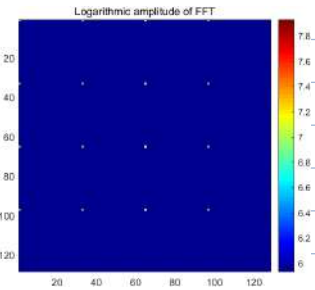
The original image and its log magnitude of FFT are shown on the left.

In the second image, the low frequency components are shifted to the middle.

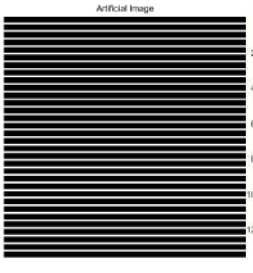
The low frequency components are the low index values

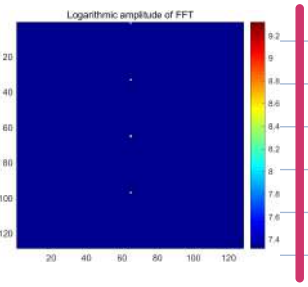
The low index values refer to the image areas varying slowly, and the natural images are smooth. Therefore, for the natural image, the low index values (low frequency components) usually processes higher values.

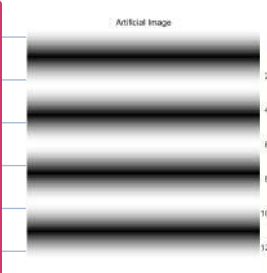


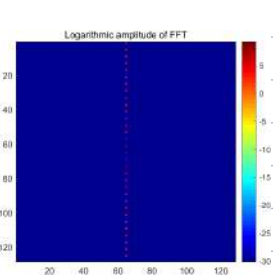


For the image of a grid of impluses shown on the left, its DFT is also impluse-like, which is because of the sampling theory.

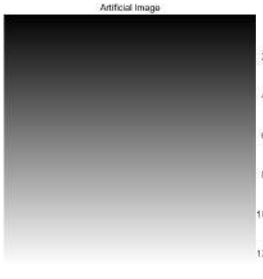


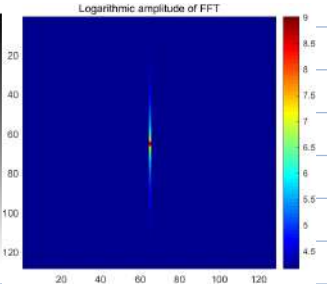






For the periodic signals, their DFTs are discrete. The reason is that the periodic signals can be written in the form $f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty}(a_n \cos nx + b_n \sin nx)$ which has discrete frequency.



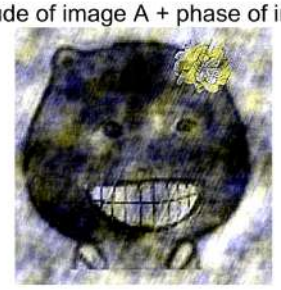


While for the non-periodic signal, their DFTs are continous. The reason is that the non-periodic signal can only be written in the form $f(x) = \int_0^{\infty}[A(\lambda)\cos\lambda x + B(\lambda)\sin\lambda x]\,d\lambda$, which has continous frequency.

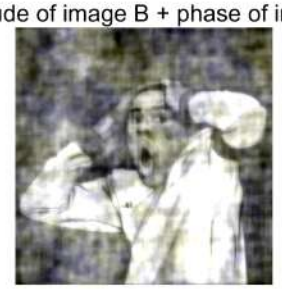
image A

image B

amplitude of image A + phase of image B

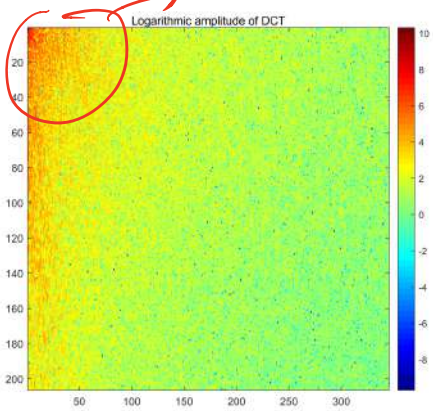


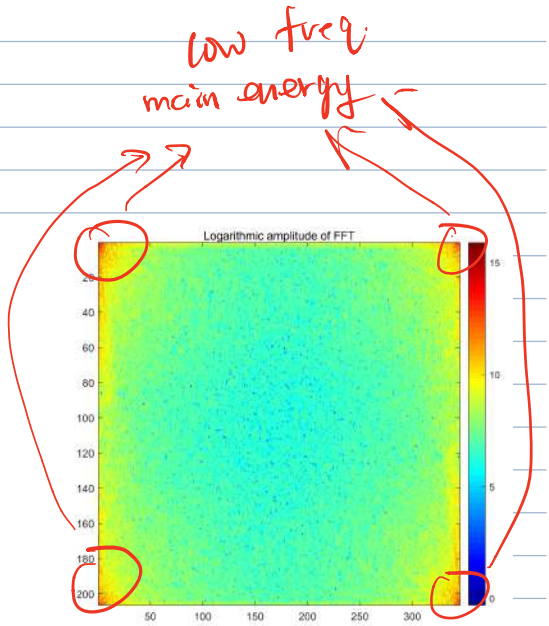
amplitude of image B + phase of image A



The phase and amplitude of two images are swapped. The two resulting images indicate that the image contents can be recognized with the phase of image. The reason is that the phase of image determine the locations of frequency within the image, which means that the edges (high frequency components) are preserved by the phase. Therefore, we can recognize the image content based on the edges in the image.

2. Discrete Cosine Transform

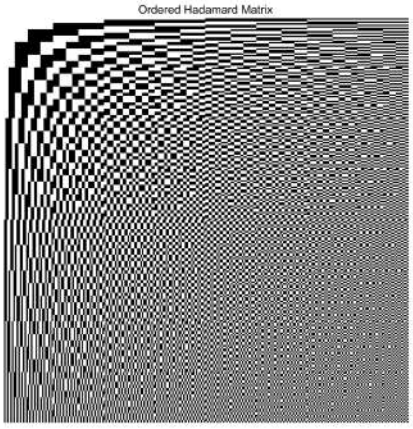




The DCT and FFT of the image are shown on the right. The low frequency components of FFT are NOT shifted to the middle, therefore the low frequency components are located at the four corners.

In DCT, the low frequency components are located at the top-left corner, so the main energy is concentrated at the top-left corner.

3. Hadamard Transform



The ordered Hadamard Matrix and the Hadamard Transform of Image are shown above. The figure of Hardamard Transform indicates that the main energy is located at the top-left corner.

Advantages of Hadamard Transform

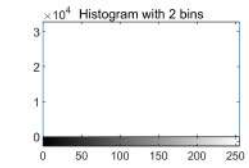
1°. The large fraction of energy is packed within very few transform coefficients, which are located at the top-left corner. Therefore, the image compression can be performed by keeping these few coefficient and setting the rest to zero.

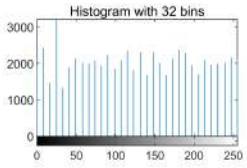
2° The large Hadamard matrix can be calculated from small Hadamard Matrix

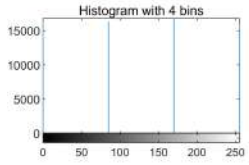
3° The Hadamard matrix only consists of 1s and -1s, which is resistant to errors

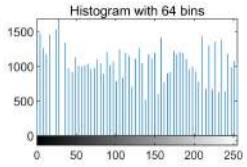
Part I: Image Enhancement

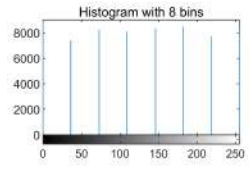
1. Histogram Equalisation

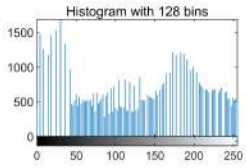


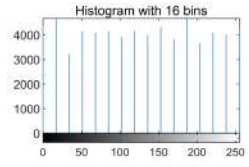


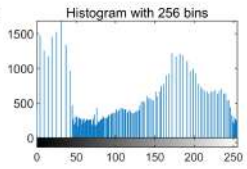








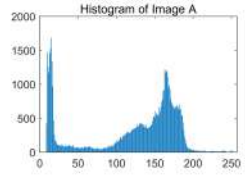


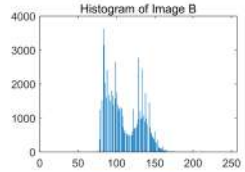


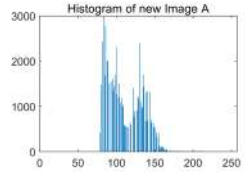
The histograms with a wide range of bins are used to equalise the image.

For the histogram with k bins, there are k gray levels in the equalised image. For the histograms with smaller number of bins, the equalised images have less gray level, while they have the higher contrast. All this results give the enhanced images.

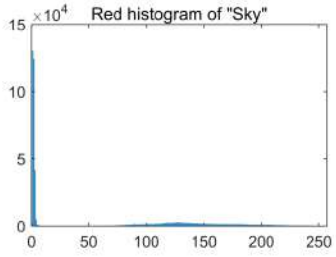
2. Histogram Modification

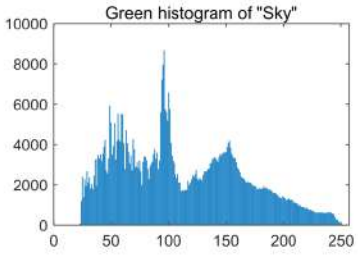


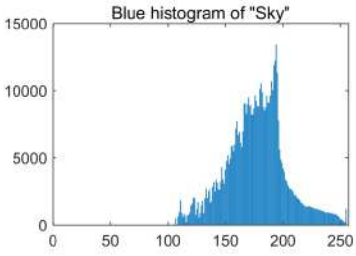


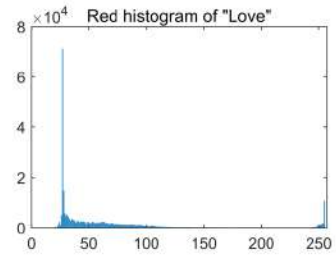


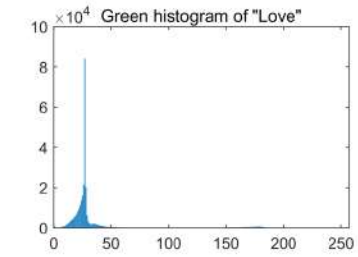
The histogram of image A is modified to the histogram of image B. The result shows that the modified image has the similar colour tone as the image B as they have almost the same histogram.

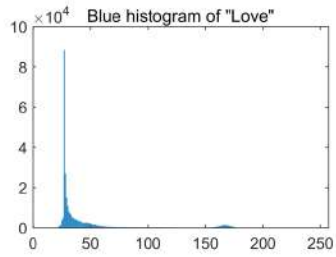


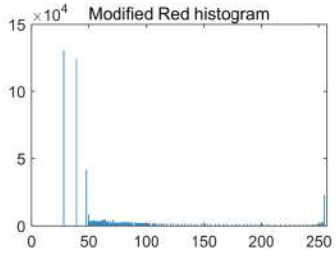


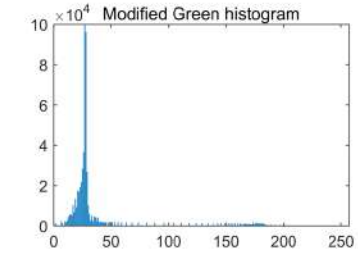


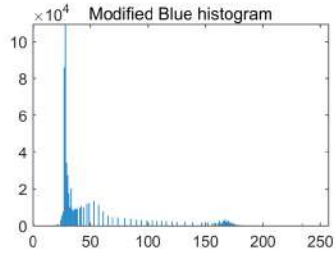






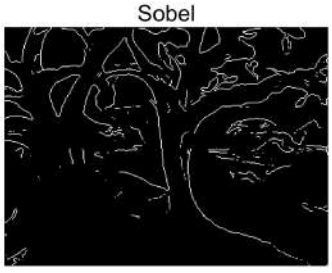


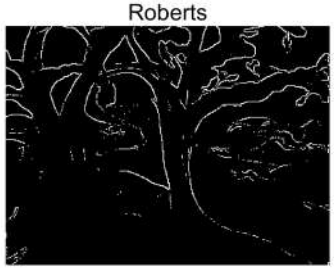


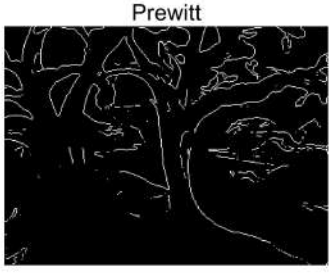


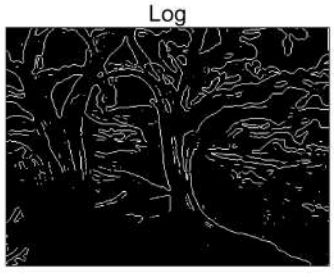
The histogram modification can also be applied on coloured image, such as the example above. The three colour channel (R,G,B) are modified repectively, and we can obtained a image with the similiar colour tone as the image whose histograms are used. In this example, we get the pink sky.

3. Edge Detection









The figures above shows the results of edge detection using four different spatial maskes. (Sobel, Roberts, Prewitt and Log)

The Sobel and Prewitt has the similiar results, because they have the similiar structure, The difference between Sobel and Perwitt mask is merely the value in the middle

$$\begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix} \quad \begin{pmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}$$

Prewitt masks

$$\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \quad \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

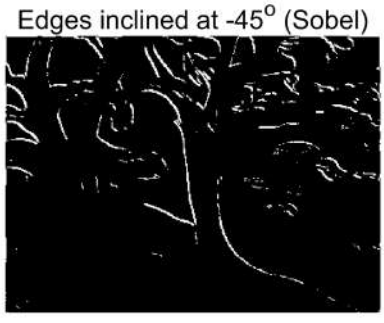
Sobel masks

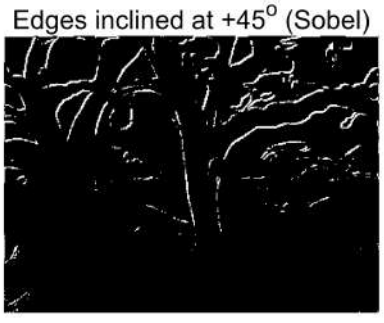
The result of Robert is a little worse than the results of Sobel and Prewitt

$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & 0 & 0 \end{pmatrix}$$

Robert masks

The result of Log catches up more details than all other results, where there are more edges.





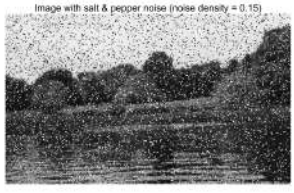
The Edges inclined at -45° are detected using a Sobel mask

$$\begin{pmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{pmatrix}$$

The Edges inclined at +45° are also detected using a Sobel mask

$$\begin{pmatrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \end{pmatrix}$$

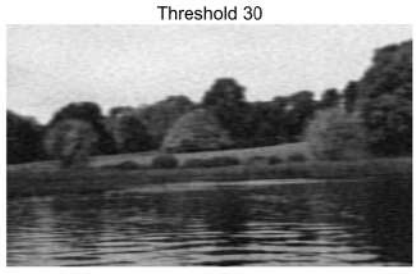
4. Median Filter

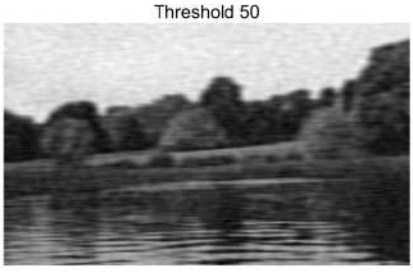


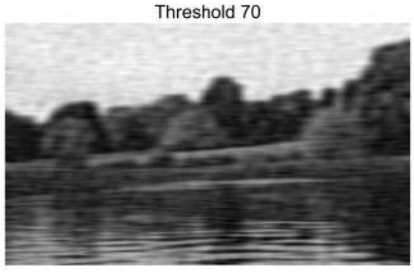
For the median filter with larger neighbourhood dimensions, the filtered image is more blurred. The reason is that the principal function of median filter is to focus pixels to be like their neighbours.

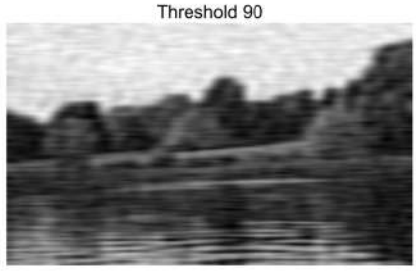
In the above result, we can notice that median filter is powerful to remove the salt & pepper noise. For the salt & pepper with higher noise density, the median filter with higher neighbourhood dimensions should be applied.

Part III Image Compression





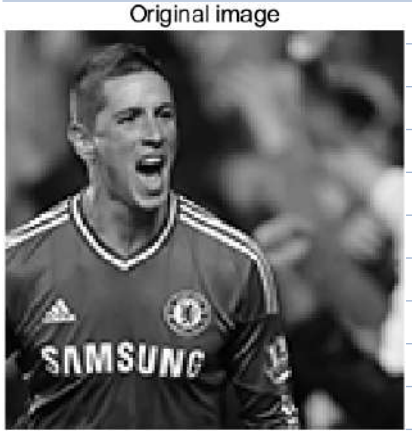




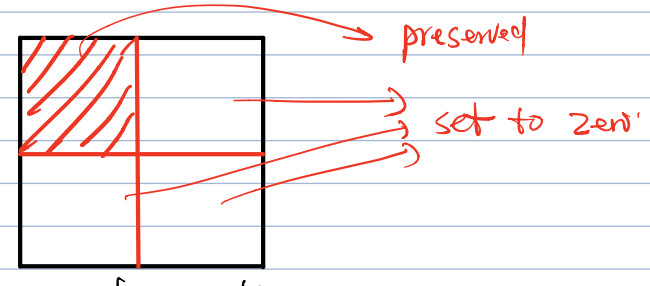
By setting the DCT coefficients that are smaller than a threshold to zero, the above results are generated. These results indicates that the higher thresholds lead to the more blurred result images. According to the properities of DCT coefficient, the low frequency components usually have high coefficient values, while the high frequency components usually have the low coefficient values. Therefore, this operation will set the high frequency components to zero. Higher the threshold is, more high freq components are removed. Thus, more fine details are removed from the original image. leading to the more blurred image. Although the image is blurred, we can still recognise the image content, which lead to the usage of this operation in compression.

The 5 different thresholds are used in the 5 small blocks in the image on the left. Although there are varying degrees of blur in the 5 small blocks, it is not easy to perceive them if we do not look at this image seriously, especially from a far place. This indicates that our human vision systems are not sensitive to the fine details of the image, therefore some fine details (high frequency components) can be removed in image compression.

Part IV. Design Exercise.



Assume the 128 x 128 image on the left is transmitted. To achieve the compression, the image is divided into several N x N small blocks, and the DCT is performed on each small block. As the compression of 4:1 is required, only a quarter of DCT coefficients of each small block is transmitted. According to the property of DCT that the most energy is packed within the low frequency components, the top-left part of DCT coefficients is transmitted.



DCT of small blocks

For noiseless case, varying sizes of small blocks are tried.

These results indicates that there are blocky effect when the block size is small. Without considering the computational complexity, the larger the block size is, the better the received image is. However, the computational complexity cannot be ignored when transmitting a sequence of images. Therefore, the blocks of 8x8 or 16x16 are usually used. In this experiment, the blocks of 16x16 are used.

For noisy case, the gaussian noise is assumed. The results are shown below

Part V: Image Restoration

1. Inverse Filtering

The model of degradation in frequency domain is

$$Y(u,v) = H(u,v) \cdot F(u,v) + N(u,v)$$

The inverse filtering is that

$$F(u,v) = \frac{Y(u,v)}{H(u,v)}$$

This technique works well in noise free case, but not in noisy case.

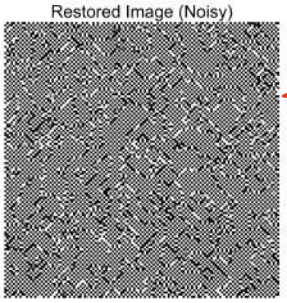
For the Noise case, the pseudo inverse filtering was proposed.

$$F(u,v) = \begin{cases} \frac{Y(u,v)}{H(u,v)} & H(u,v) \geq \text{threshold} \\ 0 & H(u,v) < \text{threshold} \end{cases}$$

Both noiseless and noisy cases are tested in this experiment.

The Gaussian degradation is assumed, and the noise is assumed to be additive Gaussian noise with BSNR = 20 dB

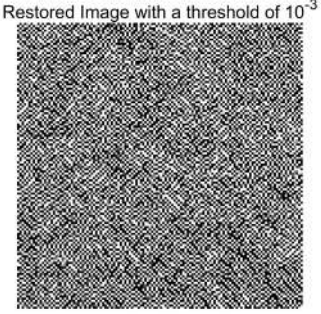
The degraded images with and without noise are shown above. The inverse filtering is applied on both degraded images.



The results on the left indicate that the degraded image without noise can be well restored by inverse filtering, while the restored image in noisy case is noise-like because of the noise amplification effect of inverse filtering.

To solve the noise amplification problem, the Pseudo inverse filtering can be applied.

Restored Image with a threshold of 10^{-3}



Restored Image with a threshold of 10^{-1}

For the noisy case, two different thresholds are tried and the results are shown on the left. When the threshold is 10^{-3}, the restored image is still noise-like. When the threshold is increased to 10^{-1}, a good restored image is obtained.

The results of 7x7 Gaussian blur is shown below

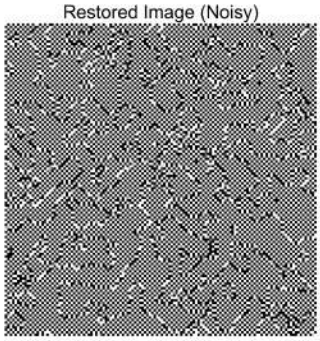
Original Image

Degraded Image (Noiseless)
7x7 Gaussian blur (σ^2=1)

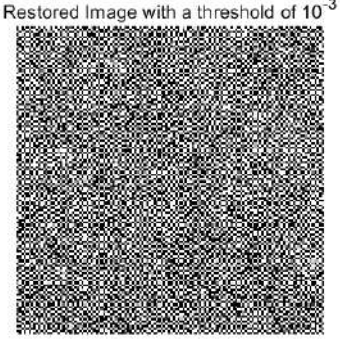
Degraded Image (Noisy)
7x7 Gaussian blur (σ^2=1) | BSNR=20dB

Restored Image (Noiseless)

Restored Image (Noisy)



Restored Image with a threshold of 10^{-3}



Restored Image with a threshold of 10^{-1}

Noisy case
pseudoinverse filter

2. Wiener Filtering

$$W(u,v) = \frac{S_{ff}(u,v) \cdot H^*(u,v)}{S_{ff}(u,v)|H(u,v)|^2 + S_{nn}(u,v)}$$

$$\hat{F}(u,v) = W(u,v) \cdot Y(u,v)$$

Applying the wiener filter on the degraded image with noise.

The results in 5x5 and 7x7 Gaussian blur cases are shown on the left.

It is obvious that the wiener filter has the better performance than pseudo inverse filter.

Main MATLAB Code.

Part 1

DCT & FFT

```
function plotfft(img, shift)
if size(img,3) == 3
    img = rgb2gray(img);
end
img_fft = fft2(img);
if shift == 1
    img_fft = fftshift(img_fft);
end
colormap(jet(64)), imagesc(log(abs(img_fft))), colorbar
axis square;
title('Logarithmic amplitude of FFT');
end
```

```
function plotdct(img)
if size(img,3) == 3
    img = rgb2gray(img);
end
img_dct = dct2(img);
colormap(jet(64)), imagesc(log(abs(img_dct))), colorbar
axis square;
title('Logarithmic amplitude of DCT');
end
```

Ordered Hadamard Transform

```
function plothadamard(img)

if size(img,3) == 3
    img = rgb2gray(img);
end

h1=size(img,1);
h2=size(img,2);

% Get ordered Hadamard matrix
H1 = orderedHadamard(h1);
H2 = orderedHadamard(h2);

% ordered Hadamard Transform
J=H1*img*H2/sqrt(h1*h2);
subplot(1,2,1);
imshow(im2uint8(H1)); title('Ordered Hadamard Matrix');
subplot(1,2,2);
imshow(im2uint8(J)); title('Hadamard Transform of Image');

end
```

```
function orderedH = orderedHadamard(size)
H=hadamard(size);
kernal = [0,0,0;0,1,-1;0,0,0];
zeroCorssing = abs(conv2(H,kernal,'same'));
zeroCorssing(zeroCorssing~=2) = 0;
[~,I] = sort(sum(zeroCorssing,2)/2); % get the order
orderedH = H(I,:); % order the image
end
```

Swap phase and amplitude of FFT

```
function [imC1, imC2] = swapAmPh(imA, imB)
    if size(imA, 3) == 1
        % for gray image
        [xA, yA] = size(imA);
        [xB, yB] = size(imB);
        x = max([xA xB]);
        y = max([yA yB]);
        imA_fft = fft2(imA);
        imA_ampl = zeros(x, y); % amplitude of image A
        imA_ampl(1:xA, 1:yA) = abs(imA_fft);
        imA_pha = zeros(x, y); % phase of image A
        imA_pha(1:xA, 1:yA) = angle(imA_fft);

        imB_fft = fft2(imB);
        imB_ampl = zeros(x, y); % amplitude of image B
        imB_ampl(1:xB, 1:yB) = abs(imB_fft);
        imB_pha = zeros(x, y); % phase of image B
        imB_pha(1:xB, 1:yB) = angle(imB_fft);

        % Swap the amplitude and phase
        imC1 = ifft2(imA_ampl .* exp(1i*imB_pha));
        imC2 = ifft2(imB_ampl .* exp(1i*imA_pha));

    else
        % for coloured image
        [imC1R, imC2R] = swapAmPh(imA(:, :, 1), imB(:, :, 1));
        [imC1G, imC2G] = swapAmPh(imA(:, :, 1), imB(:, :, 2));
        [imC1B, imC2B] = swapAmPh(imA(:, :, 1), imB(:, :, 3));
        imC1 = cat(3, imC1R, imC1G, imC1B);
        imC2 = cat(3, imC2R, imC2G, imC2B);
    end
end
```

Part II

Histogram Modification

```
function [imag_modeling,histogram] = histogram_model(target_histogram_normal,imag)
% modeling the histogram of the image to the target histogram
h = my_histogram(imag);
s = size(imag);
num = s(1)*s(2);
imag_modeling = uint8(zeros(s(1),s(2)));
h_normal = h/num; % normalize the histogram

%calculate the distribution function
target_histogram_normal_df = zeros(1,256);
h_df = zeros(1,256);
target_histogram_normal_df(1) = target_histogram_normal(1);
h_df(1) = h_normal(1);
for i=2:256
    target_histogram_normal_df(i) = target_histogram_normal(i)+target_histogram_normal_df(i-1);
    h_df(i) = h_normal(i)+h_df(i-1);
end

%calculate the transformation
map = zeros(1,256);
for i=1:256
    h = h_df(i);
    h_a_b_difference = abs(target_histogram_normal_df - h);

    %find the minimum difference
    a = find(h_a_b_difference == min(h_a_b_difference));
    map(i) = a(1);
end

%transform the image
for i=1:s(1)
    for j=1:s(2)
        imag_modeling(i,j) = map(imag(i,j)+1);
    end
end
histogram = my_histogram(imag_modeling);
end
```

Edge Detection

```
function I_edge = my_edge(I, kernal)
% get the edge of the image based on the kernal
scale = 4;
I_edge = abs(conv2(I, kernal, 'same'));
cutoff = scale * sum(I_edge(:),'double') / numel(I_edge);
thresh = sqrt(cutoff);
I_edge(I_edge > thresh) = 1;
I_edge(I_edge <= thresh) = 0;

end
```

Part IV

```
clc
clear all
close all

im = im2gray(imread('torres.jpg'));
im = im2double(im);
figure(),
imshow(im),title('Original image');

N = 16;
s = size(im,1);
numBlock = s / N;
K = 4; % compression ratio

% divide the image into small blocks
blocks = zeros(N,N,numBlock*numBlock);
for i=1:numBlock
    for j = 1:numBlock
        b = im(N*(i-1)+1:N*i, N*(j-1)+1:N*j);
        index = numBlock*(i-1)+j;
        blocks(:,:,index) = b;
    end
end

%% Noiseless
% transmitter and channel
block_dct = zeros(size(blocks));
for i=1:numBlock
    for j = 1:numBlock
        index = numBlock*(i-1)+j;
        b = blocks(:,:,index);
        b_dct = dct2(b);
        block_dct(1:N/2,1:N/2,index) = b_dct(1:N/2,1:N/2); % no noise
    end
end

% receiver
im_receive = zeros(size(im));
for i=1:numBlock
    for j = 1:numBlock
        index = numBlock*(i-1)+j;
        b_dct = block_dct(:,:,index);
        b = idct2(b_dct);
        im_receive(N*(i-1)+1:N*i, N*(j-1)+1:N*j) = b;
    end
end

figure();
imshow(im_receive);title({'Reconstructed image (noiseless)'; ['Block Size = ' num2str(N) 'x' num2str(N)]});

%% Noisy
singa_2 = 0.0001;
% transmitter and channel
block_dct = zeros(size(blocks));
for i=1:numBlock
    for j = 1:numBlock
        index = numBlock*(i-1)+j;
        b = blocks(:,:,index);
        b_dct = dct2(b);
        n = sqrt(singa_2) * randn(size(b_dct));
        block_dct(1:N/2,1:N/2,index) = b_dct(1:N/2,1:N/2);
        block_dct(:,:,index) = block_dct(:,:,index)+n;
    end
end

% receiver
im_receive = zeros(size(im));
for i=1:numBlock
    for j = 1:numBlock
        index = numBlock*(i-1)+j;
        b_dct = block_dct(:,:,index);
        b = idct2(b_dct);
        im_receive(N*(i-1)+1:N*i, N*(j-1)+1:N*j) = b;
    end
end
figure();
imshow(im_receive);
title({['Reconstructed image (noisy \sigma^2=' num2str(singa_2) ')']; ['Block Size = ' num2str(N) 'x' num2str(N)]});
```

Part V.

```m
clc
clear all
close all

im = im2double(imread('torres.jpg'));
im = im2gray(im);
[width,height] = size(im);
figure();
subplot(1,3,1);
imshow(im); title('Original Image');

% Distortion Matrix (Noise Blur)
N = 5;
H=fspecial('gaussian',[7 7],1); % Gaussian Filter
degrade_im = imfilter(im, H, 'conv', 'circular'); % circular convolution
subplot(1,3,2);
imshow(degrade_im); title(['Degraded Image (Noiseless)'; '7x7 Gaussian blur (\sigma^2=1)']);

BSNR_dB = 20;
BSNR = 10^(BSNR_dB/10);
degrade_im_mean = mean(mean(degrade_im));
sigma_2 = sum(sum((degrade_im - degrade_im_mean).^2)) / (width*height) / BSNR;
n = sqrt(sigma_2) * randn(width,height);
degrade_im_noise = degrade_im + n;
subplot(1,3,3);
imshow(degrade_im_noise); title(['Degraded Image (Noisy)'; '7x7 Gaussian blur (\sigma^2=1) | BSNR=20dB']);


%% Inverse Filtering
restored_im = inverseFiltering(degrade_im, H);
restored_im_noise = inverseFiltering(degrade_im_noise, H);

figure();
subplot(1,2,1);imshow(restored_im); title('Restored Image (Noiseless)');
subplot(1,2,2);imshow(restored_im_noise); title('Restored Image (Noisy)');

% Pseudoinverse Filtering
threshold = 1E-3;
restored_im_noise = pseudoinverseFiltering(degrade_im_noise, H, threshold);
figure();
subplot(1,2,1);imshow(restored_im_noise); title('Restored Image with a threshold of 10^-3');

threshold = 1E-1;
restored_im_noise = pseudoinverseFiltering(degrade_im_noise, H, threshold);
subplot(1,2,2);imshow(restored_im_noise); title('Restored Image with a threshold of 10^-1');

%% Wiener Filtering
restored_im = wienerFiltering(degrade_im, H, sigma_2);
figure();
imshow(restored_im); title(['Restored Image by Wiener Filter'; '7x7 Gaussian blur (\sigma^2=1) | BSNR=20dB']);
```

Inverse Filter

```m
function restored_im = inverseFiltering(degrade_im,H)
[width,height] = size(degrade_im);
N = size(H,1);

H_padding = zeros(width,height);
H_padding(1:N,1:N) = H; % padding the distortion matrix with zeros
H_padding_fft = fft2(H_padding);
degrade_im_fft = fft2(degrade_im);

% Inverse Filtering
restored_im_fft = degrade_im_fft ./ H_padding_fft;
restored_im = ifft2(restored_im_fft);
end
```

Pseudoinverse Filter

```m
function restored_im = pseudoinverseFiltering(degrade_im,H,threshold)
[width,height] = size(degrade_im);
N = size(H,1);

H_padding = zeros(width,height);
H_padding(1:N,1:N) = H; % padding the distortion matrix with zeros
H_padding_fft = fft2(H_padding);
degrade_im_fft = fft2(degrade_im);

% Pseudoinverse Filtering
restored_im_fft = degrade_im_fft ./ H_padding_fft;
restored_im_fft(abs(H_padding_fft) < threshold) = 0; % Thresholding
restored_im = ifft2(restored_im_fft);
end
```

Wiener Filter

```m
function restored_im = wienerFiltering(degrade_im,H,noise_pow)
[width,height] = size(degrade_im);
N = size(H,1);

H_padding = zeros(width,height);
H_padding(1:N,1:N) = H; % padding the distortion matrix with zeros
H_padding_fft = fft2(H_padding);
degrade_im_fft = fft2(degrade_im);

pow_im = abs(degrade_im_fft).^2; % Power Spectrum of degrade image

W = (pow_im .* conj(H_padding_fft)) ./ (pow_im .* abs(H_padding_fft).^2 + noise_pow); % Wiener Filter
restored_im_fft = W .* degrade_im_fft;
restored_im = ifft2(restored_im_fft);
end
```