close all

clc

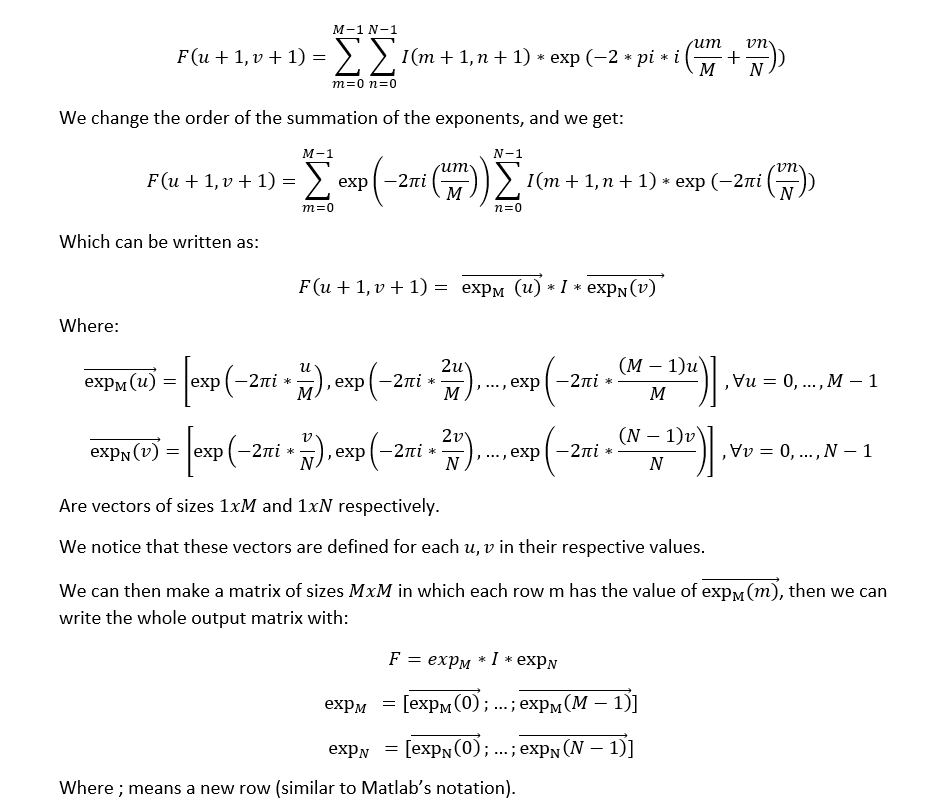
disp('Elias Assaf 315284729 - Jameel Nassar 206985152')

Elias Assaf 315284729 - Jameel Nassar 206985152

# Section 1:

if we look closely at the equation for the fourier transform, we notice that we sum over all m and all n, we can change the order of the summation a little, and give it a matrix form instead of doing it in foor loops, which would take O(M\*N) calculations for each cell, and there is M\*N cells.

we derive the matrix equations for the fft:



similarly for ifft.

the fftshift is simply dividing the matrix into 4 quadrants and swapping quadrant 1 and 4, 2 and 3.

note that for this to work N,M need to be even numbers.

# Section 1.1:

beatles = imread("beatles.png");

beatles\_norm = double(rgb2gray(beatles))/255;

beatles\_fft2 = dip\_fft2(beatles\_norm);

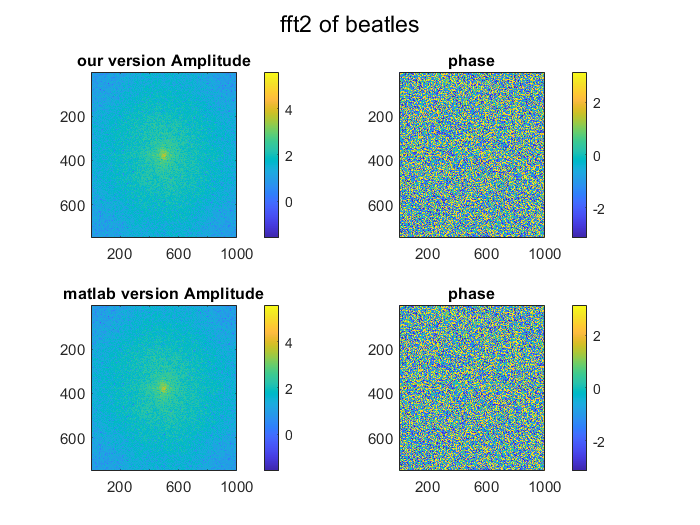
beatles\_fft2\_matlab = fft2(beatles\_norm);

figure;sgtitle("fft2 of beatles")

subplot(2,2,1);imagesc(log10(abs(dip\_fftshift(beatles\_fft2))));title("our version Amplitude");colorbar;

subplot(2,2,2);imagesc(angle(dip\_fftshift(beatles\_fft2)));title("phase");colorbar;

subplot(2,2,3);imagesc(log10(abs(dip\_fftshift(beatles\_fft2\_matlab))));title("matlab version Amplitude");colorbar;

subplot(2,2,4);imagesc(angle(dip\_fftshift(beatles\_fft2\_matlab)));title("phase");colorbar;

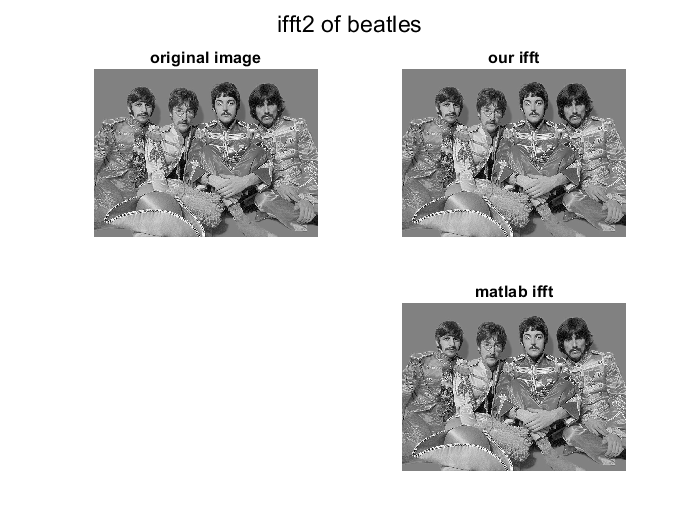
beatles\_ifft2 = dip\_ifft2(beatles\_fft2);

beatles\_ifft2\_matlab = ifft2(beatles\_fft2\_matlab);

figure;sgtitle("ifft2 of beatles")

subplot(2,2,1);imshow(beatles\_norm);title("original image");

subplot(2,2,2);imshow(real(beatles\_ifft2));title("our ifft");

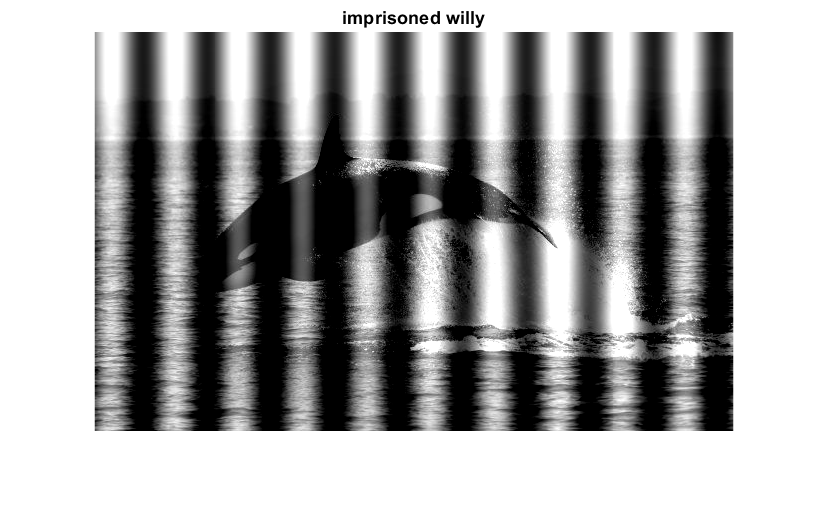
subplot(2,2,4);imshow(real(beatles\_ifft2\_matlab));title("matlab ifft");

in the next section we want to free willy.

# Section 1.2.1:

load("freewilly.mat");

imprisoned\_willy = freewilly;

figure;imshow(imprisoned\_willy);title("imprisoned willy");

we take a mental exercise where we imagine a sin function on the x axis with the parameters:

sin(2\*pi\*f\_x/N \* x), x=0,...,N-1

for f\_x=1 we only get 1, for f\_x=2 we get 2 cycles and so on...

we count the number of cycles of the black and white bars in the picture and notice the very nice fact that they are a nice even integer number of value 10, which is the f\_x we are looking for.

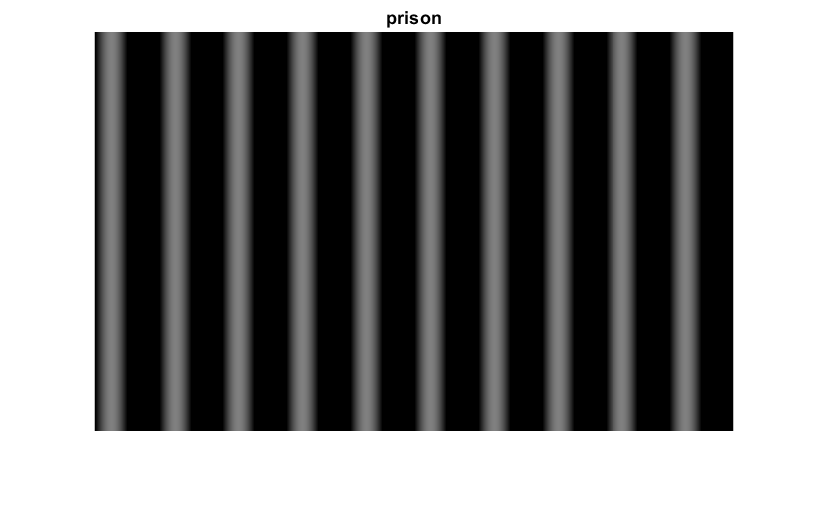
[M,N] = size(imprisoned\_willy);

[X\_grid, Y\_grid] = meshgrid(0:N-1, 0:M-1);

f\_x = 10;

prison = 0.5 \* sin(2\*pi\*f\_x/N \* X\_grid);

figure;imshow(prison);title("prison")



after building the prison ourselves and using fft on it, we expect to have a "lambda" function on the frequency domain where the peaks exist on ± f\_x.

and since we have no signal in the y axis (or DC signal with value 0) we expect its fft to be 0 at all the frequencies.

prison\_fft = fft2(prison);

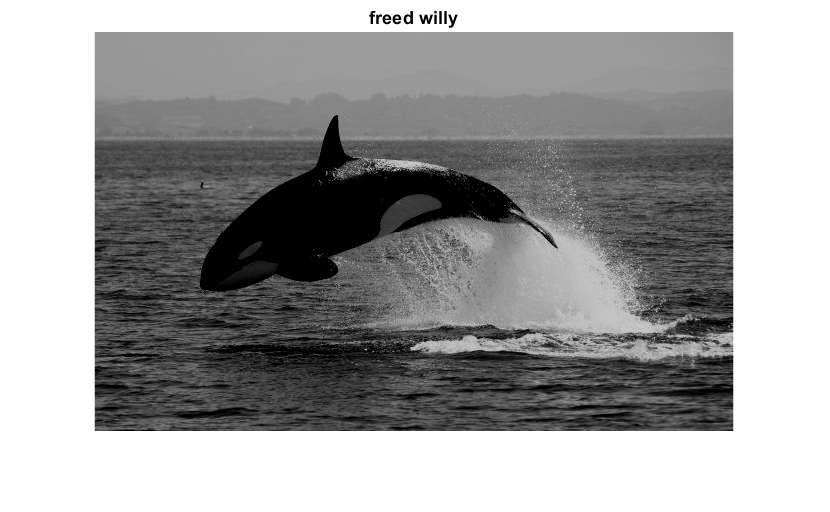
figure;imshow(abs(dip\_fftshift(prison\_fft)));title("prison fft2")

Shape

Description automatically generated

to free william (how he likes to be called after being in prison for so long) we go to the fourier domain and zero out the peaks of the prison bars, i.e at ± f\_x.

because we dont want to waste computation time, we don't use fftshift first, rather we go to y=1 and to the points ***x\_1=1+f\_x,*** ***x\_2 = 1+end-f\_x*** and zero them out, then we use the ifft to back to space domain and were done.

Free\_Willy(imprisoned\_willy);

yay

# Section 1.2.2

square\_1 = zeros(128, 128);

[square\_1\_X, square\_1\_y] = meshgrid(44:83, 44:83);

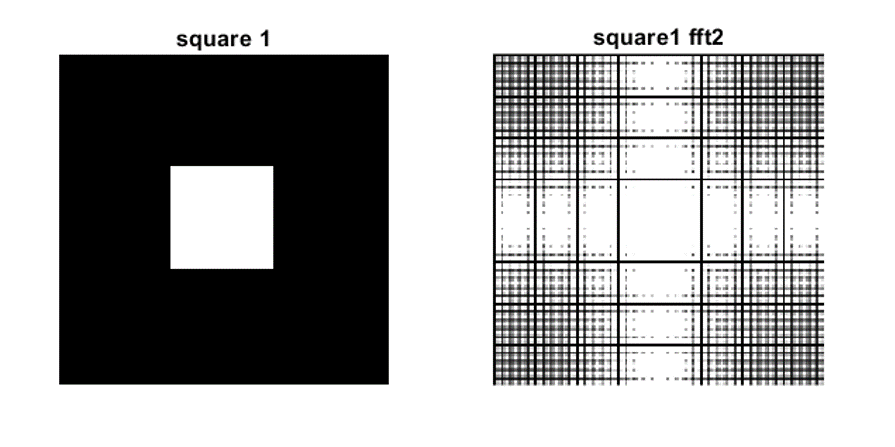
idx\_1 = sub2ind(size(square\_1), square\_1\_X, square\_1\_y);

square\_1(idx\_1) = 1;

square\_1\_fft = fft2(square\_1);

figure;

subplot(1,2,1);imshow(square\_1);title("square 1")

subplot(1,2,2);imshow(abs(dip\_fftshift(square\_1\_fft)));title("square1 fft2")

square\_2 = zeros(128, 128);

[square\_2\_X, square\_2\_y] = meshgrid(64:103, 64:103);

idx\_2 = sub2ind(size(square\_2), square\_2\_X, square\_2\_y);

square\_2(idx\_2) = 1;

square\_2\_fft = fft2(square\_2);

figure;

subplot(1,2,1);imshow(square\_2);title("square 2");

subplot(1,2,2);imshow(abs(dip\_fftshift(square\_2\_fft)));title("square 2 fft2");

A picture containing graphical user interface

Description automatically generated

figure;

subplot(1,2,1);imshow(abs(dip\_fftshift(square\_1\_fft)));title("square1 fft2")

subplot(1,2,2);imshow(abs(dip\_fftshift(square\_2\_fft)));title("square2 fft2");

A picture containing text, shoji, crossword puzzle

Description automatically generated

sum(square\_1\_fft == square\_2\_fft, 'all')

ans = 1788

we notice that most of the values are not the same, that is because when we move the square we change its phase, so they get multipled by sin/cos of the phase difference.

we base our answer on the fact that shifting in time equals multiplying by an exponent of the shifter value in frequency domain.



similarly in 2d transforms.

square\_3 = zeros(128, 128);

[square\_3\_X, square\_3\_y] = meshgrid(24:103, 54:73);

idx\_3 = sub2ind(size(square\_3), square\_3\_X, square\_3\_y);

square\_3(idx\_3) = 1;

square\_3\_fft = fft2(square\_3);

figure;

subplot(1,2,1);imshow(square\_3);title("square 3");

subplot(1,2,2);imshow(abs(dip\_fftshift(square\_3\_fft)));title("square 3 fft2");

A picture containing diagram

Description automatically generated

figure;

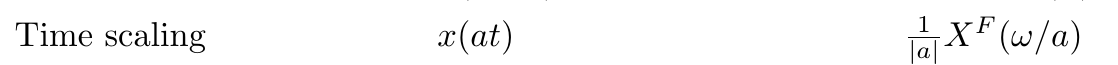
subplot(1,3,1);imshow(abs(dip\_fftshift(square\_1\_fft)));title("square1 fft2")

subplot(1,3,2);imshow(abs(dip\_fftshift(square\_2\_fft)));title("square2 fft2");

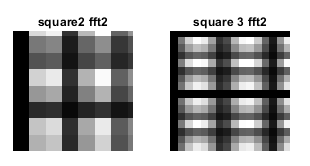
subplot(1,3,3);imshow(abs(dip\_fftshift(square\_3\_fft)));title("square 3 fft2");

A picture containing text, shoji, crossword puzzle, cage

Description automatically generated

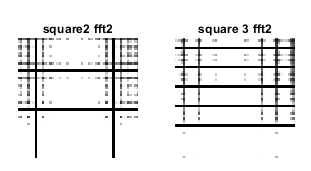


we remember the time scaling property of the fourier transform, in the x axis a=2, so in the frequency domain we can see the x axis expanded, i.e the little squares became bigger.



bottom left corner for example.

similar thing on the y axis but we have a=1/2, so in the frequency domain the y axis shrunk



upper middle section for example.

we can represent the 80x20 square by the 2 vectors

[1;...;1] \* [1,...,1], the first begin of size 80x1 and the second of size 1x20.

we pad with zeros to get the correct length of 128x128

vec\_80\_1 = [zeros(24,1); ones(80,1); zeros(24,1)];

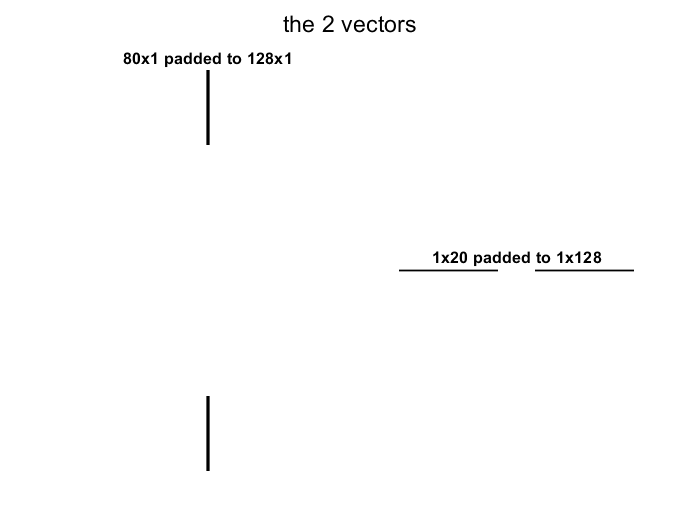
vec\_1\_20 = [zeros(1,54), ones(1,20), zeros(1,54)];

square\_80\_20 = vec\_80\_1 \* vec\_1\_20;

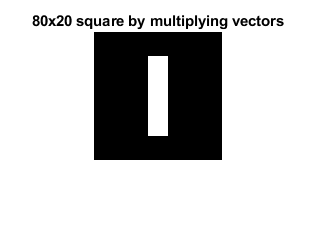
figure;sgtitle("the 2 vectors");

subplot(1,2,1);imshow(vec\_80\_1);title('80x1 padded to 128x1');

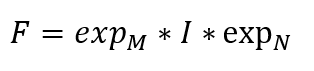
subplot(1,2,2);imshow(vec\_1\_20);title('1x20 padded to 1x128');



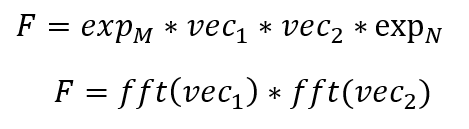
figure;imshow(square\_80\_20);title("80x20 square by multiplying vectors");



as we saw in the first section we got:



where I is our image, we can simply write it in its vector form and we get:



we write our function accordingly.

square\_3\_sep\_fft = sep\_fft2(vec\_80\_1, vec\_1\_20);

figure;

subplot(1,2,1);imshow(abs(dip\_fftshift(square\_3\_fft)));title("square3 fft2")

A picture containing text, shoji

Description automatically generatedsubplot(1,2,2);imshow(abs(dip\_fftshift(square\_3\_sep\_fft)));title("square3 fft2 by separating");

as we can see they are identical !!!!!.

# Section 2

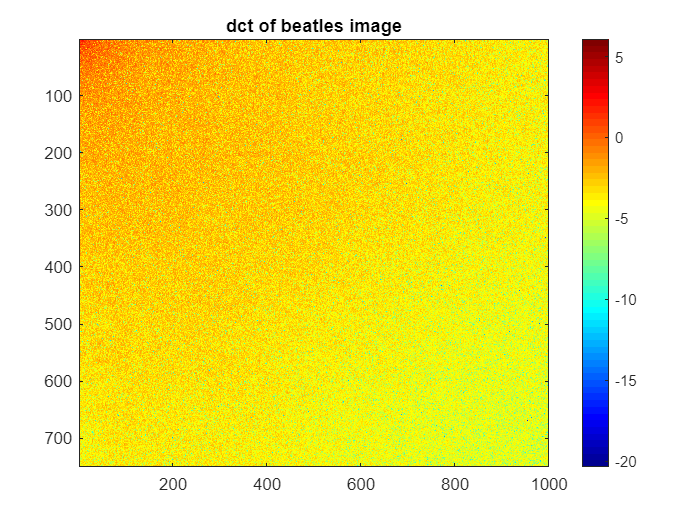
beatles = imread("beatles.png");

beatles\_norm = double(rgb2gray(beatles))/255;

beatles\_dct2 = dct2(beatles\_norm);

[M\_beatles, N\_beatles] = size(beatles\_norm);

figure;imagesc(log(abs(beatles\_dct2)));colormap(jet(64));colorbar;title('dct of beatles image')



to remove half of the values randomly, we take a random permutation of the number of elements in the dct domain, take half of them, and zeros out those indexes in the dct matrix.

half\_ones\_rand = randperm(numel(beatles\_dct2));

half\_ones\_rand = half\_ones\_rand(1:end/2);

beatles\_dct2\_half\_rand = beatles\_dct2;

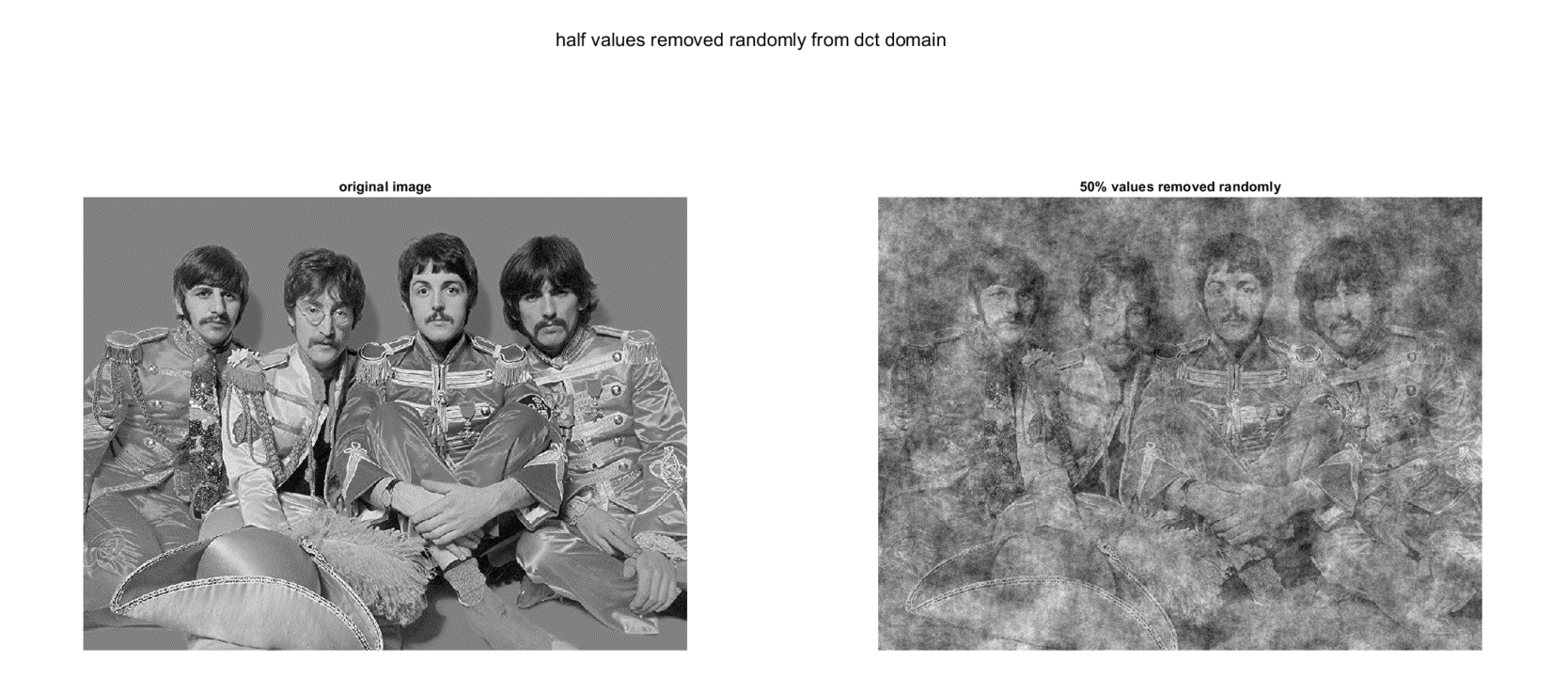
beatles\_dct2\_half\_rand(half\_ones\_rand) = 0;

beatles\_idct2\_half\_rand = idct2(beatles\_dct2\_half\_rand);

figure;sgtitle("half values removed randomly from dct domain");

subplot(1,2,1);imshow(beatles\_norm);title("original image");

subplot(1,2,2);imshow(beatles\_idct2\_half\_rand);title("50% values removed randomly");



perc = sum(beatles\_dct2\_half\_rand == 0, 'all') / numel(beatles\_dct2)

perc = 0.5000

for the next section, we need to zero out the half lower absolute values of the dct, for that we remember the definition of median, the value which separates the higher half from the lower half of the data, which means that we can zero out all the values smaller than the median value .

notice that here we only deal in absolutes.

beatles\_dct2\_half\_lowest = beatles\_dct2;

beatles\_dct2\_abs\_median = median(abs(beatles\_dct2\_half\_lowest), 'all')

beatles\_dct2\_abs\_median = 0.0365

beatles\_dct2\_half\_lowest(beatles\_dct2\_half\_lowest < beatles\_dct2\_abs\_median & beatles\_dct2\_half\_lowest > -beatles\_dct2\_abs\_median) = 0;

beatles\_idct2\_half\_lowest = idct2(beatles\_dct2\_half\_lowest);

figure;sgtitle("half absolute lowest values removed from dct domain (values lower than the median)");

subplot(1,2,1);imshow(beatles\_norm);title("original image")

subplot(1,2,2);imshow(beatles\_idct2\_half\_lowest);title("50% lowest values removed")

A group of men sitting together

Description automatically generated with medium confidence

for the next section we test which some values of a where we remove values from the dct domain which thier absolute value is smaller than a.

figure;sgtitle("values from (-a,a) removed from dct domain");

subplot(3,2,1);imshow(beatles\_norm);title('original image');

i=2;

for a = [0.02 0.05 0.07 0.09 0.15]

beatles\_dct2\_no\_a = beatles\_dct2;

beatles\_dct2\_no\_a(beatles\_dct2\_no\_a < a & beatles\_dct2\_no\_a > -a) = 0;

beatles\_idct2\_no\_a = idct2(beatles\_dct2\_no\_a);

perc = sum(beatles\_dct2\_no\_a == 0, 'all') / numel(beatles\_dct2);

subplot(3,2,i);imshow(beatles\_idct2\_no\_a);title(['a = ' num2str(a) ', percentage = ' num2str(perc)])

i = i+1;

end

A picture containing text, posing, newspaper, old

Description automatically generated

we choose a=0.05 to be the best value, it removes 60% of the values, and gives a good reconstruction of the photo without too much noise.

as we saw in class, most of the information exists in the higher values of the dct domain, so removing the about 50-70% of values still gives a reconstruction which we can understand and sometimes not even notice that it was compressed.

but for example when we randomly removed half of the values, we removed some of the important values and we lost most of the information of the image.

# Section 3

beetle = imread("beetle.jpg");

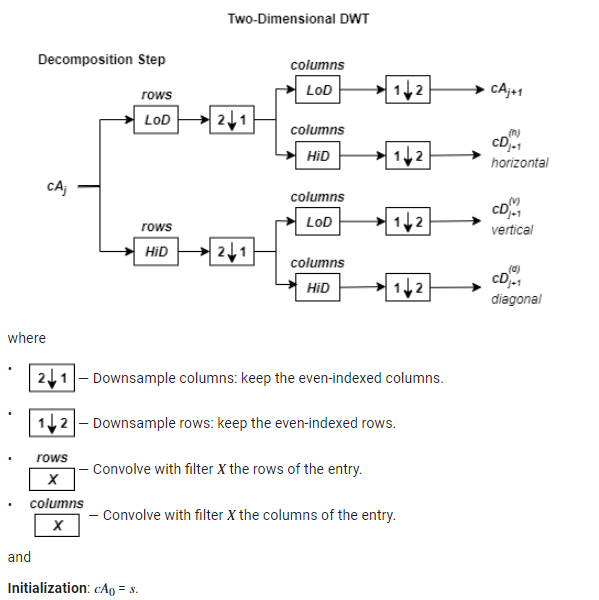
beetle\_norm = double(rgb2gray(beetle))/255;

n=5;

[c\_beetle,s\_beetle]=wavedec2(beetle\_norm,n,'haar');

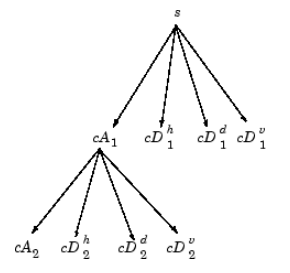
In this section we chose 5 decomposition levels.

The chart describes the basic decomposition step for the image:



LoD is the lowpass decomposition filter, and HiD is the highpass decomposition filter.

So, for *J* = 2, the two-dimensional wavelet tree has the form



As we can see in the graph with each level increase we lose decrease the x and y pixel density by half which means we get quarter resolution of the original image, which also means that we lose some detail, we can can confirm this if we look at the x and y axis sizes of the different decomposition levels

each detail coefficients is shows the changes in pixels of sertain axis according to the chosen filter in the decomposition, and we can see that in the fiures below.

* *A* contains the approximation coefficients.
* *H* contains the horizontal detail coefficients.
* *V* contains the vertical detail coefficients.
* *D* contains the diagonal detail coefficients.

for i = 1:n

[h\_beetle,v\_beetle,detail\_beetle] = detcoef2('all',c\_beetle,s\_beetle,i);

cfs2 = appcoef2(c\_beetle,s\_beetle,'haar',i);

V\_img = wcodemat(v\_beetle,255,'mat',i);

H\_img = wcodemat(h\_beetle,255,'mat',i);

D\_img = wcodemat(detail\_beetle,255,'mat',i);

A\_img = wcodemat(cfs2,255,'mat',i);

figure;colormap('gray');

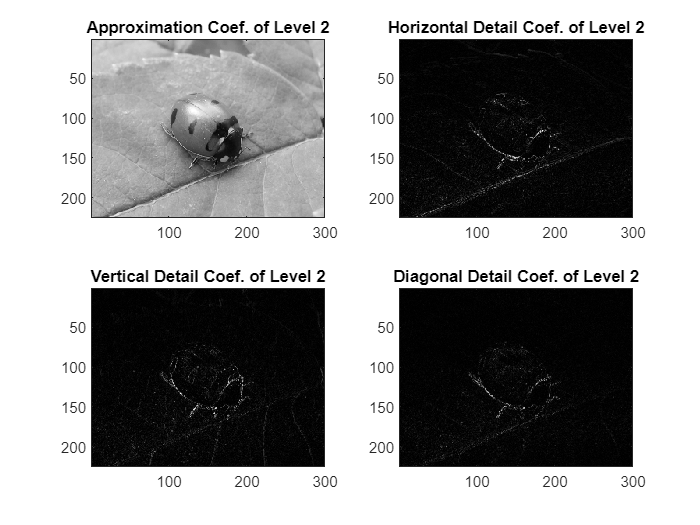
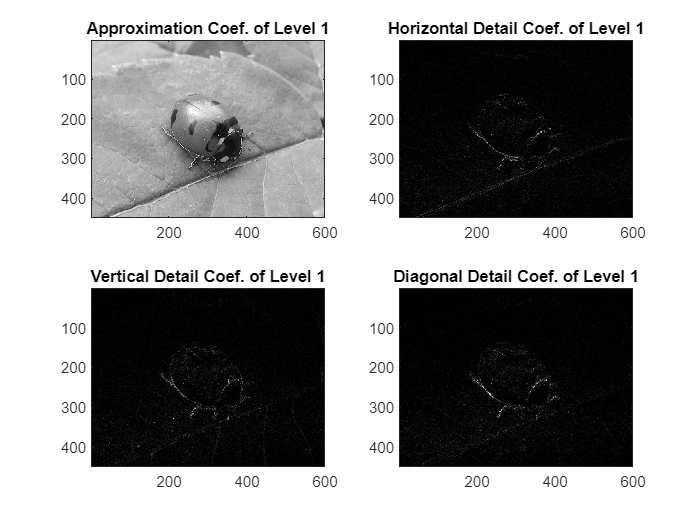
subplot(2,2,1);imagesc(A\_img);title(['Approximation Coef. of Level ' int2str(i)]);

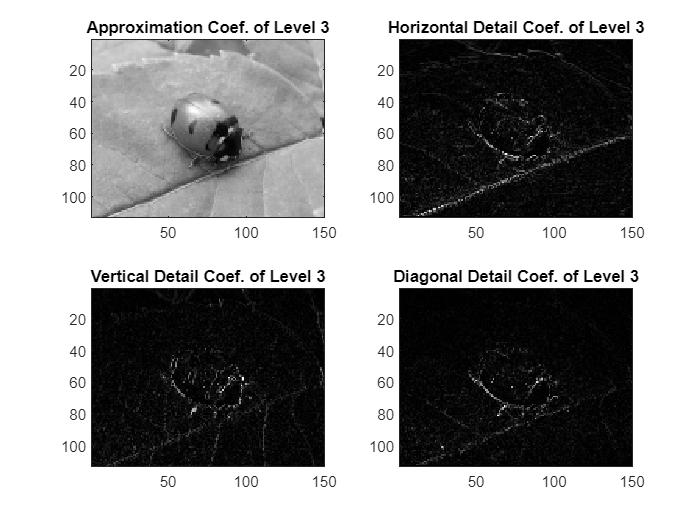
subplot(2,2,2);imagesc(H\_img);title(['Horizontal Detail Coef. of Level ' int2str(i)]);

subplot(2,2,3);imagesc(V\_img);title(['Vertical Detail Coef. of Level ' int2str(i)]);

subplot(2,2,4);imagesc(D\_img);title(['Diagonal Detail Coef. of Level ' int2str(i)]);

end





# 

# 

# functions

function [img\_fft2] = dip\_fft2(I)

[M,N] = size(I);

M\_vec = (0:M-1);

N\_vec = (0:N-1);

M\_mat = exp(-2\*pi\*1i \* (M\_vec.' \* M\_vec)/M);

N\_mat = exp(-2\*pi\*1i \* (N\_vec.' \* N\_vec)/N);

img\_fft2 = M\_mat \* I \* N\_mat;

end

function [img\_ifft2] = dip\_ifft2(F)

[M,N] = size(F);

M\_vec = (0:M-1);

N\_vec = (0:N-1);

M\_mat = exp(2\*pi\*1i \* M\_vec.' \* M\_vec/M);

N\_mat = exp(2\*pi\*1i \* N\_vec.' \* N\_vec/N);

img\_ifft2 = M\_mat \* F \* N\_mat;

img\_ifft2 = img\_ifft2/(M\*N);

end

function [shifted\_fft2] = dip\_fftshift(F)

[M,N] = size(F);

quad\_1 = F(1:M/2,1:N/2);

quad\_2 = F(M/2+1:end, 1:N/2);

quad\_3 = F(M/2+1:end, N/2+1:end);

quad\_4 = F(1:M/2, N/2+1:end);

shifted\_fft2 = [quad\_3 quad\_2; quad\_4 quad\_1];

end

function [freed\_willy] = Free\_Willy(Willy)

freed\_willy = fft2(Willy);

freed\_willy(1, 1+10) = 0;

freed\_willy(1, 1+end-10) = 0;

freed\_willy = real(ifft2(freed\_willy));

figure;imshow(freed\_willy);title("freed willy");

end

function [img\_fft2] = sep\_fft2(v1, v2)

img\_fft2 = fft(v1) \* fft(v2);

end