Alexandria University
Faculty of Engineering
Communication Department

Assignment: Lab 2
Digital Signal Processing

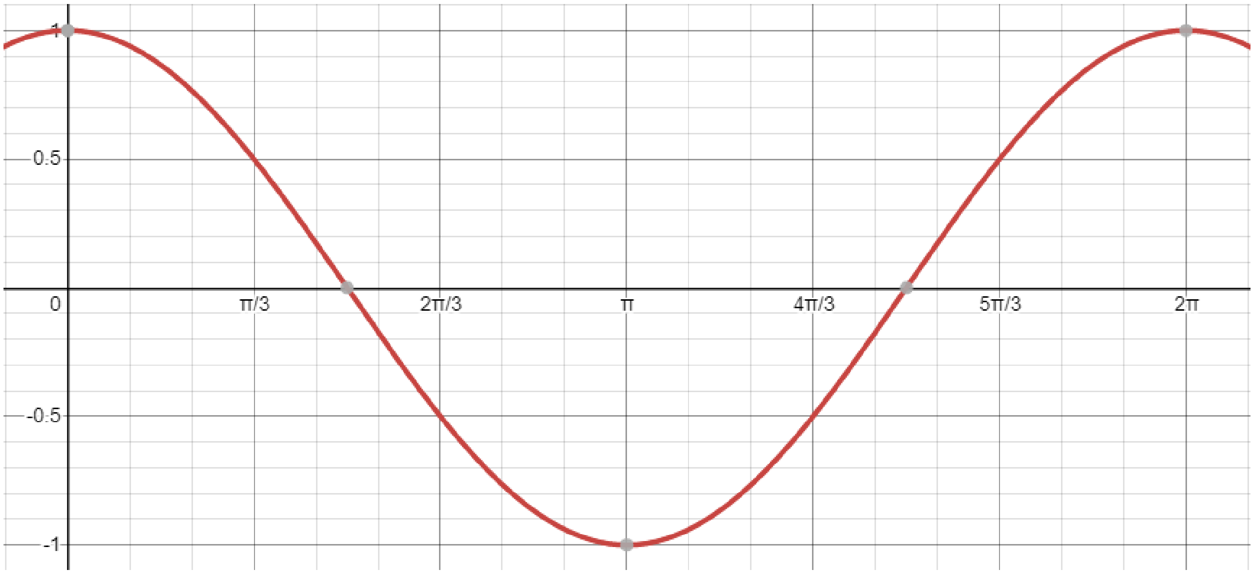
By:

18010198	احمد محمد احمد عبدالجواد
18010200	احمد محمد احمد ابوجبل
18010299	آسر محمد عاطف زايد
18011084	علي محمد طعيمة حمص
18011505	محمد طارق محمد عامر

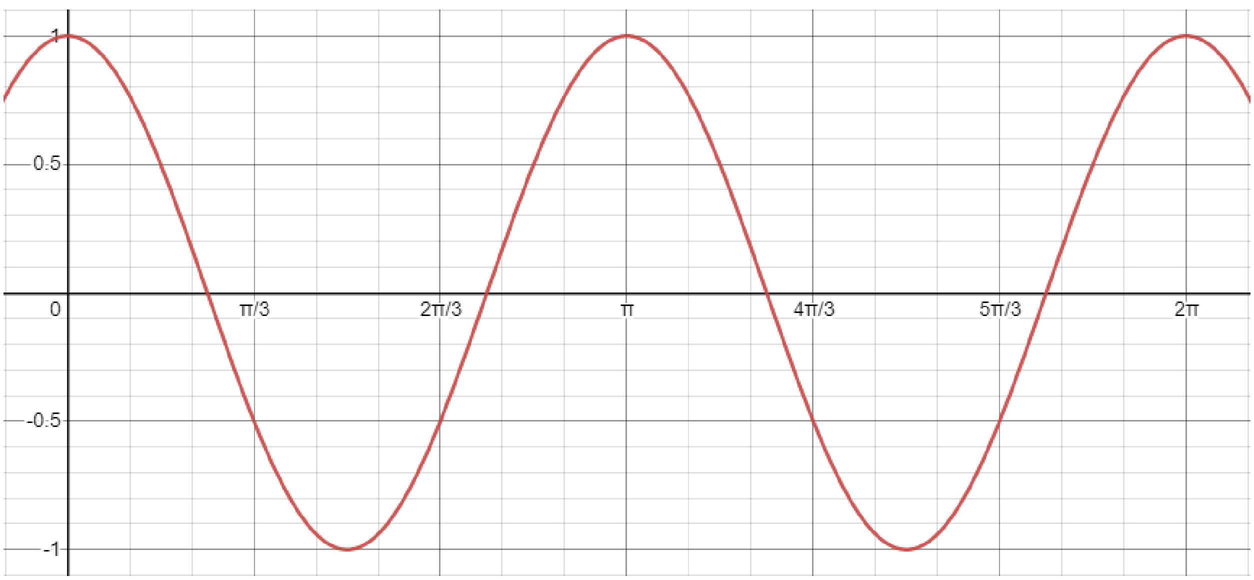
3rd year 1st semester

Introduction to DCT

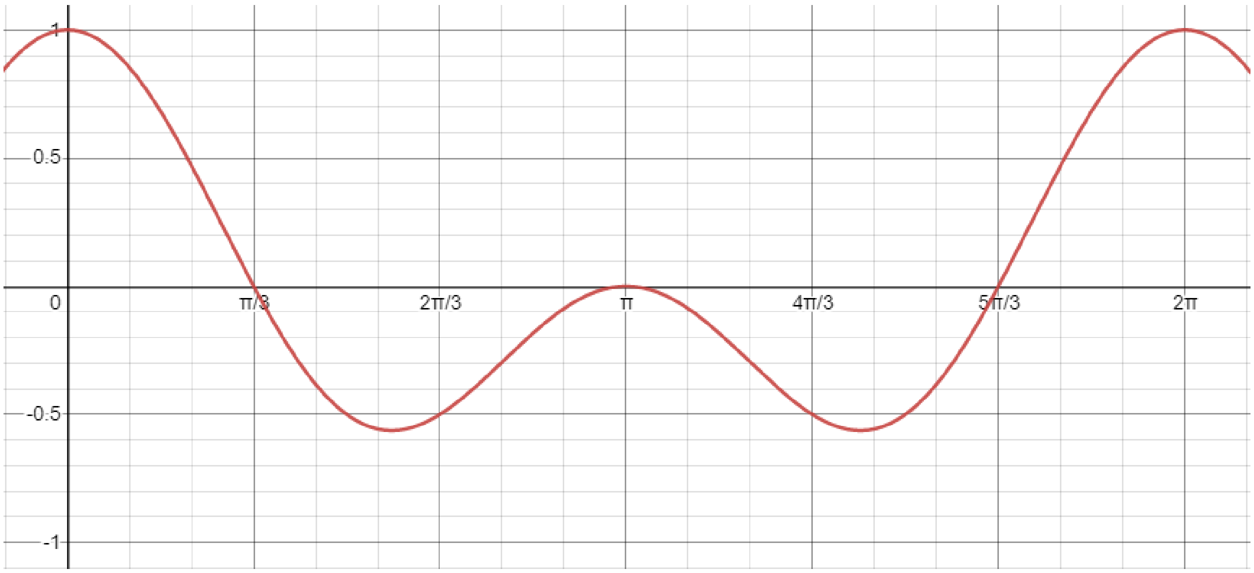
First we know that standard cos looks like this between -1 to 1



If we see another cos with higher frequency, it will look like that



If we took the average of the 2 cos waves above it will look like this



What are we trying to be to represent our image in terms of summation of a lot of cos wave.

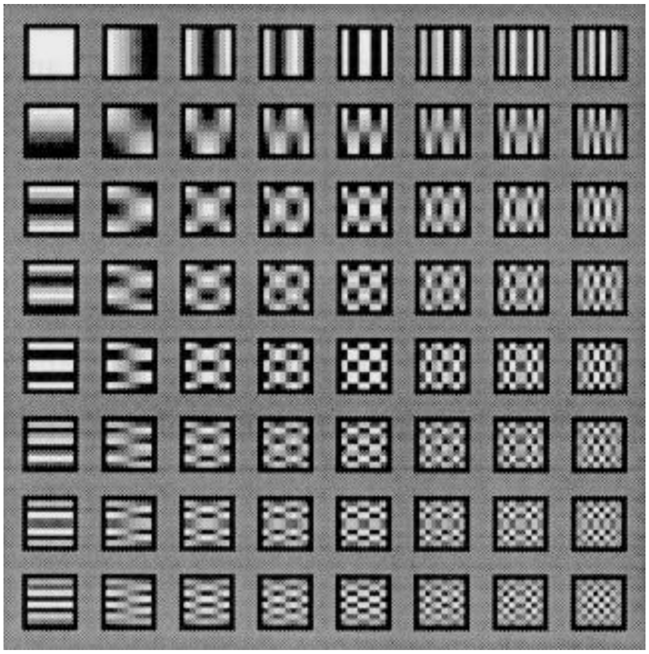
We want to remove the high frequencies of the image without distorting it.

If we have each signal with 8 components that means we need to have 8 cosines to represent it, and in JPEG we use 8x8 blocks of the image at a time.

So each 8x8 block can be represented by 8x8 cosines waves.

Now we need to calculate the DCT basis matrix that show the 8x8 cosines blocks while increasing its frequency

This will look like this



As we can see as we go down or right the frequencies of the block increase.

MATLAB function to calculate C8:

This is a function which takes the block size and gets CN

```
function CN = compute_CN(N)
    if N>1
        CN = zeros(N,N);
        CN(1,:) = sqrt(1/N);
        for i=2:N
            for j=1:N
                CN(i,j) = sqrt(2/N) * cos(pi*(i-1)*(j-1+0.5)/N);
            end
        end
    elseif N == 1
        CN = (1);
    else
        error('unvaild size, size must be 1 or more');
    end
end
```

We can call like this

```
C8 = compute_CN(8);
%check_transpose = transpose(C8);
%check_inv = inv(C8);
save C8.mat C8;
```

Output:

C8

8x8 double

	1	2	3	4	5	6	7	8	9
1	0.3536	0.3536	0.3536	0.3536	0.3536	0.3536	0.3536	0.3536	
2	0.4904	0.4157	0.2778	0.0975	-0.0975	-0.2778	-0.4157	-0.4904	
3	0.4619	0.1913	-0.1913	-0.4619	-0.4619	-0.1913	0.1913	0.4619	
4	0.4157	-0.0975	-0.4904	-0.2778	0.2778	0.4904	0.0975	-0.4157	
5	0.3536	-0.3536	-0.3536	0.3536	0.3536	-0.3536	-0.3536	0.3536	
6	0.2778	-0.4904	0.0975	0.4157	-0.4157	-0.0975	0.4904	-0.2778	
7	0.1913	-0.4619	0.4619	-0.1913	-0.1913	0.4619	-0.4619	0.1913	
8	0.0975	-0.2778	0.4157	-0.4904	0.4904	-0.4157	0.2778	-0.0975	
9									

JPEG encoding and decoding

First we need to pad the image which doesn't have number of rows or columns divisible by block size which is 8.

```
function padded_image = padding_by_block_size(gray_image,block_size)

    [rows, columns] = size(gray_image);

    %%%%%%%%%%%%%%%
    %%% padding %%%
    %%%%%%%%%%%%%%%

    % start from first row and skip blocksize pixels till reaching
    % the end of all row
    for row=1 :block_size:rows

        % start from first column and skip blocksize pixels till
        % reaching the end of all columns
        for column=1 :block_size:columns

            % at the end of the loops they will store the right size of the
            % padded image
            row_end = row + block_size -1;
            column_end = column + block_size -1;

            % if the right size excceds the actual size, pad the rest with
            % 0s
            if row_end > rows
                gray_image(rows+1:row_end,:) = 0;
            end

            if column_end > columns
                gray_image(:,columns+1:column_end) = 0;
            end

        end
    end

    padded_image = gray_image;
end
```

Then we need to split the image to blocks of block size (8)

```
function blocked_image = spliiting_image(image,block_size)

    [rows, columns] = size(image);

    % claculate the row and the columns if each block is 8x8
    number_of_blocks_horzintally = ceil(columns/block_size); %220/8=27.5=28
    number_of_blocks_vertically = ceil(rows/block_size);

    %%%%%%%%%%%%%%%%%%%%%%%
    %%% splitting image %%%
    %%%%%%%%%%%%%%%%%%%%%%%
    blocked_image = zeros(block_size,block_size,...
        number_of_blocks_horzintally,number_of_blocks_vertically);

    for row=1:number_of_blocks_horzintally

        for column=1:number_of_blocks_vertically

            blocked_image(:,:,column,row) = ...
                image(row+(row-1)*block_size-(row-1):...
                row+(row-1)*block_size-(row-1)+block_size-1,...
                column+(column-1)*block_size-(column-1):...
                column+(column-1)*block_size-(column-1)+block_size-1);
        end
    end
end
```

Encoding:

Now we need to take each 8x8 block and subtract 128 from it because cosines waves around 0 accesses while image is ranged from 0 to 255 which centers at 128.

Then we will take the DCT of the block and get the quantized block.

DCT function

```
function DCT_image = DCT_block(splitted_image,block_size)

    if block_size == 8
        load C8.mat C8
    end

    % the values are from range 0-255 which is center at 128
    % we want them to be center at 0 so we will subtract 128 from all
    % elements
    shifted_image = int16(splitted_image) -128;

    DCT_image = dct2(shifted_image);
end
```

And the output of the DCT represents the weight or the amount of C8 that has effect on the image.

Now we just need to remove the high frequencies of the image and this process called quantization which represented by a table known at the transmitter and the receiver.

Quantization table is just dividing the value of each pixel to the corresponding one in the quantization table and round them to the nearest integer.

```
function Q = quantization(splitted_image,r)

    %load the quantization matrix and
    %round it off
    quantization_matrix=[16  11  10  16  24  40  51  61;
             12  12  14  19  26  58  60  55;
             14  13  16  24  40  57  69  56;
             14  17  22  29  51  87  80  62;
             18  22  37  56  68 109 103  77;
             24  35  55  64  81 104 113  92;
             49  64  78  87 103 121 120 101;
             72  92  95  98 112 100 103  99];

    T = r.*quantization_matrix;
    Q=round(splitted_image./T);
end
```

r represent how much of contribution each block will effect to our image in the quantization process.

The greater the r the high the contribution of removing the high frequencies blocks.

Decoding:

We first dequantize each block with same quantization table and r factor

```
function Q = dequantization(quantized_image,r)

    %load the quantization matrix and
    %round it off
    quantization_matrix=[16  11  10  16  24  40  51  61;
             12  12  14  19  26  58  60  55;
             14  13  16  24  40  57  69  56;
             14  17  22  29  51  87  80  62;
             18  22  37  56  68 109 103  77;
             24  35  55  64  81 104 113  92;
             49  64  78  87 103 121 120 101;
             72  92  95  98 112 100 103  99];

    T = r.*quantization_matrix;
    Q= quantized_image.*T;
end
```

Then we take the IDCT with adding 128 to regain the image in the correct range

```
function inv_DCT_image = inv_DCT_block(splitted_image,block_size)
    if block_size == 8
        load C8.mat C8
    end

    %inv_DCT_image = (transpose(C8)*splitted_image)*C8 +128;

    % we re-add 128 to return to the normal range of image which is from
    % 0-255
    inv_DCT_image = idct2(splitted_image) + 128;

end
```

We just need to reconstruct blocks again to form our image

```
function image = reconstruct(image_4d,block_size)

    [temp,temp2,columns,rows] = size(image_4d);

    image = zeros(rows*block_size,columns*block_size);

    for row=1:rows

        for column=1:columns

            image(row+(row-1)*block_size-(row-1):...
                row+(row-1)*block_size-(row-1)+block_size-1,...
                column+(column-1)*block_size-(column-1):...
                column+(column-1)*block_size-(column-1)+block_size-1) = ...
                image_4d(:,:,column,row);
        end
    end
end
```

Let's take an example on an image

```
% read the image
original_image = imread('Lenna.png');

% get number of rows, columns of the image
% in addition too number of channels to if it's color or gray
[rows, columns, number_of_color_channels] = size(original_image);
pixels = rows*columns;

% check if the image is color or gray
% if it's color convert it to gray
% if it's gray we are ready
if number_of_color_channels == 3
    gray_image = rgb2gray(original_image);
else
    gray_image = original_image;
end

r = 1;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Encoding %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

block_size = 8;

%%%%%%%%%%%%%%%
%%% padding %%%
%%%%%%%%%%%%%%%
padded_image = padding_by_block_size(gray_image,block_size);

%%%%%%%%%%%%%%%%%%%%%%%
%%% splitting image %%%
%%%%%%%%%%%%%%%%%%%%%%%
splitted_image = spliiting_image(padded_image,block_size);

% claculate the row and the columns if each block is 8x8
number_of_blocks_horzintally = ceil(columns/block_size); %220/8=27.5=28
number_of_blocks_vertically = ceil(rows/block_size);

% temporery splted image to store the DCT of the image
DCT_image =
zeros(block_size,block_size,number_of_blocks_vertically,number_of_blocks_horz
intally);

% temporery splted image to store the quantization of the image after
% takin its DCT
quantized_image =
zeros(block_size,block_size,number_of_blocks_vertically,number_of_blocks_horz
intally);

for row=1:number_of_blocks_horzintally
    for column=1:number_of_blocks_vertically
```

```
        %%%%%%%%%%%%%%%%%
        %%% DCT Block %%%
        %%%%%%%%%%%%%%%%%

        DCT_image(:,:,column,row) =
DCT_block(splitted_image(:,:,column,row),block_size);

        %%%%%%%%%%%%%%%%%%%%
        %%% quantization %%%
        %%%%%%%%%%%%%%%%%%%%

        quantized_image(:,:,column,row) =
quantization(DCT_image(:,:,column,row),r);

    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% decoding %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% temporery splted image to store the dequantization of the image
dequantized_image =
zeros(block_size,block_size,number_of_blocks_vertically,number_of_blocks_horz
intally);

% temporery splted image to store the IDCT of the image after getting
% its dequantization
inv_DCT_image =
zeros(block_size,block_size,number_of_blocks_vertically,number_of_blocks_horz
intally);

for row=1:number_of_blocks_horzintally
    for column=1:number_of_blocks_vertically

        %%%%%%%%%%%%%%%%%%%%%%
        %%% dequantization %%%
        %%%%%%%%%%%%%%%%%%%%%%

        dequantized_image(:,:,column,row) =
dequantization(quantized_image(:,:,column,row),r);

        %%%%%%%%%%%%%%%%%%%%%%%%%
        %%% inverse DCT Block %%%
        %%%%%%%%%%%%%%%%%%%%%%%%%
```

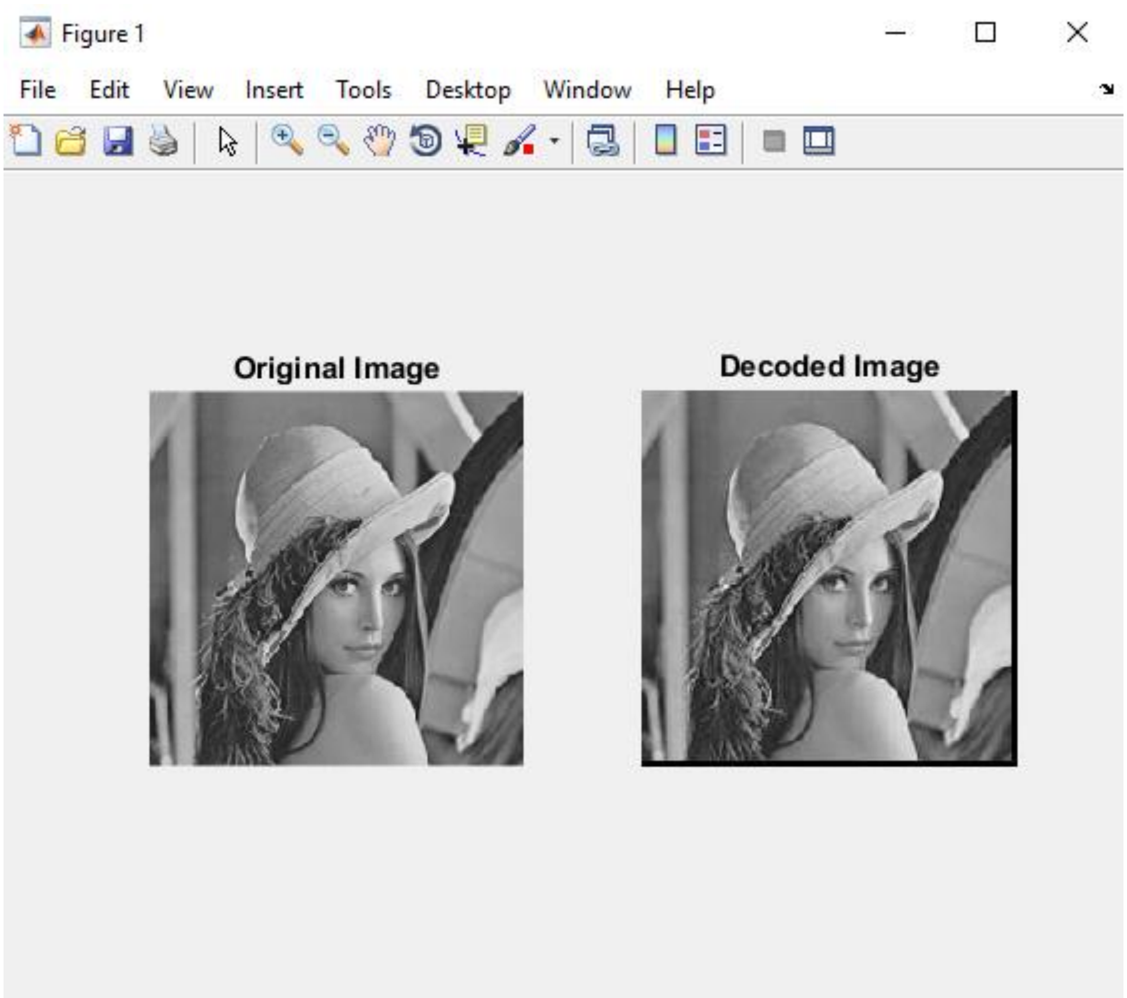
```
        inv_DCT_image(:,:,column,row) =
inv_DCT_block(dequantized_image(:,:,column,row),block_size);

    end
end

decoded_image = uint8(reconstruct(uint8(inv_DCT_image),block_size));
subplot(1,2,1)
imshow(original_image)
title('Original Image')
subplot(1,2,2)
imshow(decoded_image)
title('Decoded Image')
```

Output:

At r=1



At r=2

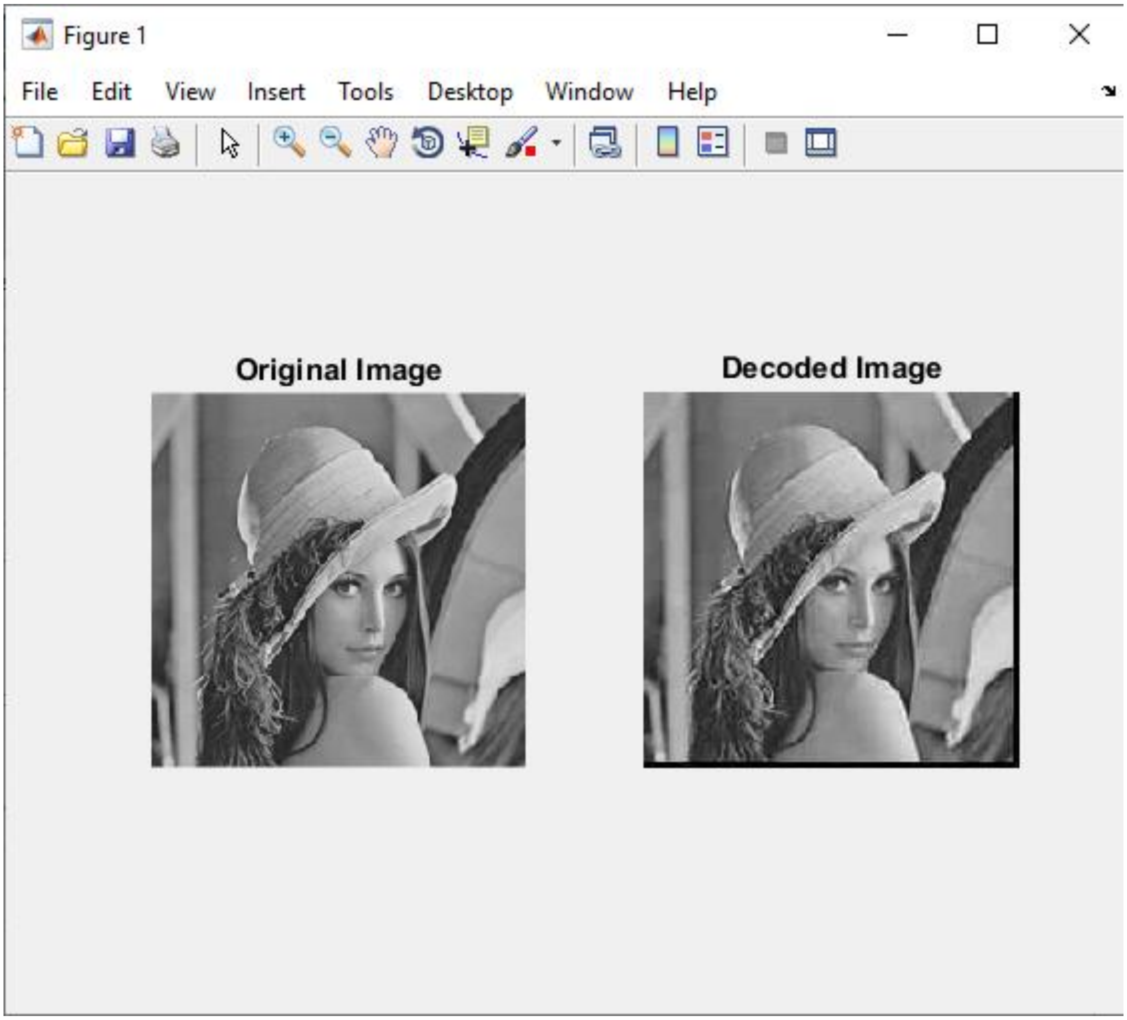


Figure 1
File Edit View Insert Tools Desktop Window Help
Original Image
Decoded Image

At r=4

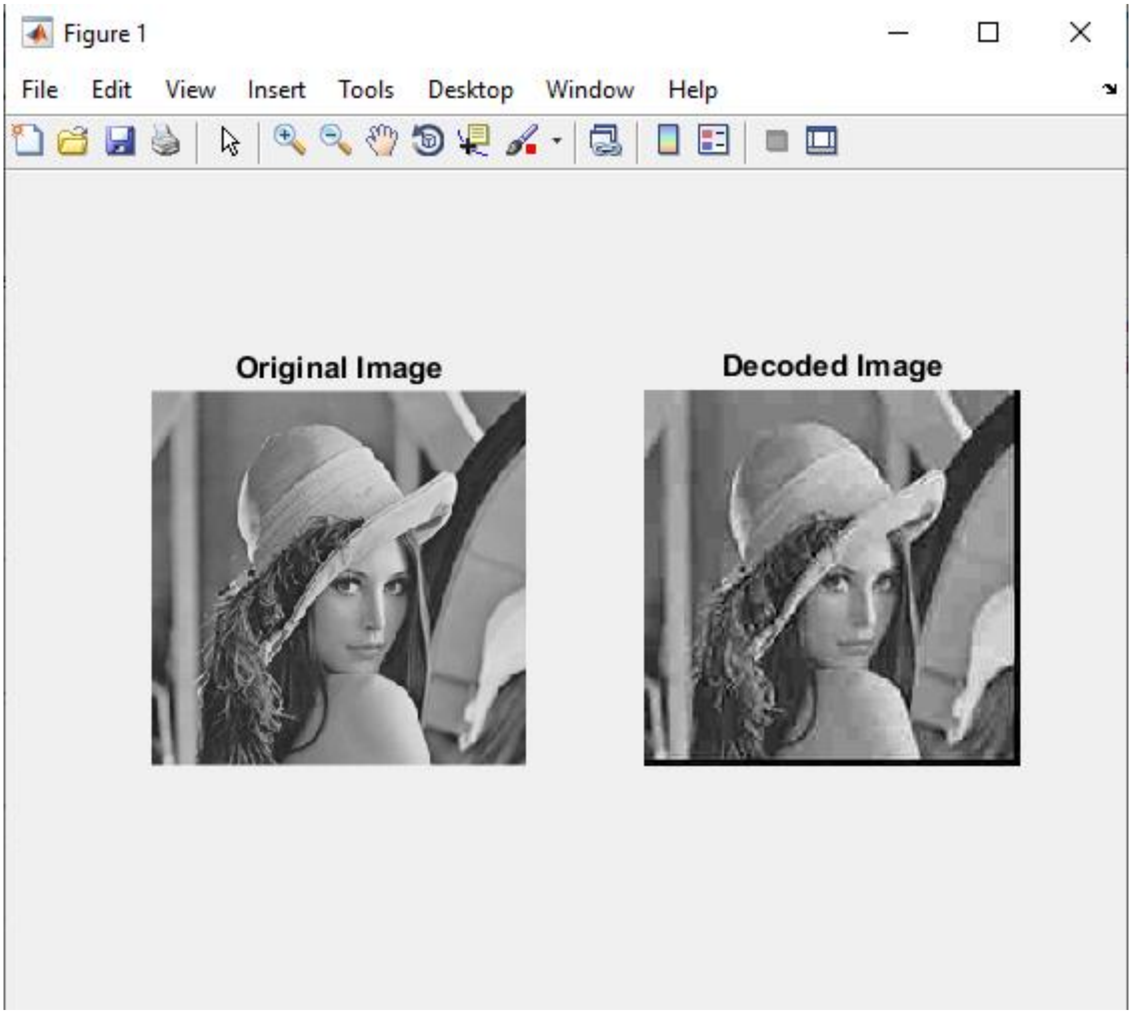


Figure 1
File Edit View Insert Tools Desktop Window Help
Original Image
Decoded Image

At r=8

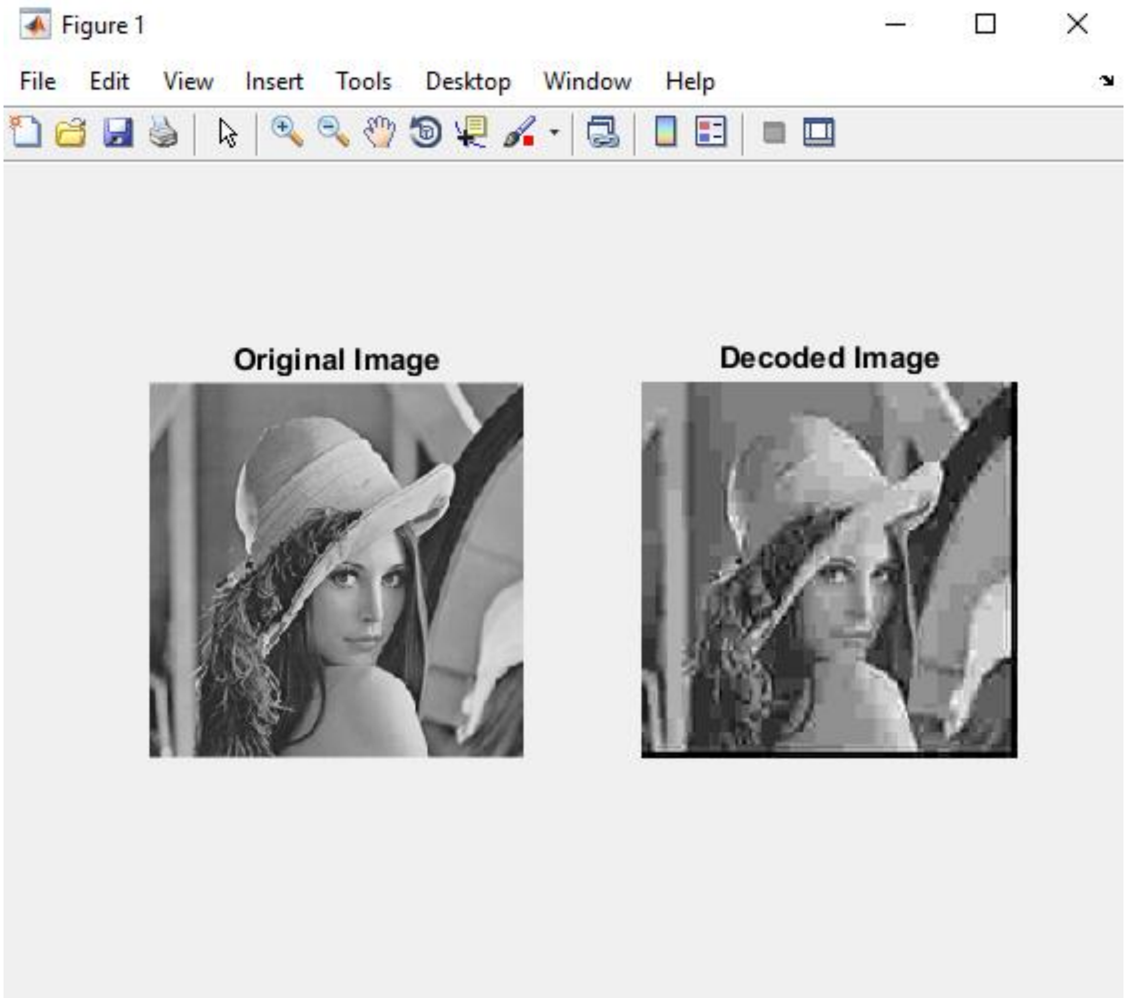


Figure 1
File Edit View Insert Tools Desktop Window Help
Original Image
Decoded Image

At r=10

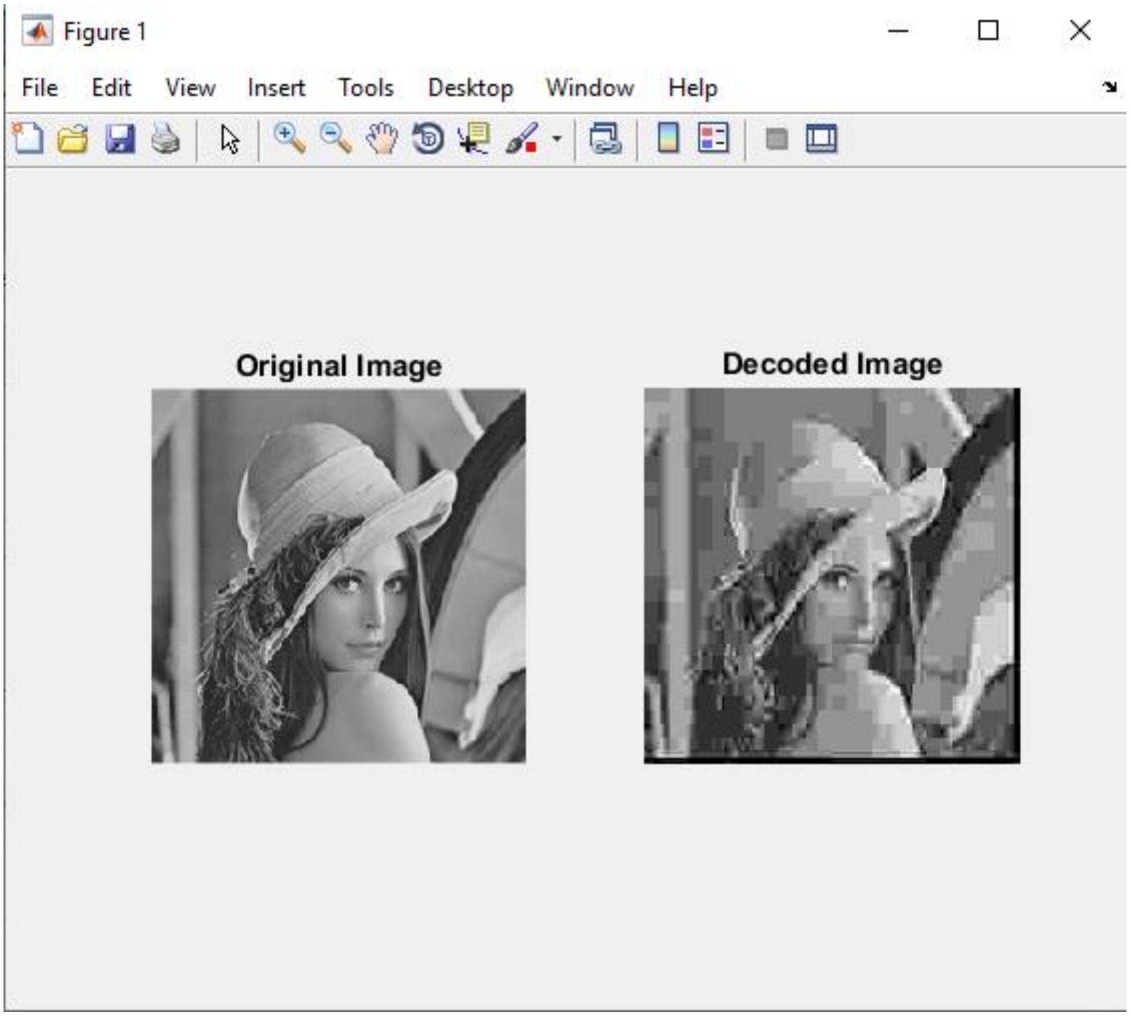
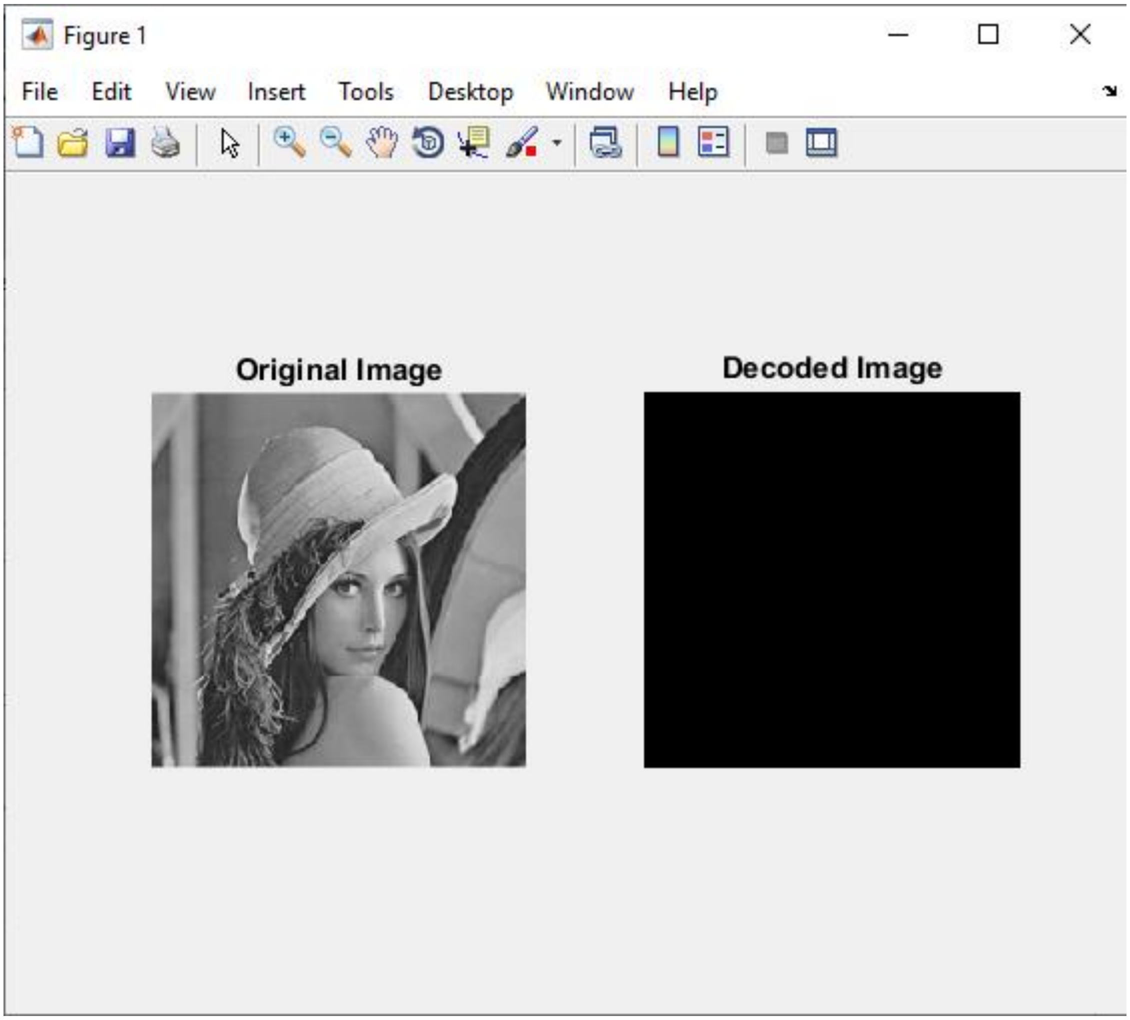


Figure 1
File Edit View Insert Tools Desktop Window Help
Original Image
Decoded Image

At r=0



As we can see by increasing r we remove more high frequencies part og the image as r affects the quantization table.

So the smaller r is the exact image I get but if we reach 0 we make the quantization 0s so the image gets black and I can’t redo that.