Xi'an Jiaotong-Liverpool University
西交利物浦大学

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING

EEE330 Lab Report 3

In Partial Fulfillment
of the Requirements for the Degree
Bachelor of Engineering

Student Name : Lingxuan Kong
Student ID : 1405867

Contents

1 Task1

1.1 Plot the vector $x = [1 : 256]$ in Matlab

```
x=0:1:255;%generate a vector x = [1:256]
t=0:255;
subplot(3,3,1),plot(t,x);%plot vector x
xlabel('Time (s)');
ylabel('Amplitude');
title('Input sequence')
```

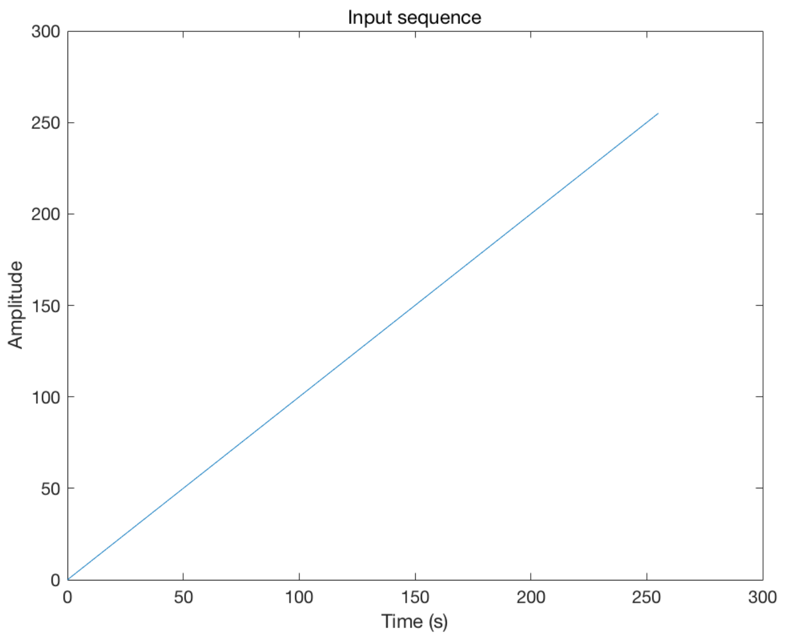


Figure 1: The graph of vector x

1.2 DCT and DFT of x

After we have the value and graph of vector x, we can apply discrete Fourier transform and discrete cosine transform of x, the code and the results will be shown:

```
%DFT
y_dft=fft(x);%Apply Fast Fourier transform to x
figure,stem(0:N-1,abs(y_dft));
xlabel('Frequency');
ylabel('|X(k)|');
title('DFT');
```

```
%DCT
y_dct=dct(x);%Apply Discrete cosine transform to x
figure,stem(abs(y_dct));%plot the result
xlabel('Frequency');
ylabel('|X(k)|');
title('DCT');
```

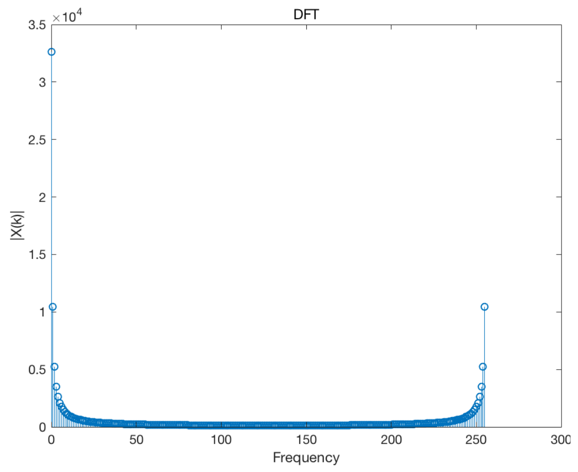


Figure 2: Discrete Fourier Transform of x

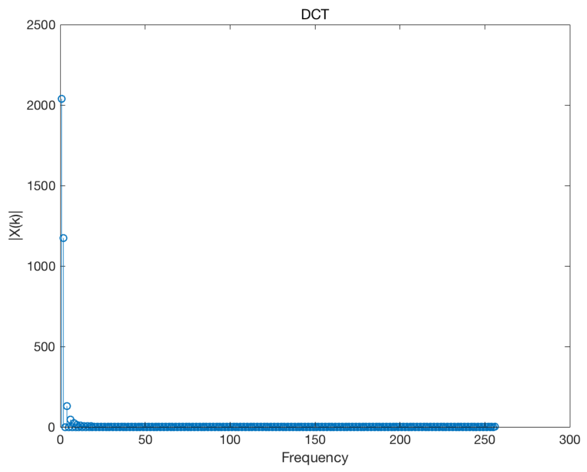


Figure 3: Discrete cosine transform of x

1.3 Display the reconstructed line

According to the previous figure2 and figure3, we could remove the high frequency component from the results of DCT and DFT:

(1) $\frac{32}{256}$

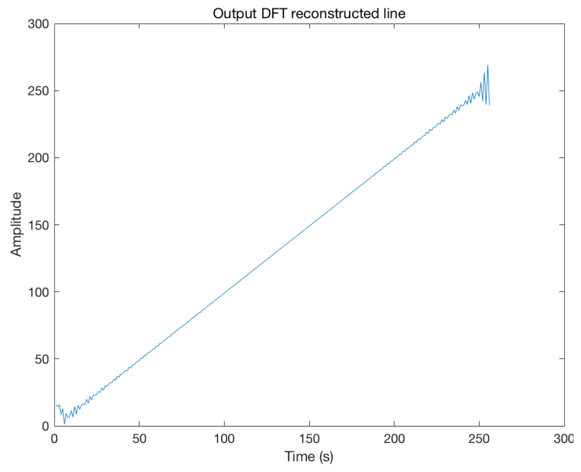


Figure 4: Output DFT reconstructed line

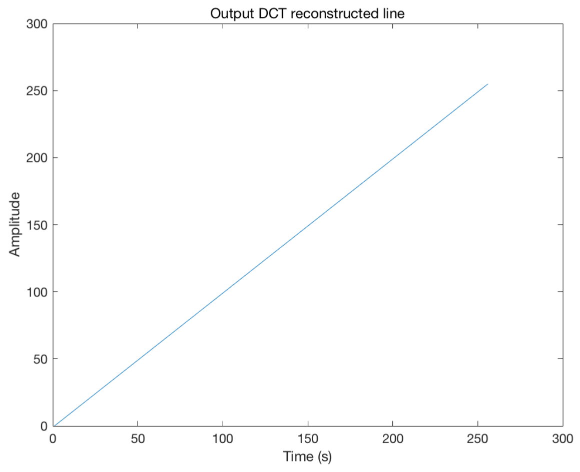


Figure 5: Output DCT reconstructed line

(2) $\frac{64}{256}$

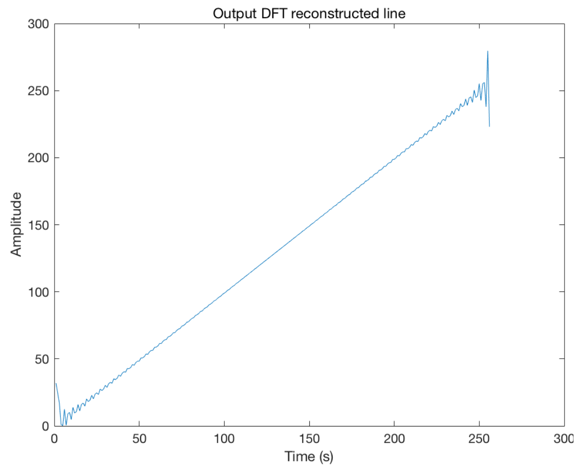


Figure 6: Output DFT reconstructed line

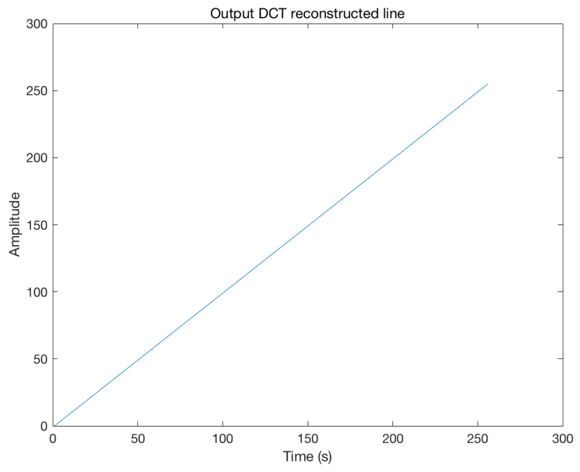


Figure 7: Output DCT reconstructed line

(3) $\frac{96}{256}$

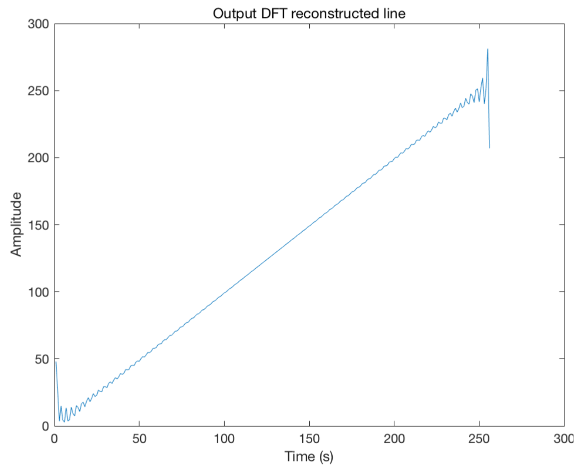


Figure 8: Output DFT reconstructed line

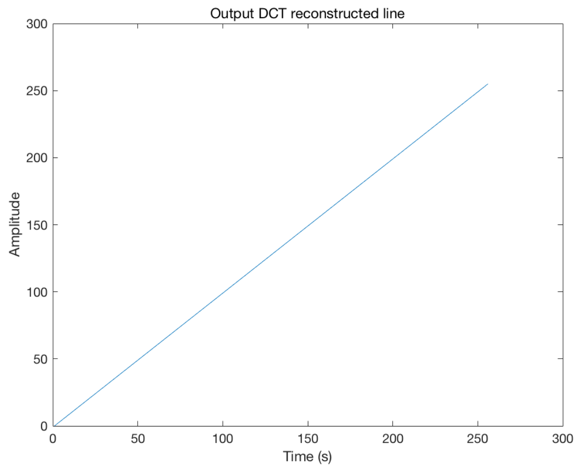


Figure 9: Output DCT reconstructed line

(4) $\frac{128}{256}$

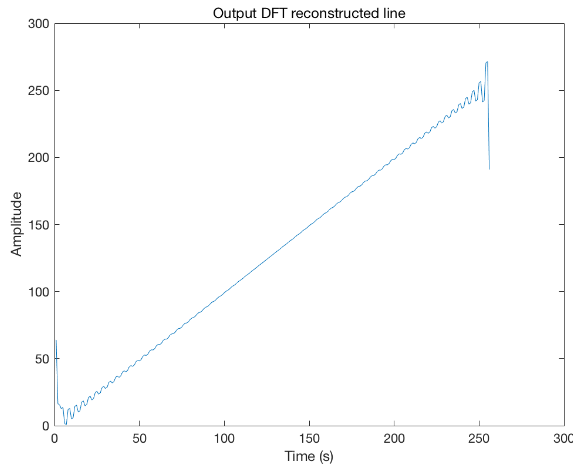


Figure 10: Output DFT reconstructed line

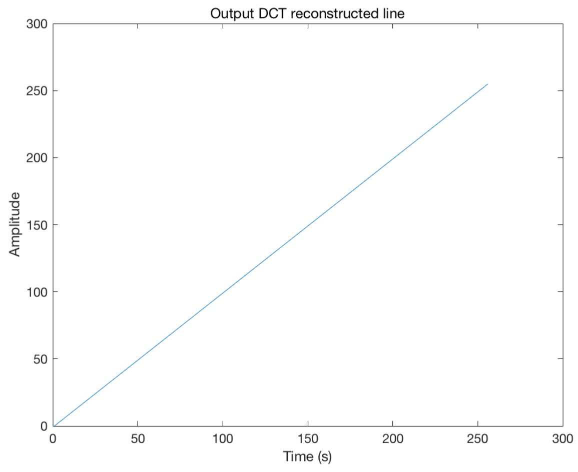


Figure 11: Output DCT reconstructed line

(5) $\frac{160}{256}$

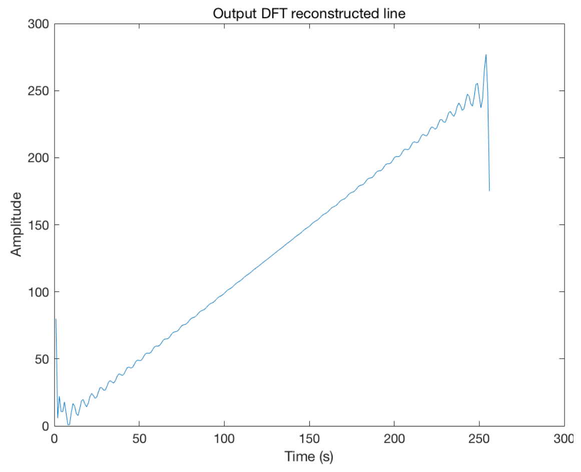


Figure 12: Output DFT reconstructed line

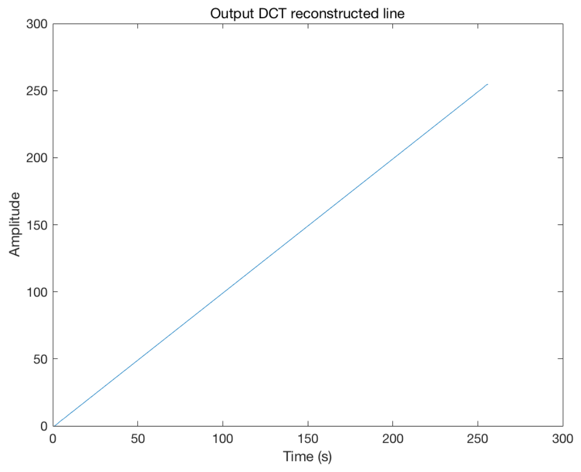


Figure 13: Output DCT reconstructed line

(6) $\frac{192}{256}$

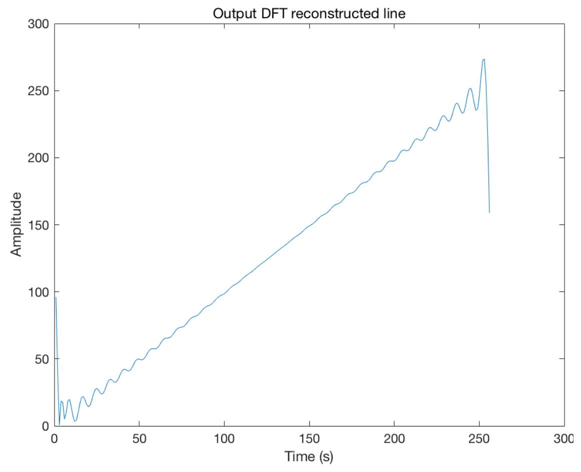


Figure 14: Output DFT reconstructed line

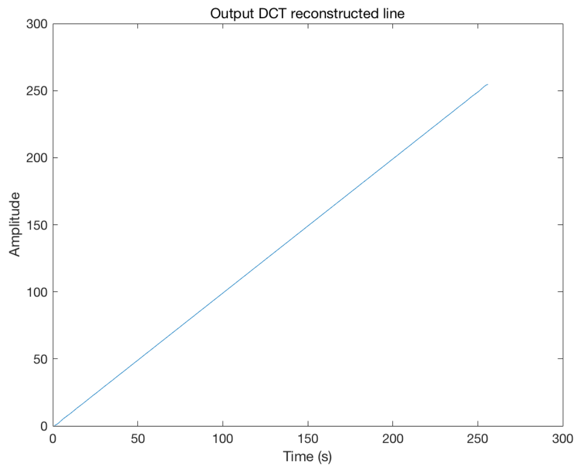


Figure 15: Output DCT reconstructed line

(7) $\frac{224}{256}$

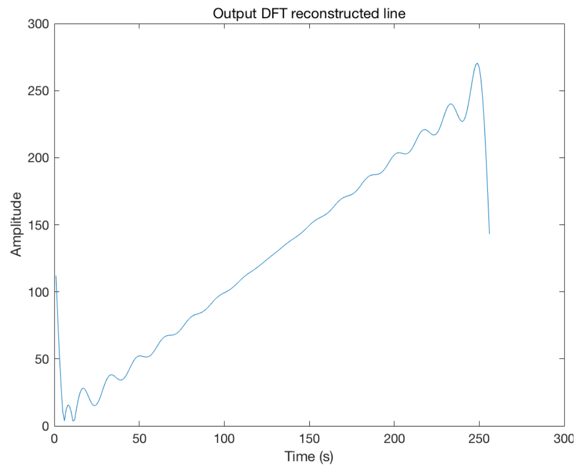


Figure 16: Output DFT reconstructed line

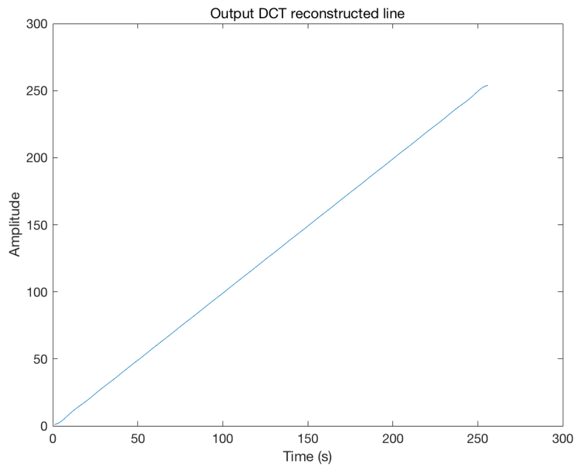


Figure 17: Output DCT reconstructed line

Then we can obtain the PSNR from the figure4 to figure17, the psnr table will be shown here:

Table 1: The PSNR of DFT and DCT

PSNR	32/256	64/256	96/256	128/256	160/256	192/256	224/256
DFT	-9.0873	-12.2730	-14.3497	-16.1008	-17.8520	-19.9291	-23.1168
DCT	67.0508	48.3202	42.0896	36.8364	31.5835	25.3546	15.8040

Based on the previous figure4 to figure17, which are the output reconstructed line and the PSNR table, we can obtain some significant conclusions:

- When the percentage of zeroed high frequencies increase, the PSNR of DCT and DFT will decrease. In another word, the percentage will make a different on the quality of the output reconstructed lines.

- After we observe the figure4 to figure17, we can find that the output DCT line is much straight than DFT line. Additional, when we increase the percentage of zeroed high frequencies, we can find that difference between different DFT output reconstructed lines, but it is difficult to find the difference of the whole DCT output lines. That is also the reason why the psnr of DCT is always higher that the DFT when they are under the same percentage of zeros high frequencies.

The matlab code of the second part will be shown here:

```matlab
%(1)
x=0:1:255;%generate a vector x = [1:256]
N =256;
t=0:N-1;
subplot(3,3,1),plot(t,x);%plot vector x
xlabel('Time (s)');
ylabel('Amplitude');
title('Input sequence')

%DFT
y_dft=fft(x);%Apply Fast Fourier transform to x
subplot(3,3,2),plot(0:N-1,abs(y_dft));
xlabel('Frequency');
ylabel('|X(k)|');
title('DFT');
%high frequency to zero
y_fftshift=fftshift(y_dft);%Apply fftshift to result of fft
subplot(3,3,3),plot(log(abs(y_fftshift)));
y_fftshift(1:32)=0;%Make high frequency component to 0
y_fftshift(225:256)=0;
subplot(3,3,4),plot(t,abs(y_fftshift));
y_idft=ifft(fftshift(y_fftshift));%applt inverse fourire transform
subplot(3,3,5),plot(abs(y_idft));%plot the result

%DCT
y_dct=dct(x);%Apply Discrete cosine transform to x
subplot(3,3,6),plot(abs(y_dct));%plot the result
xlabel('Frequency');
ylabel('|X(k)|');
title('DCT');
%High frequency to zero
y_dct(225:256)=0;%make high frequency to 0
subplot(3,3,7),plot(t,y_dct);%plot the results
y_idct=idct(y_dct);%Apply inverse DCT
subplot(3,3,8),plot(y_idct);%plot the results

%%%PSNR
PSNR_DFT=psnr(y_idft,x);%calculate the psnr of DFT
PSNR_DCT=psnr(y_idct,x);%calculate tge psnr of DCT
```

2 Task2

2.1 Run the code

```
%%%%(1)%%%%
width = 256;
N=width/16;
n=[1:width]-1;
x = 1+cos(2*pi*n/N);
im = ones(width, 1)*x;
figure,imshow(abs(im));
```

Then we run the code, the output will be shown in the following figure:

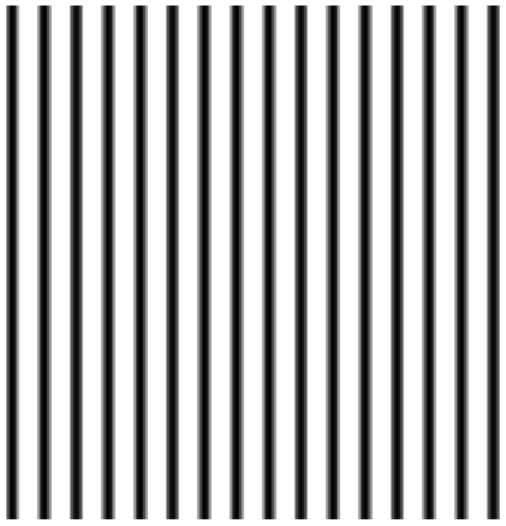


Figure 18: Output figure

The figure18 shows the output of the code. We can easy find that this figure have periodicity.

2.2 Display the 2D-DFT

```
%%%%(1)%%%%
width = 256;
N=width/16;
n=[1:width]-1;
x = 1+cos(2*pi*n/N);
```

```
im = ones(width, 1)*x;
figure,imshow(abs(im));
%%%%%2D-DFT%%%%%%
im_dft=fft2(im);
figure,imshow(log(abs(im_dft)),[]);
im_shift=fftshift(im_dft);%Apply fftshift
figure,imshow(log(abs(im_shift)),[]);
```

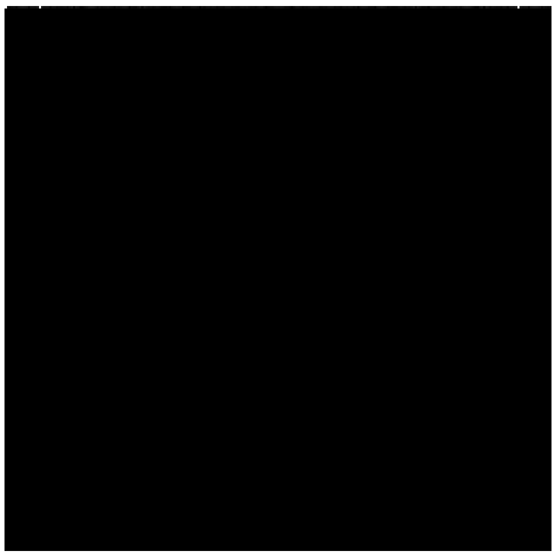


Figure 19: 2D-DFT

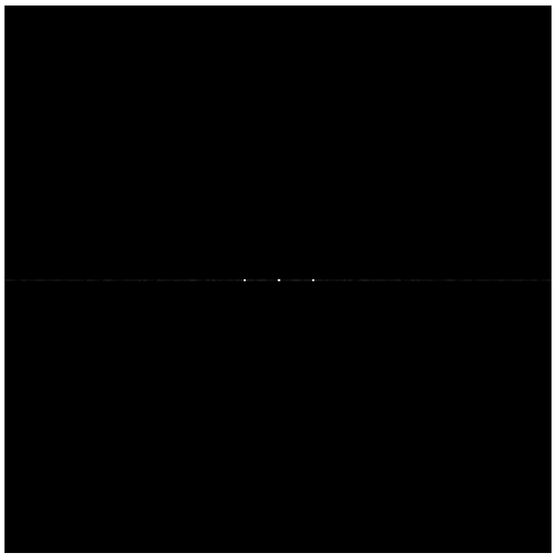


Figure 20: After fftshift function

- When we observe figure19, the 2D-DFT of the previous result, im. We can find that there are three white points on the top of the figure, two on the left side, the other one on the right side. It is not easy to find these three points.
- After the function fftshift, we can easily find that these three points have been moved to the center of this figure, the three white dots are lined up in a row, which are easy to observe.

2.3 Change N

```
%%%%(1)%%%%
width = 256;
N=width/8;
n=[1:width]-1;
x = 1+cos(2*pi*n/N);
im = ones(width, 1)*x;
figure,imshow(abs(im));
%%%%%2D-DFT%%%%%%
im_dft=fft2(im);
figure,imshow(log(abs(im_dft)),[]);
im_shift=fftshift(im_dft);%Apply fftshift
figure,imshow(log(abs(im_shift)),[]);
```

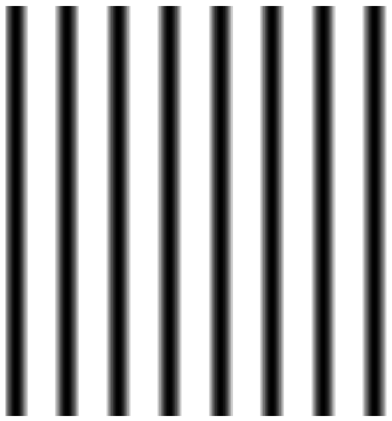


Figure 21: Output figure

Then we compare the figure18 and figure21, we can easily find that the period of figure21 is larger than figure18, because N in figure21 is larger than the N in figure18.

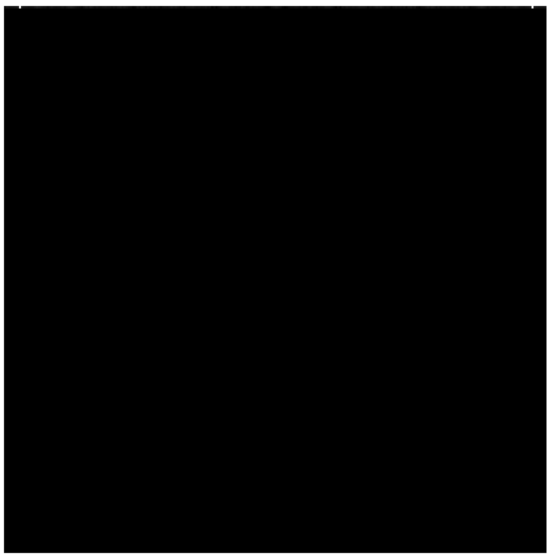


Figure 22: 2D-DFT

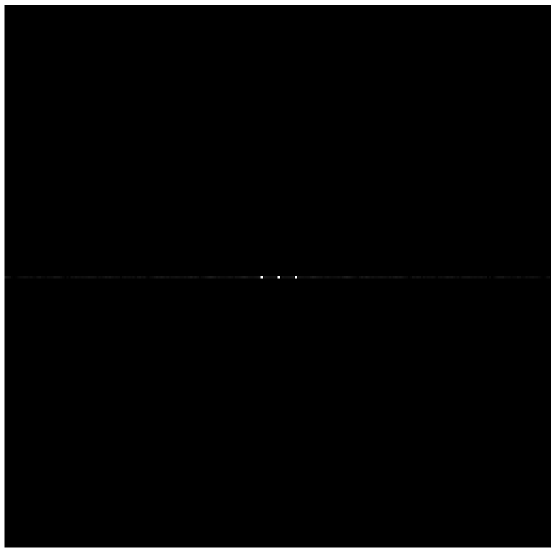


Figure 23: After fftshift function

- We compare the figure19 and figure22, those two figures are all the results of 2D-DFT, we can find that there are three white points on the top of the image. However, the distance of these three white points are different in figure19 and figure22. The reason why the distance are different because the period of figure19 and figure22 are different. The period in figure22 is larger than the period in figure19. The frequency in figure19 is larger than figue22.
- Then we compare the figure20 and figure23, which are results after the function fftshift. We can easily find that the distance of three white points in figure20 is larger than the distance in figure23. Because the frequency of figure20 is larger than the frequency of figure20.
- The value of N will make a difference on the distance of the white points. In another word, the value determines the period of the image. After fourier transform, N will influence the distance.

2.4 Change *x*

2.4.1 N=width/16

```
%%%%(1)%%%%
width = 256;
N=width/16;
n=[1:width]-1;
x = 1+cos(2*pi*n/N)+ cos(4*pi*n/N);
im = ones(width, 1)*x;
figure,imshow(abs(im));
%%%%%2D-DFT%%%%%%
im_dft=fft2(im);
figure,imshow(log(abs(im_dft)),[]);
im_shift=fftshift(im_dft);%Apply fftshift
figure,imshow(log(abs(im_shift)),[]);
```

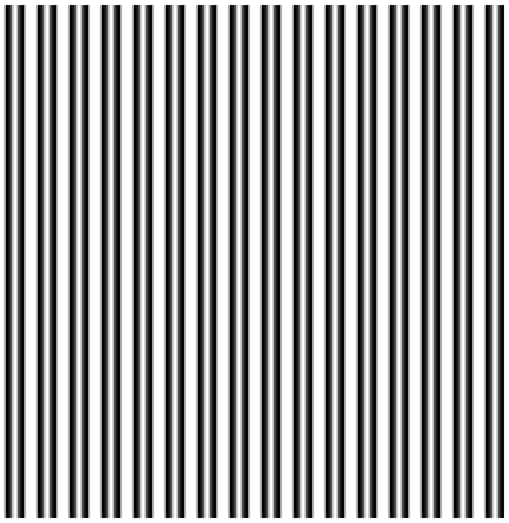


Figure 24: Output figure

Figure 25: 2D-DFT

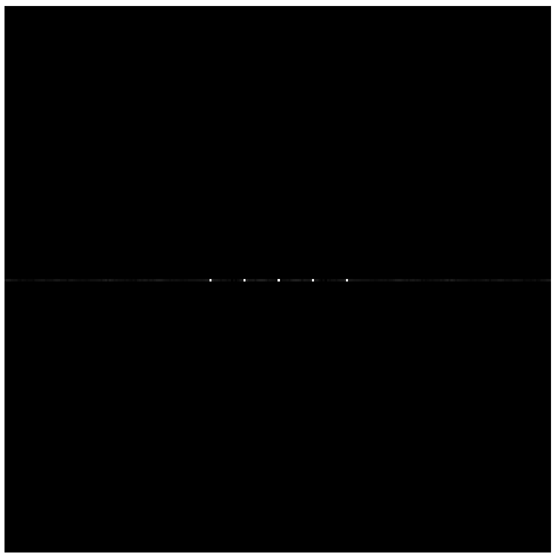


Figure 26: After fftshift function

- When we add a cosine component to the x, the output 2D-DFT graph also added a new component. When we compare the figure24 and figure18, we can find that the period of these two images does not change. However, in figure24, there is a new component, which should be a new cosine component.
- We compare figure20 and figure26: The distance between the white points does not change. However, in figure20, there are only three white points, but in figure26, there are 5 white points, which means after FFT, the value of x will make a difference on the number of white points.

2.4.2 N=width/8

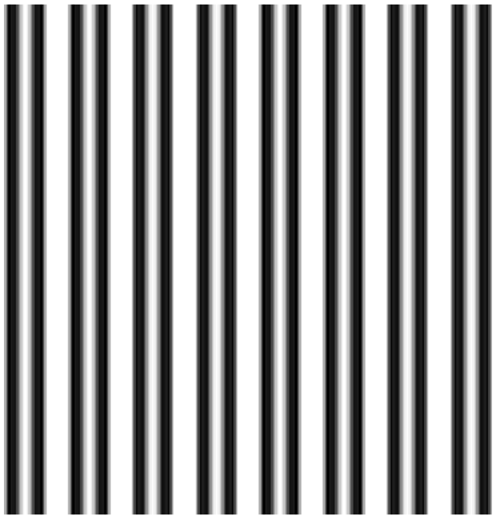


Figure 27: Output figure

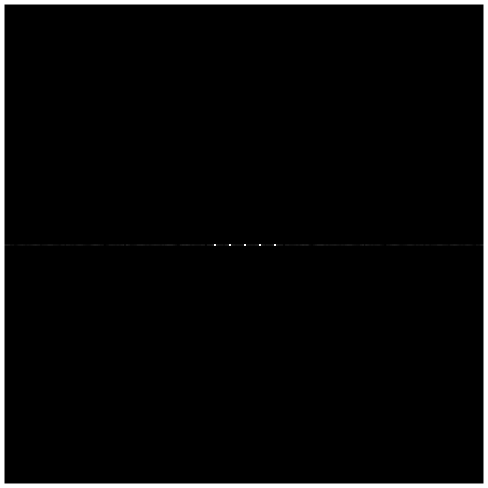


Figure 28: 2D-DFT

Figure 29: After fftshift function

- Compare figure21 and figure27, it is found that the value of N influence the period of the results of 2D-DFT, but it does not change the components of the results.
- Then we compare the figure29 with figure23: Those two figure have the same distance of white points, but figure29 has 5 white points but figure23 only has three.
- The value if N can determine the period of input value. After Fourier transform, the period will determine the distance of the white points in frequency domain. Additional,

the value of x can determine the components of the input value. When x includes both sine and cosine components, the output results after Fourier transform will have more white points in frequency domain.

3 Task3

In this task, the students should design a filter, which could remove the fence in the photo, which is a JPEG photo. If we want to remove the fence in the fence in the frequency domain, we should first apply Fourier transform to this photo and then apply a low pass filter.

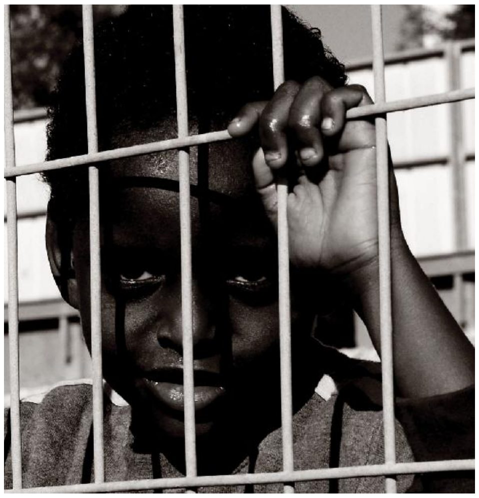


Figure 30: The original photo

Based on the figure30, the original photo. The white hence could be considered as the high frequency components in frequency domain. Then we apply Fast Fourier transform for this photo, and apply fftshift function to the result:

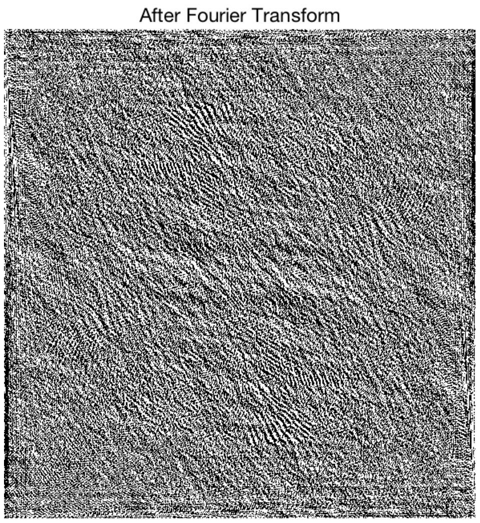


Figure 31: 2D-DFT

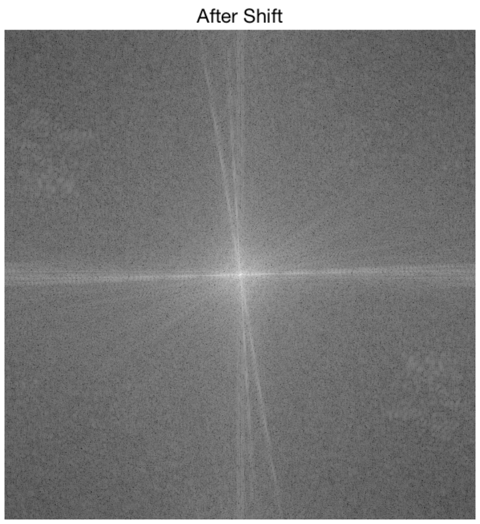


Figure 32: After fftshift function

- We observe the figure32, we can find there are two three significant lines. The first line is a horizontal line, the other two lines are vertical lines, which could be considered as the horizontal fence and vertical line. Remove the fence in time domain is equals to remove the white lines in frequency domain. Therefore, we need a filter to remove the white lines in frequency domain.

- There are two significant low pass filters that we can choose to use. The first one is ideal low pass filter:
  Assume that the cut-off frequency in frequency domain is D_0, then the transfer fucntion of ideal low pass filter is:

$$H(u,v) = \begin{cases} 1, & D(u,v) \le D_0 \\ 0, & D(u,v) > D_0 \end{cases}$$

  In this equation, $D(u,v) = (u^2 + v^2)^{\frac{1}{2}}$ means the distance from point (u,v) to the origin point, D_0 means the distance from cut-off frequency to the origin point.

- The second filter is Butterworth low pass filter, the transfer function of this filter is:

$$H(u,v) = \frac{1}{1 + [\frac{D(u,v)}{D_0}]^{2n}}$$

  In this equation, D_0 means the cut-off frequency.

- In the butterworth filter, the signal is continuous attenuation, which is different from the ideal low pass filter. In another word, if we use butterwordth low pass filter, we can avoid the blur of the output photo. However, if we choose the ideal low pass filter, the output photo will be much blur than the butterworth filter. Therfore, in this task, we choose to use Butterworth filter to remove the fence.

After ifftshift

Figure 33: The frequency domain after filter

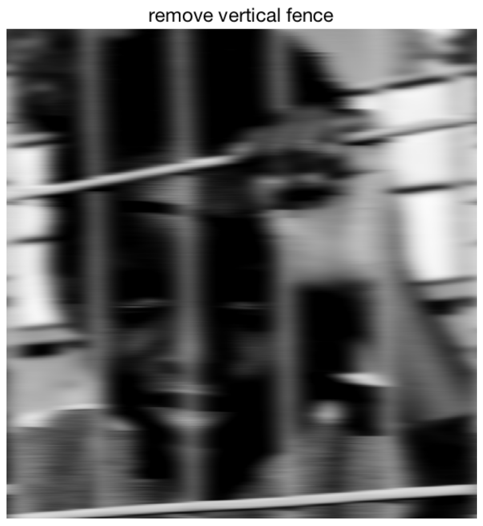


Figure 34: Output photo removed vertical fence

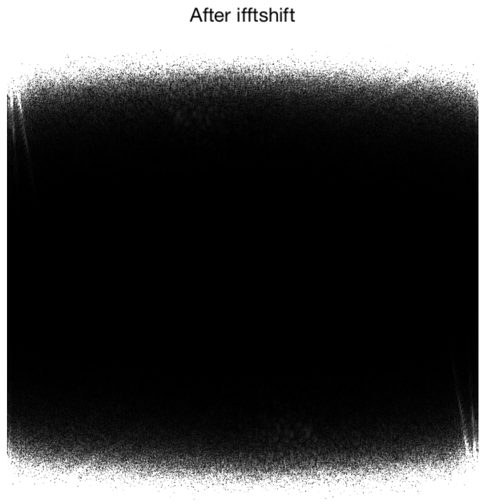


Figure 35: The frequency domain after filter

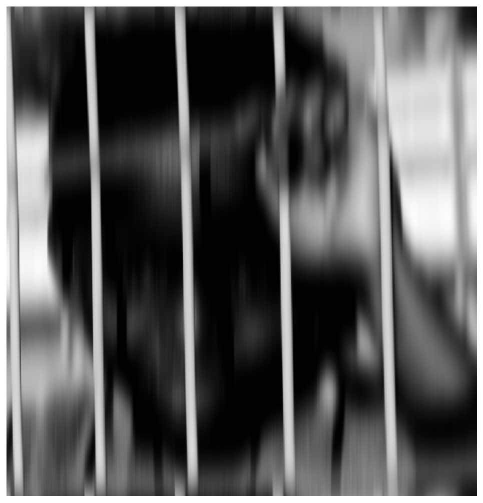


Figure 36: Output photo removed horizontal fence

- In the figure34, after the low pass filter, the photo remove the vertical fence. And figure33 shows the the output photo in frequency domain.
- In figure35, which shows the frequency domain, we can find that the horizontal components has been removed. In figure36, the horizontal fence has been removed.

- There are still some limitations in this filter, if we both remove horizontal and vertical fence, the output photo will be much blur, and lost lots of significant information.

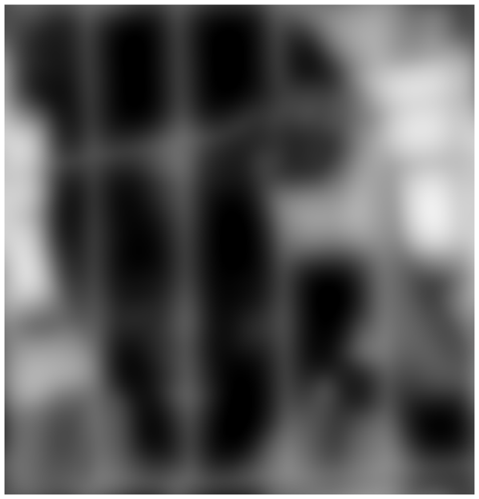


Figure 37: Output photo after filter

```matlab
x=imread('fence.jpg');
x=rgb2gray(x);%change the photo into gray
x=double(x);%change type to double
g=fft2(x);%apply FFT to photo
figure,imshow(g),title('After Fourier Transform');
g=fftshift(g);%Apply fftshift
figure,imshow(log(abs(g)),[]),title('After Shift');
[M,N]=size(g);%obtain the size of input frequency domain photo
nn=2;
d0=5;%set the cut-off frequency
m=fix(M/2);n=fix(N/2);
for i=1:M
    for j=1:N
        d=sqrt((j-n)^2);%the vertical fence
        d1=sqrt((i-m)^2);%the horizontal fence
        h=1/(1+0.414*(d/d0)^(2*nn));%calculate the low pass filter ...
            transfer function
        output(i,j)=h*g(i,j);
    end
end
output=ifftshift(output);
figure,imshow(abs(output)),title('After ifftshift');
output1=fftshift(output);
J2=ifft2(output);
J3=uint8(J2);
figure,imshow(J3);
```

4 Task4

4.1 Zero-padding to scale-up

```
function output = zero_padding(input,resize)
input_size=size(input);%obtain the photo size of input
if input_size≥resize%when target size smaller than the original size
    disp('wrong input resize');
    return
end
%obtain the four side
input_left=(resize(2)-input_size(2))/2;
input_right=(resize(2)-input_size(2))/2;
input_top=(resize(1)-input_size(1))/2;
input_bottom=(resize(1)-input_size(1))/2;
%make the new size to 0
output_1=[zeros(input_top,resize(2))];
output_2=[zeros(input_size(2),input_left) input ...
    zeros(input_size(2),input_right)];
output_3=[zeros(input_bottom,resize(2))];
%obtain the new matrix, new photo
output=[output_1;output_2;output_3];
end
```

```
Ref=imread('lenna512.bmp');
A=imread('lenna_ds440.bmp');%load the image
new_size=[512 512];%inout the new size
%ser the normalization factor,depens on the original size and new size
normalization1=new_size(:,1)/size(A,1);
normalization2=new_size(:,2)/size(A,2);
normalization=normalization1*normalization2;
subplot(2,3,1),imshow(A),title('Original photo');%show original photo
A_fft=fft2(A)*normalization;%apply fft on original photo
subplot(2,3,2),imshow(log(abs(A_fft)),[]),title('frequency domain ...
    photo');%show fft frequency photo
A_fft=fftshift(A_fft);%shift high frequency
subplot(2,3,3),imshow(log(abs(A_fft)),[]),title('fftshift ...
    photo');%show iamge after fftshift
y=zero_padding(A_fft,new_size);%apply zero-padding function
subplot(2,3,4),imshow(log(abs(y)),[]),title('after zero-padding ...
    photo');%show zero-padding photo
y_fftshift=fftshift(y);%shift frequency zero-padding
subplot(2,3,5),imshow(log(abs(y_fftshift)),[]),title('fftshift ...
    photo');%show photo after zero-padding
output=ifft2(y_fftshift);%apply inverse fft
output=uint8(output);%change data type to uint8
subplot(2,3,6),imshow(output),title('New photo');%show new photo
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
output_nearest=imresize(A,[512,512],'nearest');
output_lanczos3=imresize(A,[512 512],'lanczos3');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
PSNR_zero=psnr(output,Ref);
PSNR_nearest=psnr(output_nearest,Ref);
PSNR_lanczo3=psnr(output_lanczos3,Ref);
```

4.2 Scale the photo

4.2.1 im420

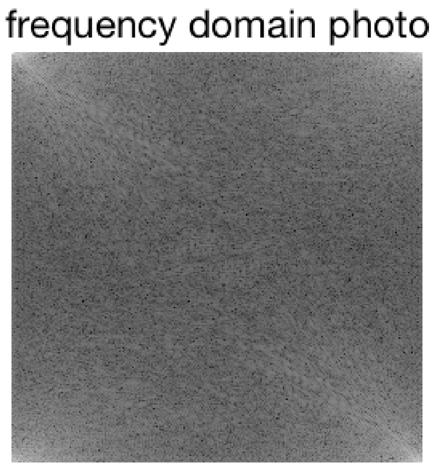


Figure 38: The frequency domain after DFT

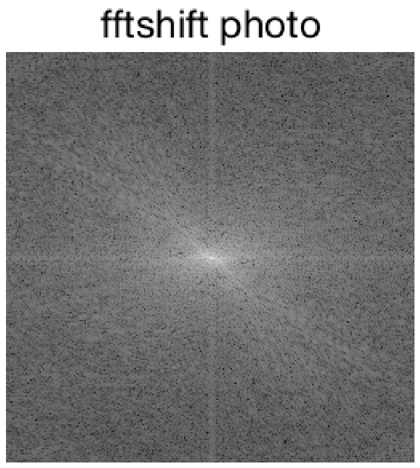


Figure 39: The frequency domain after fftshift

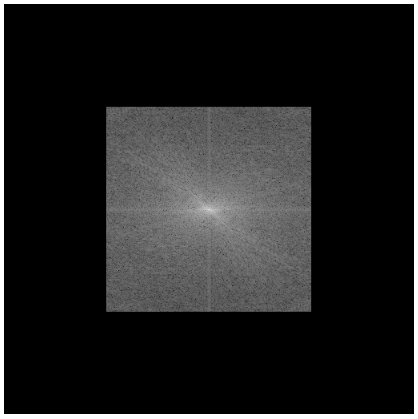


Figure 40: The frequency domain after zero-padding

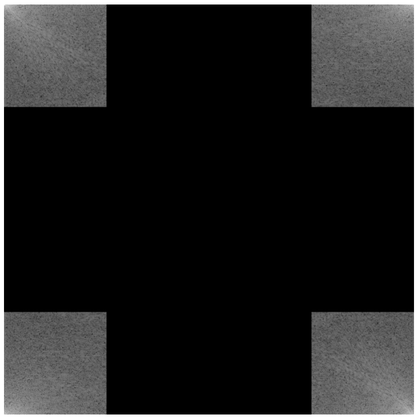


Figure 41: Zero-padding after shift

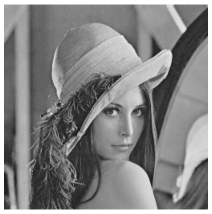


Figure 42: The original photo

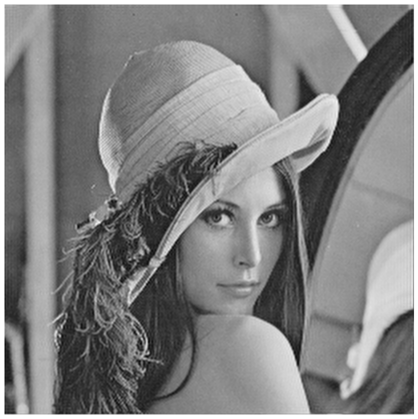
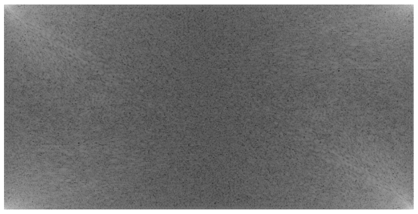
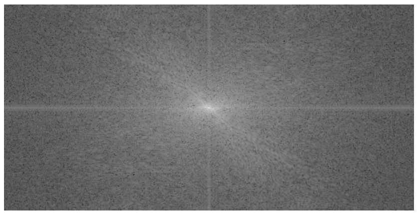
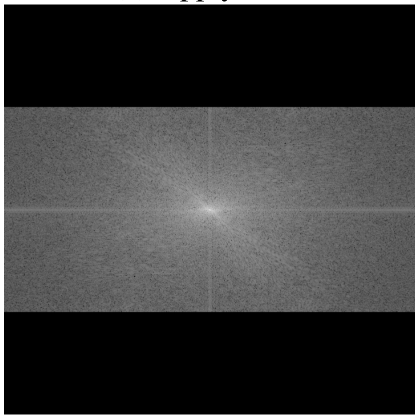


Figure 43: The new photo after zero-padding

4.2.2 im440


(a) Apply DFT


(b) Apply shift


(c) Apply zero-padding in frequency domain

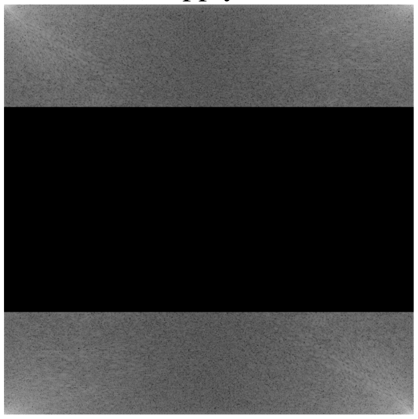

(d) After fftshift function

Figure 44: Apply zero-padding technique

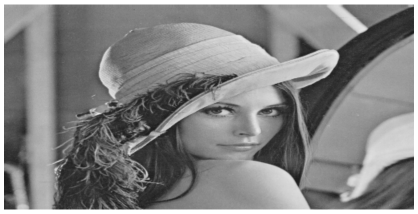


Figure 45: The original photo

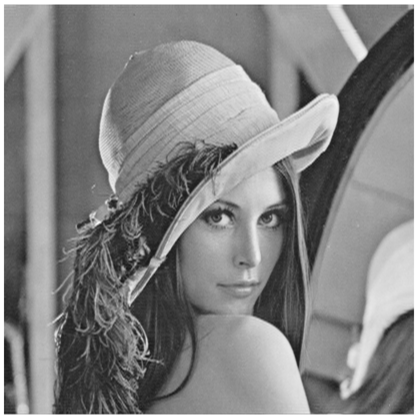


Figure 46: The new photo after zero-padding

4.2.3 PSNR

After scale-up the figure45 and figure42, we can evaluate the PSNR of the two new photos, figure46 and figure43:

Table 2: The table of PSNR

	im420	im440
zero-padding	32.3187	36.8003
Imresize, 'nearest'	28.2869	32.7515
Imresize, 'lanczo3'	29.9145	34.4147

According to figure38 to figure46, we can draw some important significant conclusions:

- Observe the figure42 and figure43, we can find that the zero-padding scale-up this photo from 256×256 to 512×512 without lose the information. Also in the frequency domain, the zero-padding add the black area on the four sides. Then we observe the figure45 and figure46, figure45 shows a photo with 256×512, which length not equals to height. Then we use zero-padding to scale-up this photo to 512×512 without lose the information.

- In the frequency domain, when the photo apply zero-padding, the output figure will be fill with 0Hz area, from figure40, we can see it is black area. In the time domain, the zero-padding could only change the resolution of the original photo. However, if we did not multiply a normalization factor, the color of output new photo will be unmoral. The normalization factor depends on the size of new photo and original photo.

- As for the scale-up for photo, the psnr of zero-padding is larger than nearest and lanczo3. In another word, after scale-up, the photo quality of zero-padding is much better than 'nearest' and 'lanczos3'.

- Therefore, zero-padding can make a difference on scale-up: Add zero frequency in the frequency domain, and scale-up the photo in time domain. The psnr for zero-padding technique is lager than the other two methods, 'nearest' and 'lanczos3'.

5 Task5

5.1 Basis functions

The two dimension discrete cosine transform(DCT) can be written as:

$$F(u,v) = a(u)a(v)\sum_{m=0}^{N-1}\sum_{n=0}^{N-1} f(m,n)cos\frac{(2m+1)u\pi}{2N}cos\frac{(2n+1)v\pi}{2N}$$

Where, $m,n = 0,1,2,...,N-1$,

$$a(u) = a(v) = \begin{cases} \sqrt{\frac{1}{N}}, & u = 0, v = 0 \\ \sqrt{\frac{2}{N}}, & u,v = 1,2,...,N-1 \end{cases}$$

According to the previous equation, we write this equation into matlab, and obtain the following photo:

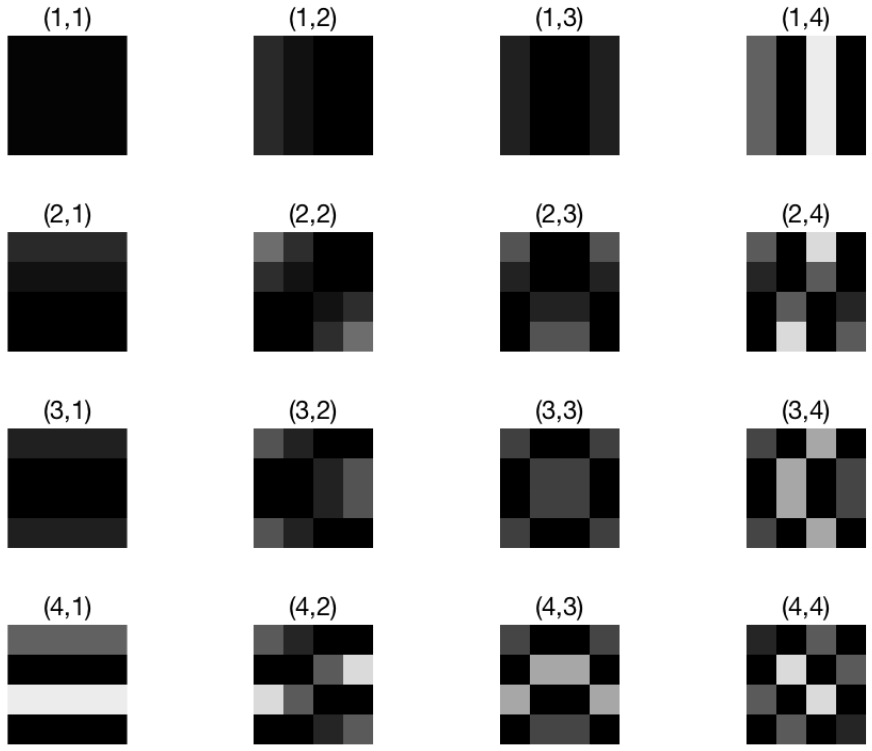


Figure 47: Basis functions

- The two-dimensional DCT is obtained applying the 1-D transform to the columns (rows) and then to the rows (columns) independently.
- When the input size $N = 4$, there will be 4×4 basis images after DCT.

```
function F=dct_basic(N)
f=zeros(N);
F=cell(N,N);
for u=0:N-1%define u,row
    for v=0:N-1%define v,column
        for m=0:N-1
            for n=0:N-1
f(m+1,n+1)=cos((2*m+1)*u*pi/2/N)*cos((2*n+1)*v*pi/2/N);
            end
        end
        F{u+1,v+1}=f;%define a(u) and a(v)
    end
end
%%%%%%%%%%%%%%multiply v and u
F{1,1}=F{1,1}/N;%when u an v equals to 0,N=1
for u=1:N-1
    F{1,u}=sqrt(2)*F{1,u}/N;
    F{u,1}=sqrt(2)*F{u,1}/N;
end
for u=1:N-1%whenu,v are not equals to 0
    for v=1:N-1
        F{u,v}=2*F{u,v}/N;
    end
end
end
```

```
I=dct_basic(4);
axis off;
subplot(4,4,1),imshow(I{1,1}),title('(1,1)');
subplot(4,4,2),imshow(I{1,2}),title('(1,2)');
subplot(4,4,3),imshow(I{1,3}),title('(1,3)');
subplot(4,4,4),imshow(I{1,4}),title('(1,4)');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
subplot(4,4,5),imshow(I{2,1}),title('(2,1)');
subplot(4,4,6),imshow(I{2,2}),title('(2,2)');
subplot(4,4,7),imshow(I{2,3}),title('(2,3)');
subplot(4,4,8),imshow(I{2,4}),title('(2,4)');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
subplot(4,4,9),imshow(I{3,1}),title('(3,1)');
subplot(4,4,10),imshow(I{3,2}),title('(3,2)');
subplot(4,4,11),imshow(I{3,3}),title('(3,3)');
subplot(4,4,12),imshow(I{3,4}),title('(3,4)');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
subplot(4,4,13),imshow(I{4,1}),title('(4,1)');
subplot(4,4,14),imshow(I{4,2}),title('(4,2)');
subplot(4,4,15),imshow(I{4,3}),title('(4,3)');
subplot(4,4,16),imshow(I{4,4}),title('(4,4)');
```

5.2 DCT

In this program, we should write a function, which could apply DCT with the previous basis functions, and combine them together into one photo.

```
function [im_DCT, DCT_bases]=projection_an_image_on_its_DCT_bases(im)
N=size(im,1);%length of input image
DCT_bases=dct_basic(N);%apply the previous basic function
im_DCT=zeros(N,N);%zeros intital matrix
for i=1:N
    for j=1:N
        x=DCT_bases{i,j}.*im;
        im_DCT(i,j)=sum(x(:));%calculate F(k,l)
    end
end
end
```

5.2.1 Input-1

When the input image are:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

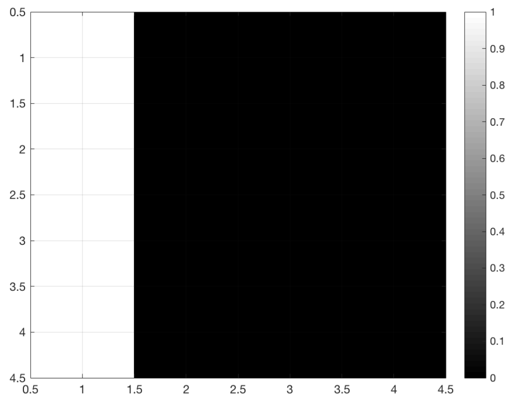


Figure 48: Input figure ’im’

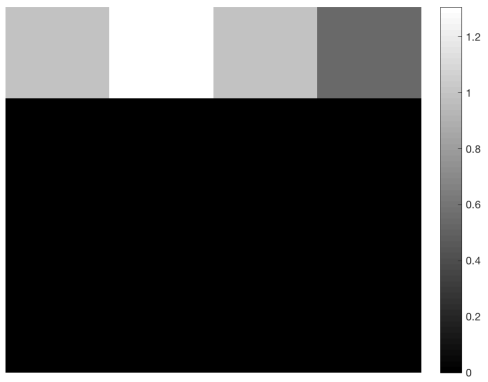


Figure 49: Output figure after DCT

```
N=4;%define the input N
im=[1 0 0 0;1 0 0 0;1 0 0 0;1 0 0 0];%define the input matrix
output=projection_an_image_on_its_DCT_bases(im);%use previous DCT ...
    function
figure,imagesc(im),colormap(gray),colorbar,axis on,grid on;
figure,imagesc(output),colormap(gray),colorbar,axis off;
```

5.2.2 Input-2

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

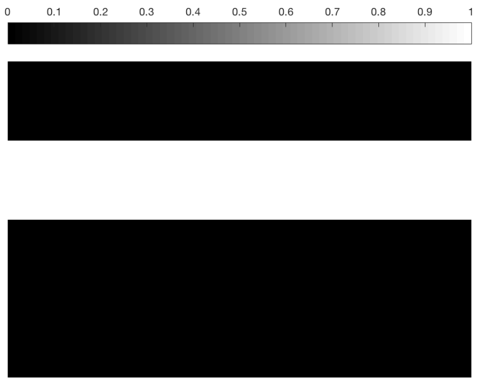


Figure 50: Input figure ’im’

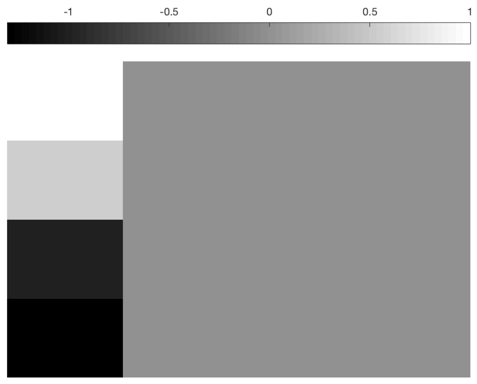


Figure 51: Output figure after DCT

```
N=4;%define the input N
im=[0 0 0 0;1 1 1 1;0 0 0 0;0 0 0 0];%define the input matrix
output=projection_an_image_on_its_DCT_bases(im);%use previous DCT ...
    function
figure,imagesc(im),colormap(gray(64)),colorbar,axis off;
figure,imagesc(output),colormap(gray(64)),colorbar,axis off;
```

5.2.3 Input-3

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

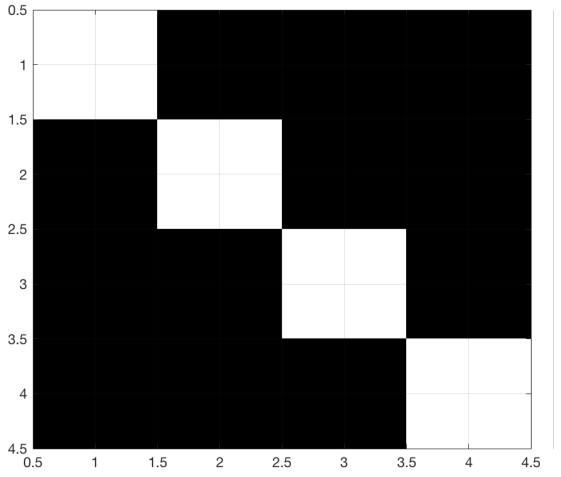


Figure 52: Input figure 'im'

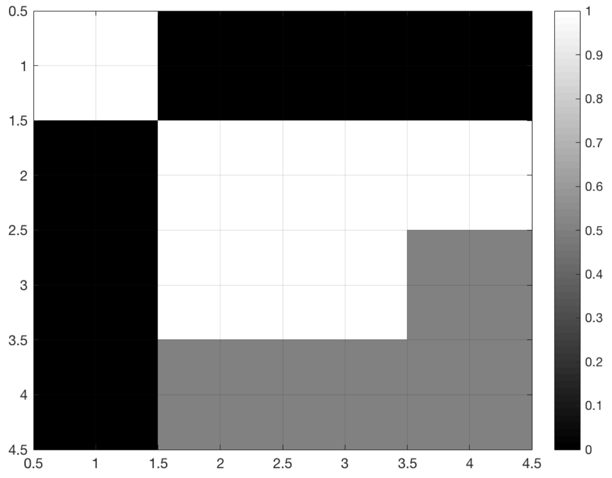


Figure 53: Output figure after DCT

```
N=4;%define the input N
im=[1 0 0 0;0 1 0 0;0 0 1 0;0 0 0 1];%define the input matrix
output=projection_an_image_on_its_DCT_bases(im);%use previous DCT ...
    function
figure,imagesc(im),colormap(gray(64)),colorbar,axis on,grid on;
figure,imagesc(output),colormap(gray(64)),colorbar,axis on,grid on;
```

From figure48 to figure53, we can draw some conclusions:

- When the input is a 4×4 matrix, the output will also be a 4×4 matrix.
- The DCT helps separate the image into spectral sub-bands of differing importance with respect to the image's visual quality. The output image could be considered as the sum of all previous basis functions.

5.3 Inverse DCT

The inverse DCT could be regarded as transform the image from frequency domain to time domain:

$$f(m,n) = a(u)a(v)\sum_{m=0}^{N-1}\sum_{n=0}^{N-1} F(u,v)cos\frac{(2m+1)u\pi}{2N}cos\frac{(2n+1)v\pi}{2N}$$

Where, $m,n = 0,1,2,...,N-1$,

$$a(u) = a(v) = \begin{cases} \sqrt{\frac{1}{N}}, & u = 0, v = 0 \\ \sqrt{\frac{2}{N}}, & u,v = 1,2,...,N-1 \end{cases}$$

Then we write a function in MATLAB to achieve the inverse DCT, and show the input and output image:

```
N=4;%define the input N
inverse_output=inverse_dct(output);%use previous DCT function
figure,imagesc(output),colormap(gray(64)),colorbar,axis on,grid on;
figure,imagesc(inverse_output),colormap(gray(64)),colorbar,axis ...
    on,grid on;
```

```
function F=inverse_dct(im)
[im_DCT, DCT_basis]=projection_an_image_on_its_DCT_bases(im);
N=size(im,1);%obtain the  length of image
F=zeros(N,N);%inversed image to zero matrix
for i=1:N
    for j=1:N
        F=F+DCT_basis{i,j}*im_DCT(i,j);
    end
end
end
```

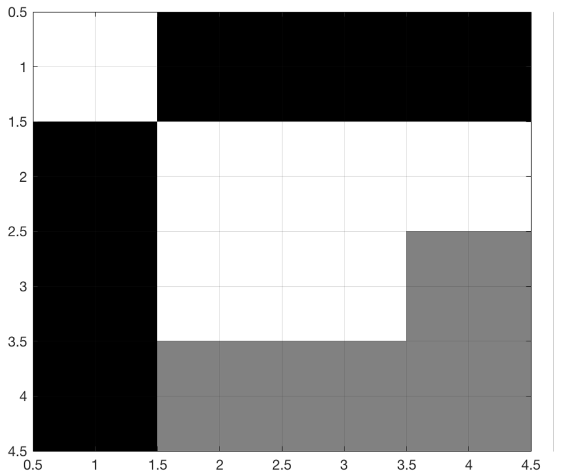


Figure 54: Input figure in frequency domain

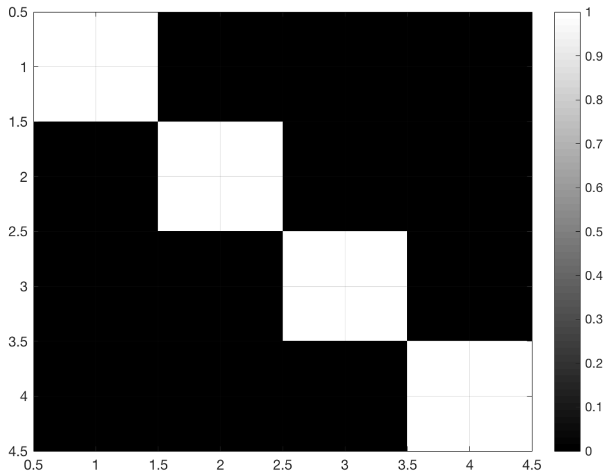


Figure 55: Output figure after inverse DCT

- Observe the figure54 and figure55, we can find that this inverse DCT have transform the image from frequency domain to time domain.