2D-Fourier Transform

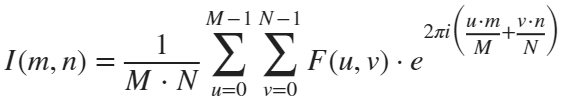
In this section we will implement the 2D Fourier Transform on images and learn about some of its properties.

1.1.1 the 2D FFT

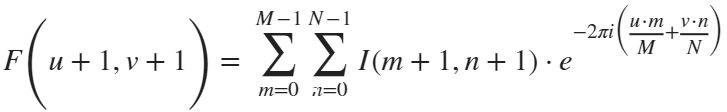
The equations for the FFT and iFFT for an image I of size  are as follows:

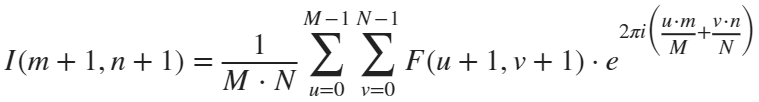


and the inverse transform:

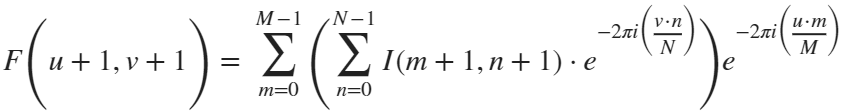


accounting to MATLAB's indexing, we will rewrite the array indices as such:





as for the algorithm, we will implement it by slightly rearranging the equation:



Our code:

function F = dip\_fft2(I)

% Get the size of the input image

[M, N] = size(I);

% Generate arrays representing indices for rows and columns

n = 0:N-1;

m = 0:M-1;

% Generate arrays representing indices for frequency domain

u = 0:M-1;

v = 0:N-1;

% Create matrices of products of indices for later calculations

um = u'.\*m;

vn = v'.\*n;

% Calculate the exponential term in the frequency domain for rows

e1 = exp(-1i \* 2 \* pi \* (vn/N));

% Calculate the exponential term in the frequency domain for columns

e2 = exp(-1i \* 2 \* pi \* (um/M));

% Perform the row-wise Fourier transform using matrix multiplication

sum = I \* e1;

% Perform the column-wise Fourier transform using matrix multiplication

F = e2 \* sum;

end

function F = dip\_ifft2(I)

% Get the size of the input image

[M, N] = size(I);

% Generate arrays representing indices for rows and columns

n = 0:N-1;

m = 0:M-1;

% Generate arrays representing indices for frequency domain

u = 0:M-1;

v = 0:N-1;

% Create matrices of products of indices for later calculations

um = u'.\*m;

vn = v'.\*n;

% Calculate the exponential term in the frequency domain for rows

e1 = exp(1i \* 2 \* pi \* (vn/N));

% Calculate the exponential term in the frequency domain for columns

e2 = exp(1i \* 2 \* pi \* (um/M));

% Perform the row-wise inverse Fourier transform using matrix multiplication

sum = I \* e1;

% Perform the column-wise inverse Fourier transform using matrix multiplication

F = (1/(M\*N)) \* e2 \* sum;

end

1.1.2

We wrote our own function named dip\_fftshift(FFT), shift zero-frequency component to center of spectrum along two dimensions:

function shiftedFFT = dip\_fftshift(FFT)

% Get the size of the input FFT

[M, N] = size(FFT);

% Calculate the center indices for both dimensions

centerX = floor(M / 2) + 1;

centerY = floor(N / 2) + 1;

% Perform the shift along the rows and columns

shiftedFFT = FFT([centerX:M, 1:centerX-1], [centerY:N, 1:centerY-1]);

end

This function takes an input FFT and shifts the zero-frequency components to the center along both dimensions using MATLAB indexing. It first calculates the center indices and then rearranges the FFT values accordingly. The result is the shifted FFT.

1.1.3. we read the beatles.png image accompanied to this assignment, and converted it to grayscale normalized image. We used our function:

function I\_normalized = imread\_normalized(src)

I = imread(src);

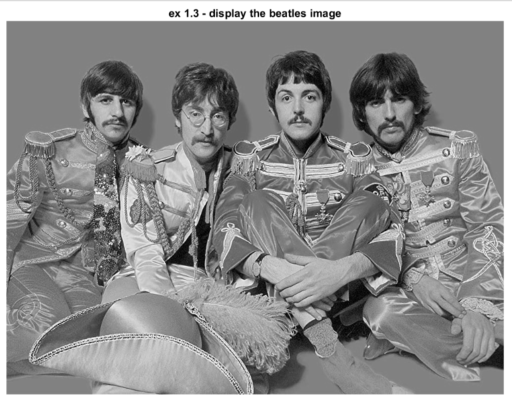
I = rgb2gray(I);

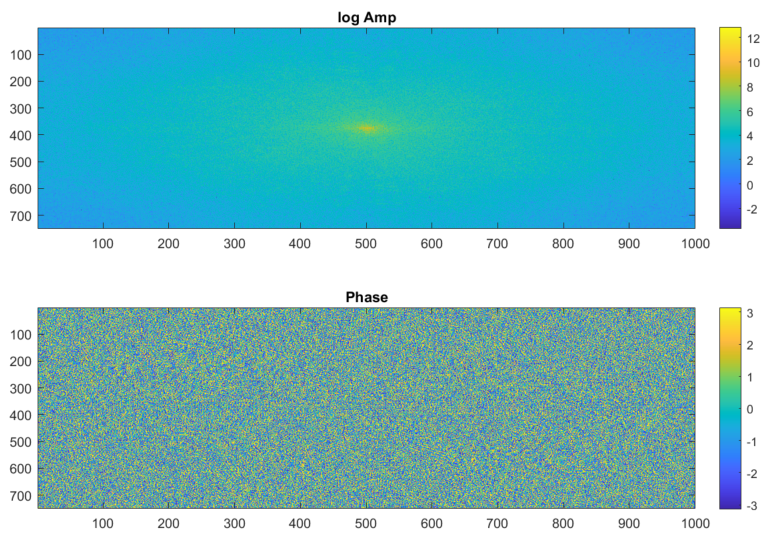
I = im2double(I);

I\_min = min(I(:));

I\_max = max(I(:));

I\_normalized = (I - I\_min) / (I\_max - I\_min);

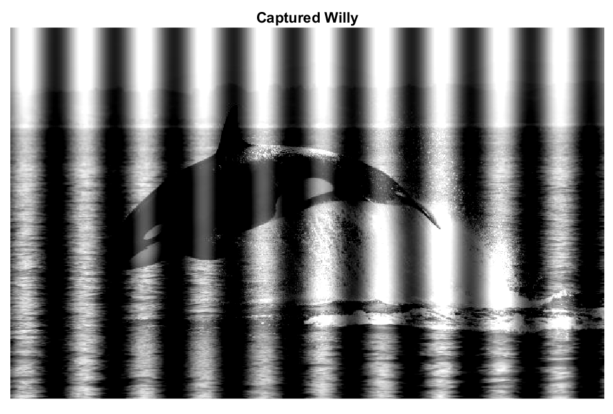
end

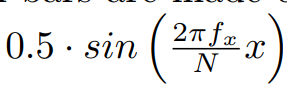
1.1.4. We compute the 2D-FFT of the image and shifted the output image using dip\_fftshift(FFT) function. The log of the amplitude and the phase of the resulting image are shown here:

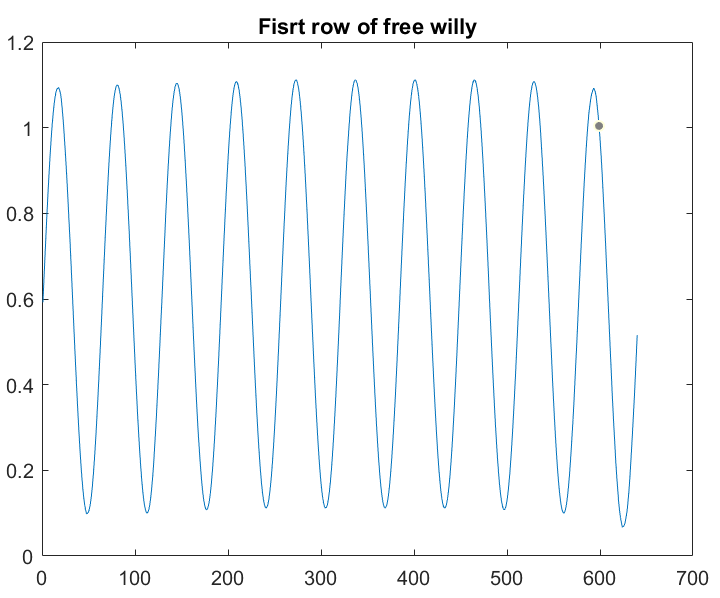
1.1.5. We reconstructed the original image by using our inverse-FFT function. It is identical to the original image !

Note that the output of the iFFT are complex numbers – we display only the real part of the image using imshow(real(-)).

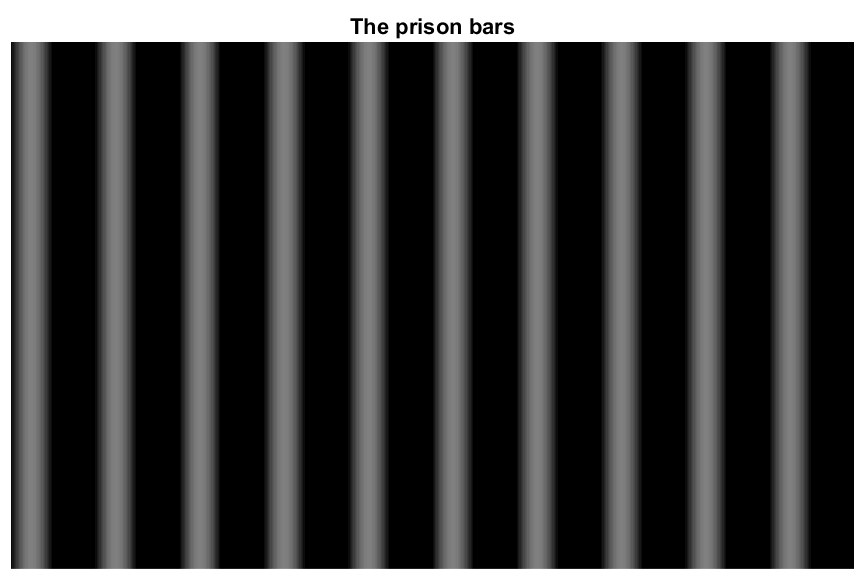
**1.2 Transformation properties**

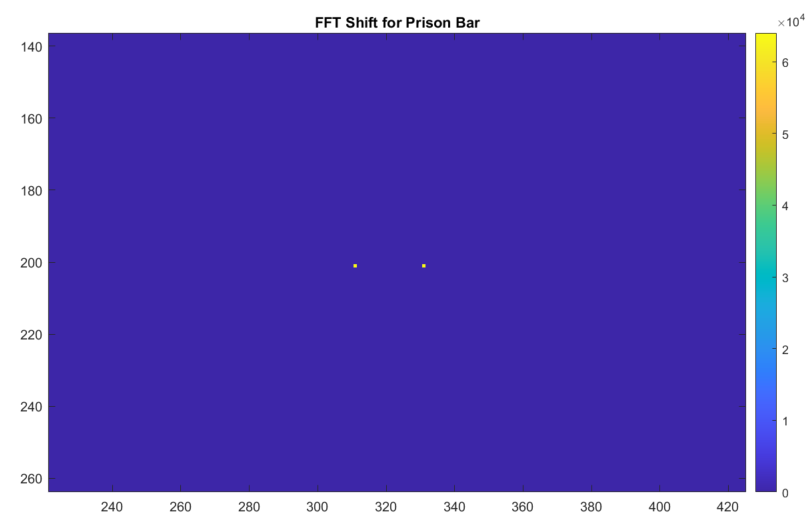
1.2.a. **Linearity (Free Willy)**

1.2.b. Willy the whale is imprisoned. We will free Willy! We are told that the prison bars are made of a sinusoidal signal in the X axis that was added to the original image:

where N is the number columns in the image. Given this image, find the spatial frequency of the prison bars . Let’s plot the first row of the image:

We can see it is indeed a sinusoidal wave. We used the function ‘findpeaks’ that can give us the number of peaks in the graph, and we received . **\*\* אולי להרחיב הסבר**

1.2.c. We created a matrix with a sinusoidal wave with this frequency:

 1.2.d. We compute the 2D-FFT of the prison bars image, and display its amplitude:

We obtained two deltas, meaning two singular points symmetrically positioned in the center of the transformation image. As known, the Fourier transform of a sine consists of two symmetric deltas centered around zero.

We will use the linearity of the FFT to calculate the FFT of just one of the exp,

So, in the same way we will get,

as we expected for the amplitude of the image's FFT.

1.2.e. We can free willy by subtracting the sine frequencies (prison bar) from the image frequencies in the frequency domain. We will transform into the frequency domain using the 2D FFT, subtract the sine frequencies, and go back to the image domain using the 2D iFFT.

We will do all this using our function Free\_Willy(Willy) that returns and displays Willy without the prison bars:

function free\_willy = Free\_Willy(Willy)

% Count the number of peaks in the first row of Willy

fx = size(findpeaks(Willy(1, :)), 2);

% Get the number of columns in Willy (assuming it represents an image)

N = size(Willy, 2);

% Create a meshgrid for the x-axis

x = meshgrid(0:N-1, 1:size(Willy, 1));

% Generate a sine wave as a prison bar pattern

prison\_bar = 0.5 \* sin((2 \* pi \* fx / N) \* x);

% Perform FFT on the images (Willy and prison\_bar)

Willy\_fft = fft2(Willy);

prison\_fft = fft2(prison\_bar);

% Subtract the sine frequencies (prison bar) from the image frequencies

free\_willy\_fft = Willy\_fft - prison\_fft;

% Perform Inverse FFT to get the modified image

free\_willy = ifft2(free\_willy\_fft);

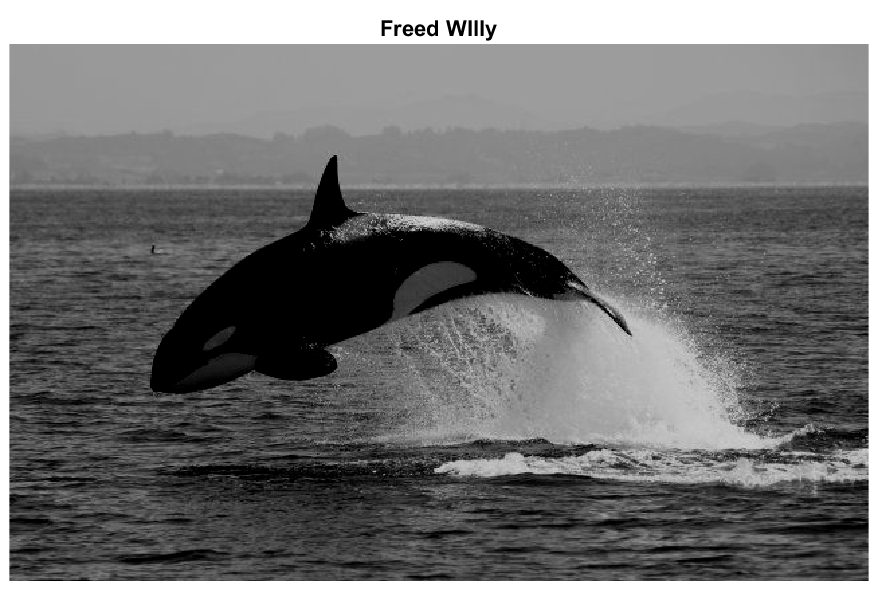
% Display the resulting image

figure;

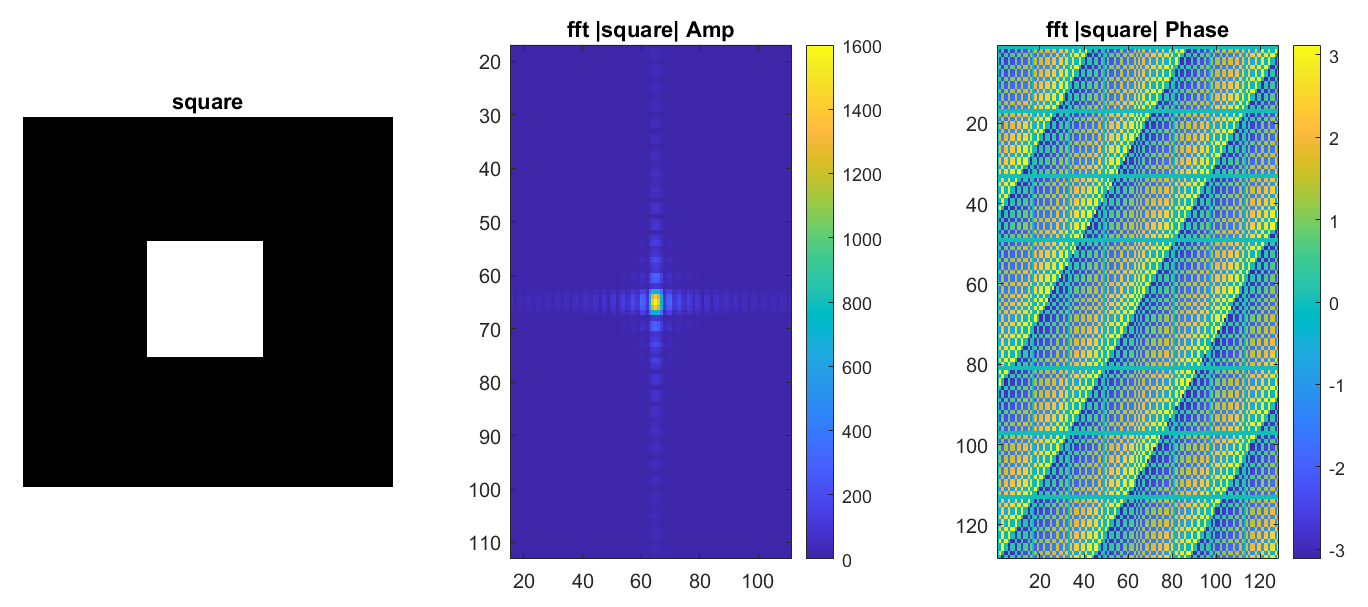
imshow(free\_willy);

title(‘Freed Willy’)

end

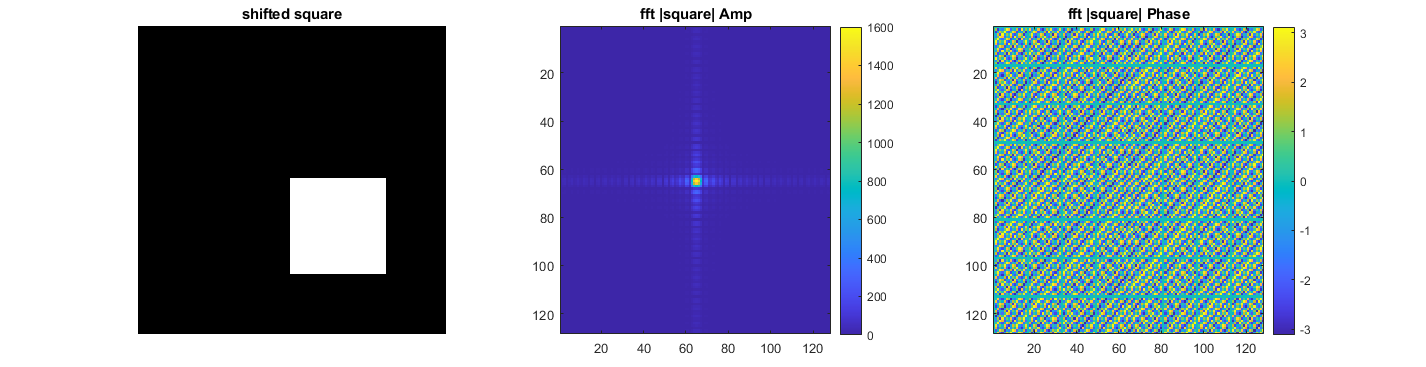
The result is:

**2. Scaling, translation and separability**

2.a. We initialize a 128×128 all-zeros matrix. At the center of it, we placed a 40×40 all-ones square (in pixels 44:83 in each dimension). We display the image and its 2D-FFT:

From the amplitude image, we can see that we got a sinc function – a known result of FFT of a rectangular window. We can see the cut of the function with both x and y axes, and that the main lobe is at (0,0).

2.b now we initialize a square of equal size and locate the square at the bottom right. We displayed the image and its 2D-FFT:

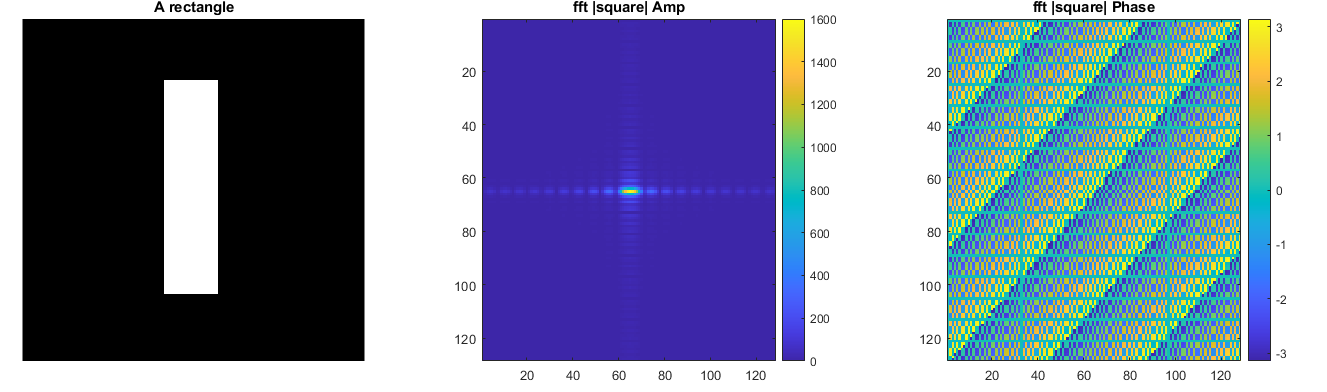


We can see that the amplitude image we receive contains the sinc function again, same as earlier in (a). that is what we would expect, the square is the same as earlier, just moved in space. We know that one of the properties of the fft is that spatial displacements translates to a multiplication of the phase with a linear phase component:

This phase component has an amplitude of 1 thus, there is no change in the amplitude plot due to the multiplication.

The phase on the other hand, changes due to the multiplication. in our case, the square was spatially translated in 2 dimensions: so we get a multiplication of a phase that is a linear combination of a phase in the dimension and a phase in the dimension:

2.c now we initialized a rectangle instead of the square, at the center of the matrix. We displayed the image and its 2D-FFT:



Now, we get a similar result to a but the amplitude plot has changed. This is because, as we know, scaling a sequence in a factor of scales it's fourier transform in a factor of , more intuitively, stretching in the time domain results in compressing in the frequency domain

So, since we the squeezed square on the horizontal axis, we see a stretch in the FFT amplitude on the horizontal axis. And since we stretched it on the vertical axis, the FFT amplitude is compressed on the vertical axis

2.d yes we can! Notice that

Which is the product of two waves:

Which is the

let's read the beatles.png image, convert it to grayscale, and normalize to 

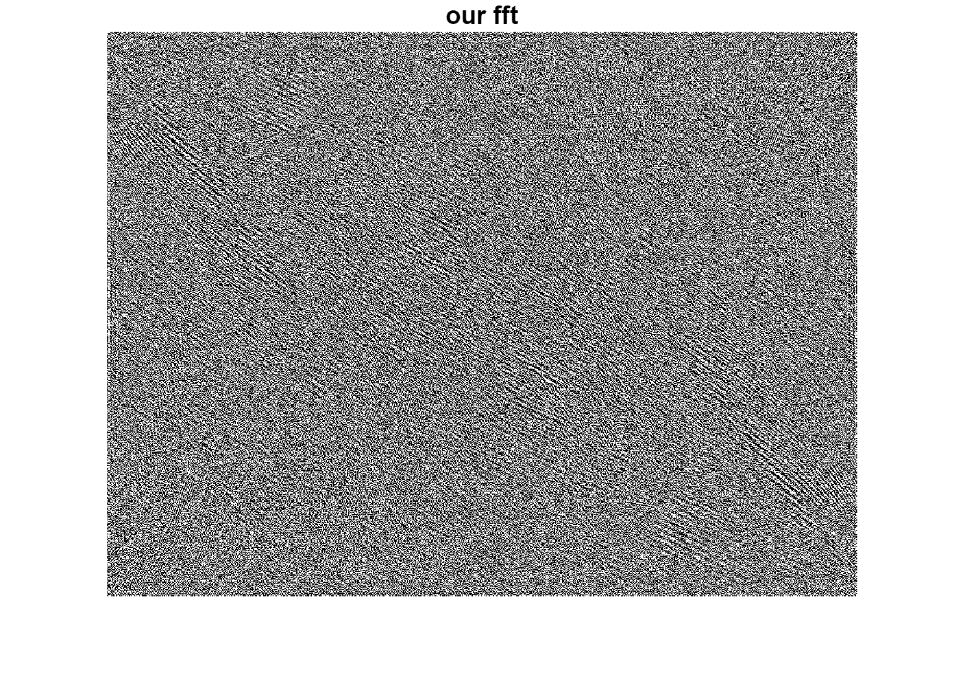
I = imread\_normalized("beatles.png");

% 4

my\_FFT = dip\_fft2(I);

imshow(real(my\_FFT))

title('our fft')

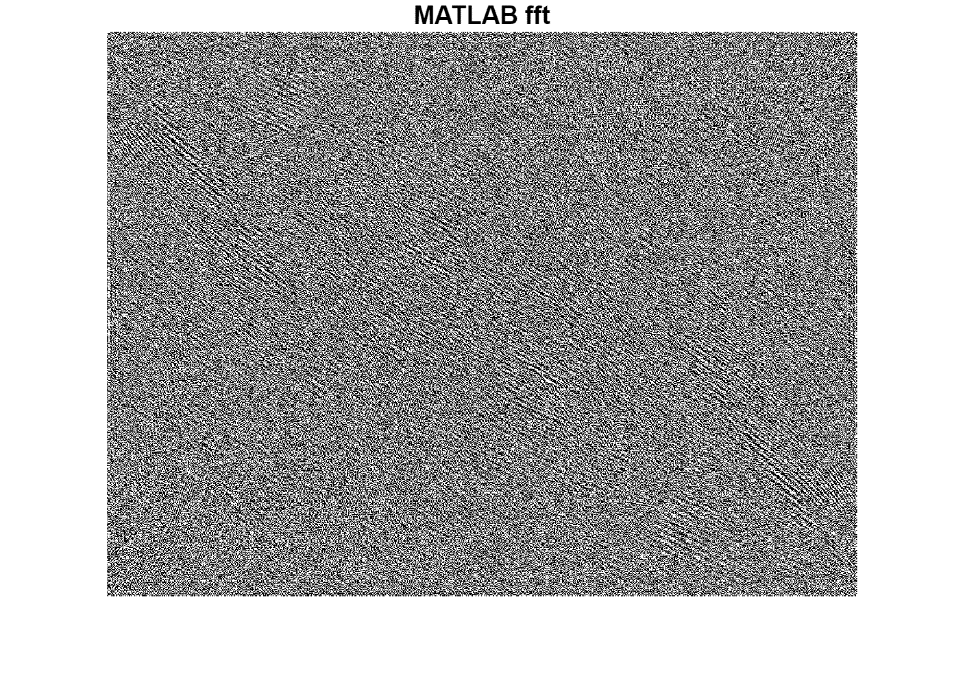


F = fft2(I);

let's just check how our algorithm compares to MATLAB's implementation:

imshow(real(F))

title('MATLAB fft')



imshow(F-my\_FFT);

Warning: Displaying real part of complex input.

title('the difference between our implementation and MATLAB built in')



## Reconstruct the original image

by using your inverse-FFT function. Is it identical to the original image? Note that the output of the iFFT are complex numbers - you should display only the real part of the image using imshow(real(-))

reconstructed\_img = dip\_ifft2(my\_FFT);

imshow(real(reconstructed\_img))

title('reconstructed image')



but are they identical? to answer that we will compute the error, to estimate the loss to the original image by applying 

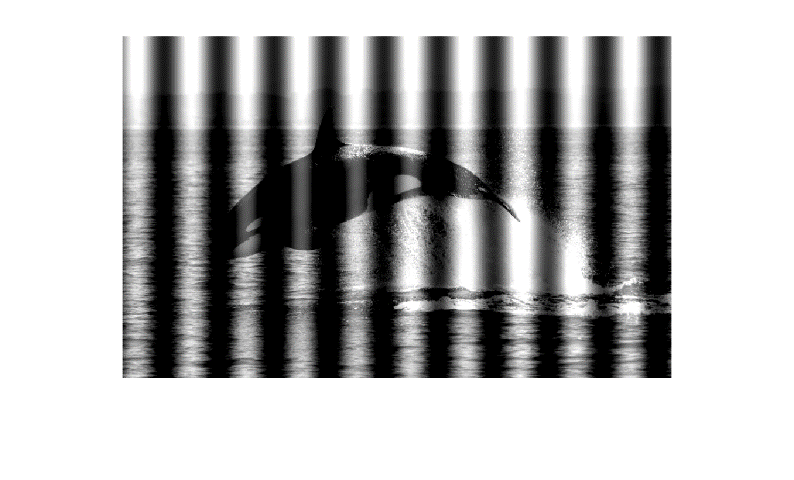
mse = mean(mean((reconstructed\_img - I) .^ 2))

mse = 2.5708e-28 - 5.5161e-29i

1.2 Transformation properties

load("freewilly.mat");

imshow(freewilly)



## b. find the frequency of the bars

first, we plotted the first row, which seems to contain the least of the original image information.

row1 = freewilly(1,:)

row1 = *1×640*

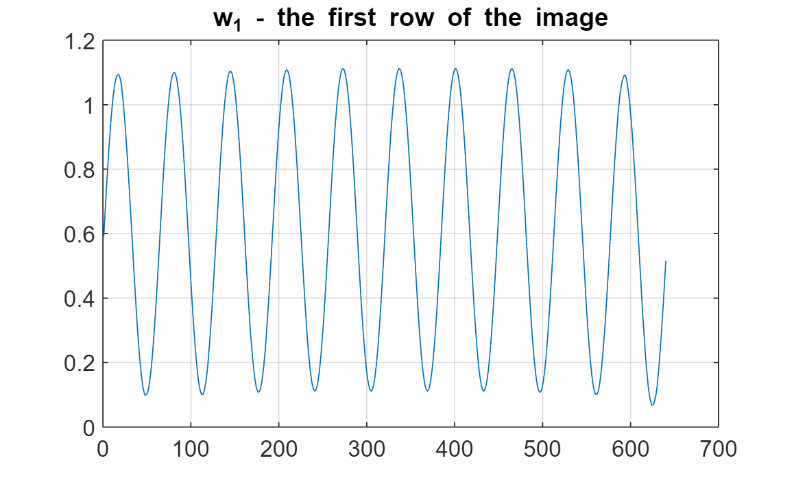
0.5922 0.6412 0.6897 0.7373 0.7835 0.8279 0.8699 ⋯

plot(row1);

F = fft(row1);

title('w\_1 - the first row of the image')

grid on



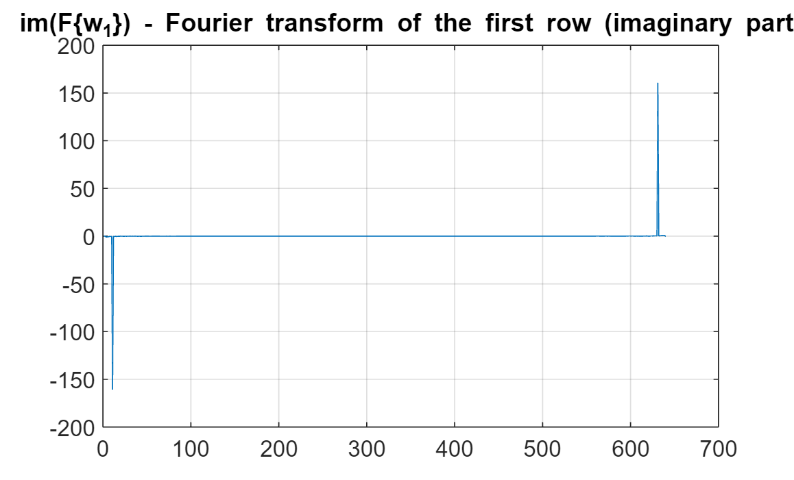
by plotting it, it seems clear that  and that we can already estimate  by counting peaks.

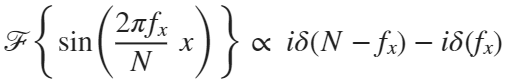
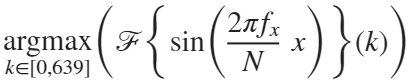
however, for more accurate and scientific assesment, we chose to go further and calculate it's fourier transform

plot(imag(F));

title('im(F\{w\_1\}) - Fourier transform of the first row (imaginary part)')

grid on



by plotting the imaginary part we can use the descrete transform identity  . so we can find  by taking the 

of course, let's not forget to add one, to account for MATLAB's indexing.

[argvalue, argmin] = min(imag(F))

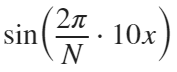
argvalue = -160.4060

argmin = 11

[argvalue, argmax] = max(imag(F))

argvalue = 160.4060

argmax = 631

we get that  and so the sinusoidal bars follow the equation of . it makes a lot of sense that this is a good estimation, because there are roughly 10 bars in total of N pixels.

## c. creating the image of the prison bars

we will use the equation that we have to create the pattern along the  axis. we will repeat it along the axis to recieve the image of the bars.

[M, N] = size(freewilly);

bars = 0.5\*sin(2\*(pi/N)\*10\*(1:N));

bars = repmat(bars,M,1);

imshow(bars)



## d.Compute the 2D-FFT of the prison bars image

let's see how this is represented in the frequancy domaimn:

bars\_fft = fft2(bars);

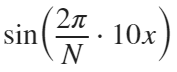
imshow(bars\_fft)

Warning: Displaying real part of complex input.

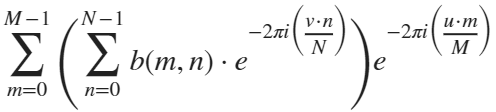


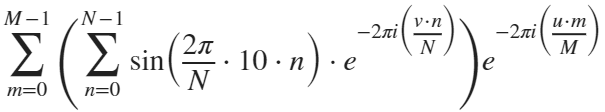
it makes sense: what we see is a blank image (zeros everywhere) and two non zero amplitudes, at the pixels corresponding to the frequencies we discovered in section (b), given by the two  functions given by transforming the sine waves on the  axis.

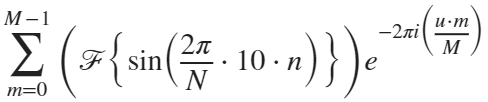
our prison bars are represented by

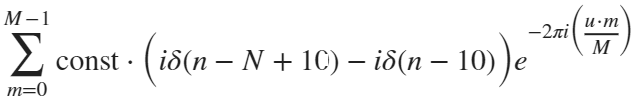


so the fourier transform

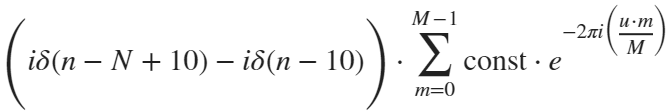








but this depends only in 







on the  axis on the other hand, the bars image is constant, which results in a delta function at the 0 index (1 in MATLAB). so, we see our two pulses on the 10'th and negative 10'th columns are activated only on the first row, beacause of the multiplication with 

## e. freeing willy

due to the linearity of the fourier transform, given by the identity  we might be able to free willy by subtracting the from the spectral domain of the original image:

willy\_fft = fft2(freewilly);

willy\_fft = willy\_fft - bars\_fft;

willy\_free = ifft2(willy\_fft);

imshow(willy\_free)



we maneged to drastically decrease the bars intensity though it didn't work perfectly. we can try erasing the bars frequency alltogether by eliminating the 2 delta functions that represent their transform

willy\_fft = fft2(freewilly);

willy\_fft(abs(bars\_fft)>1) = 0;

willy\_free = ifft2(willy\_fft);

imshow(willy\_free)



2. Scaling, translation and seperability

(a) we Initialized a 128×128 all-zeros matrix. At the center of it, we placed a 40×40 all-ones square, and displayed the image and its 2D-FFT:

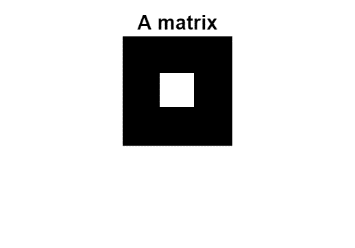
A = zeros(128,128);

A(44:83,44:83) = 1;

figure;

imshow(A);

title('A matrix');



FA = fft2(A);

figure;

subplot(1, 3, 1);

imshow(abs(FA));

title('|F\{A\}|')

subplot(1, 3, 2);

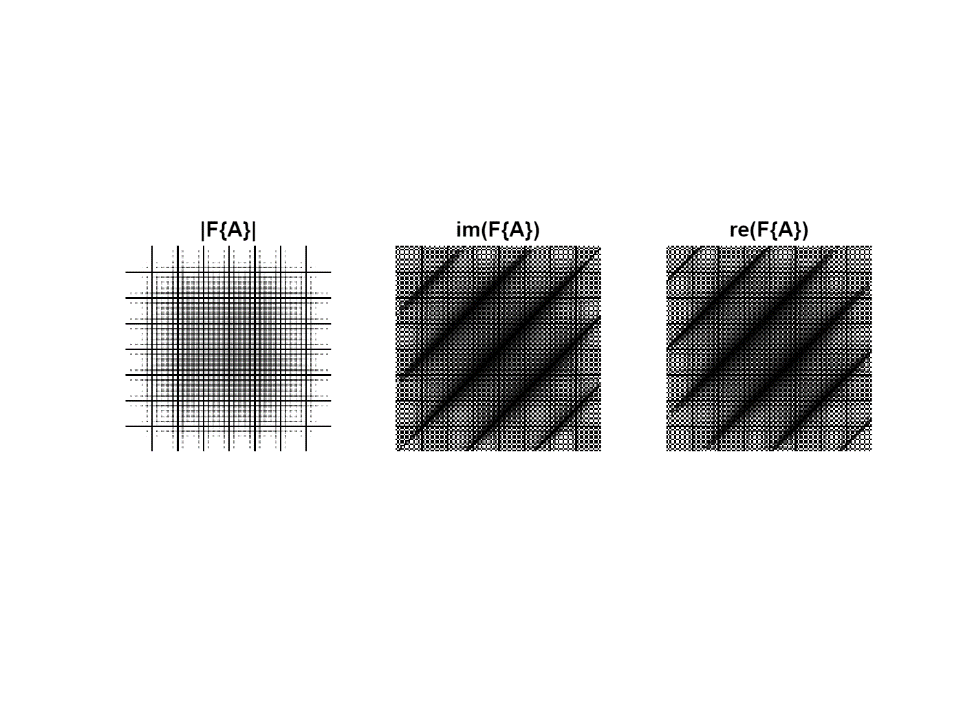
imshow(imag(FA));

title('im(F\{A\})')

subplot(1, 3, 3);

imshow(real(FA));

title('re(F\{A\})')

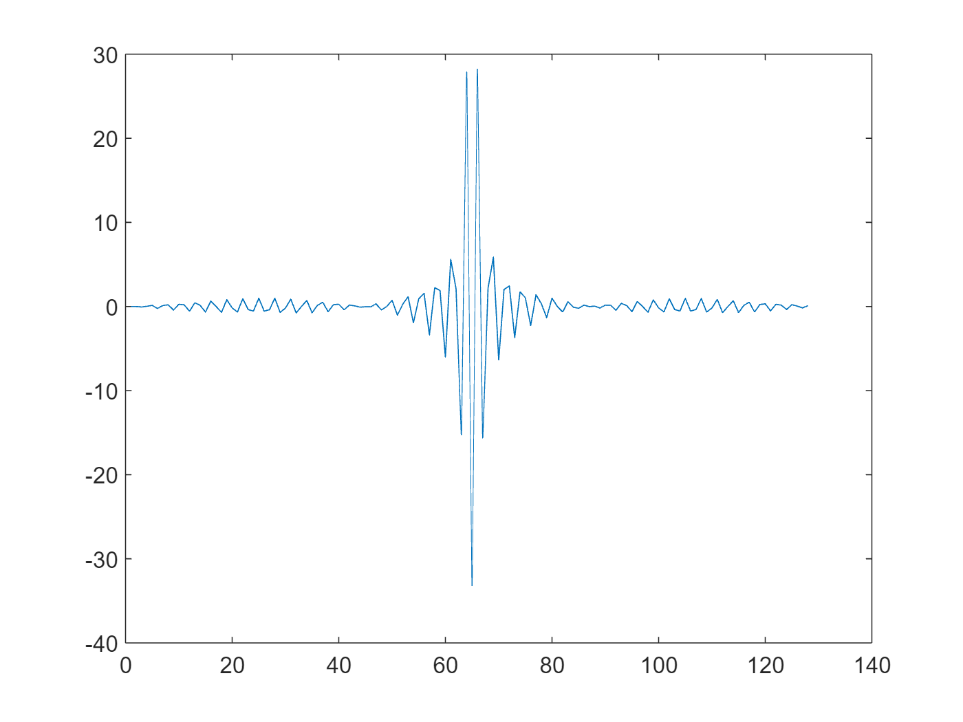


why does it look the way it does?

figure;

middlerow = imag(circshift(FA(64,:),64));

plot(middlerow);



(b) now we put the square at the bottom right,

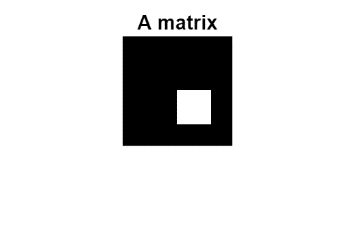
A = zeros(128,128);

A(64:103,64:103) = 1;

figure;

imshow(A);

title('A matrix');



FA = fft2(A);

figure;

subplot(1, 3, 1);

imshow(abs(FA));

title('|F\{A\}|')

subplot(1, 3, 2);

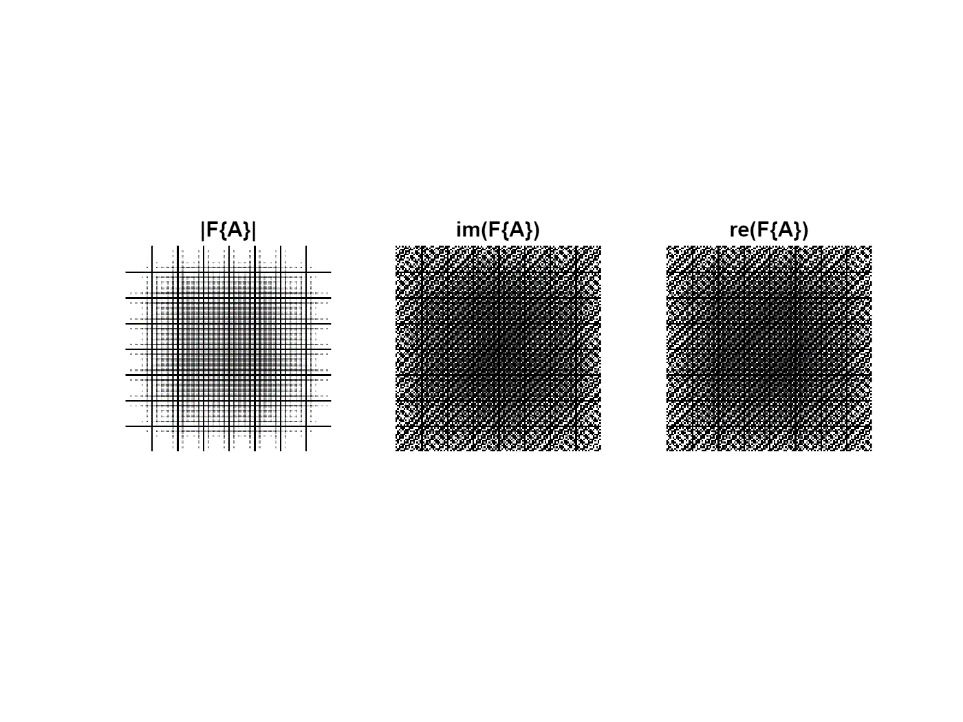
imshow(imag(FA));

title('im(F\{A\})')

subplot(1, 3, 3);

imshow(real(FA));

title('re(F\{A\})')



(c) now we place a rectangle at the center,

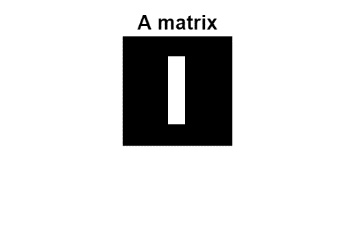
A = zeros(128,128);

A(24:103,54:73) = 1;

figure;

imshow(A);

title('A matrix');



FA = fft2(A);

figure;

subplot(1, 3, 1);

imshow(abs(FA));

title('|F\{A\}|')

subplot(1, 3, 2);

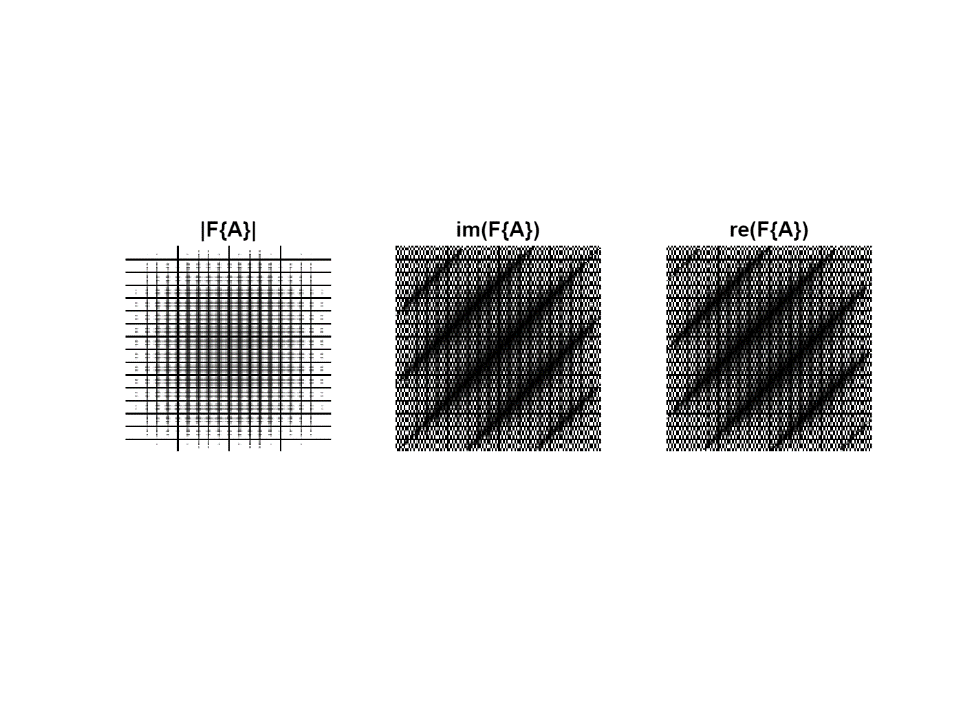
imshow(imag(FA));

title('im(F\{A\})')

subplot(1, 3, 3);

imshow(real(FA));

title('re(F\{A\})')



figure;

middlerow = abs(FA(64,:));

plot(middlerow);

