Report for Lab 2: The Wavelet Transform

Exercise 3

3.1 Discrete Wavelet Transform Filter Bank

(For your report after the lab): Explain how this Matlab function works, and check that it implements the recursion formulas similar to $c_{m-1,n} = \sqrt{2}\sum_i h_0[i-2n]c_{m,i}$ and $d_{m-1,n} = \sqrt{2}\sum_i h_1[i-2n]c_{m,i}$. (Are the filters **h0** and **h1** in the Matlab code *exactly* the same filters as h_0 and h_1 in these formulas? Explain any difference(s) you discover).

Since the formulas are recursive, the MATLAB codes constructed the dwt_haar function using a recursive method as well.

```
% Set the Haar analysis filter
h0 = [1/2  1/2];        % Haar Low-pass filter
h1 = [-1/2 1/2];        % Haar High-pass filter
```

In the function, the low-pass filter is exactly the same filter as h0 in formula, however, the high-pass filter is [-1/2, 1/2] which is different from the h1 = [1/2, -1/2] in the formula. However this is not the problem. Since the formulas combine both the filtering and down-sampling process in one step. Actually, the dwt_haar function split these two steps.

First, filtering the given sequence by convolution with h0 and h1 respectively. (Remind the convolution process: we first reflect h[i] to h[-i], since h0[0] = h1[0] = 1/2, no change is needed, whereas, h1[0] does not equal to h1[1], that's why the sequence of h1 elements are different between the codes and the formula.)

```
% Filter the signal
lowpass_c = conv(h0, c);
hipass_c  = conv(h1, c);
```

$$c_{m-1}[i] = \sum_i h_0[n-i]\cdot c_m[i] = h[i] * c_m[i]$$

The formula above is the first filtering step in the code. (For d, the process is the same)

Second, down-sampling and scaling the result after filtering.

```
% Subsample by factor of 2 and scale
c1 = sqrt(2)*lowpass_c(2:2:end);
d1 = sqrt(2)*hipass_c(2:2:end);
```

$$c_{m-1}[i] = \sqrt{2}\cdot c_{m-1}[\frac{i-1}{2}]$$

The formula above is the second down-sampling and scaling step, in which $\sqrt{2}$ is the normalization factor. (For d is also the same)

After these two step, link the d_{m-1} sequence after the c_{m-1} sequence, which constructs an integrated c_{m-1} sequence.

Besides, the dwt_function use a recursive way of programming.

```
% If no steps to do, or the sequence is a single sample, the DWT is itself
if (0==N | steps == 0)
    dwtc = c;
    return
end
```

This is the terminating or ending condition for the recursive loop. The condition is when steps reach to 0.

```
% Recursively call dwt_haar on the low-pass part, with 1 fewer steps
dwtc1 = dwt_haar(c1, steps-1);
```

This is the recursive call for dwt_haar function. Each time, it reduces 'steps' by 1.

Test this function by taking the Haar DWT of the following signal, to 1 step:

s = [1 6 7 9 7 4 -2 3]

Calculating 1-step Haar DWT for sequence s

```
% DWT of sequence s (step by step)
s = [1 6 7 9 7 4 -2 3];
dwt_1 = dwt_haar(s, 1);
```

The output (1-step Haar DWT):

```
>> dwt_1

dwt_1 =

    4.9497   11.3137    7.7782    0.7071   -3.5355   -1.4142    2.1213   -3.5355
```

Calculate by formulas

$s = [1\ 6\ 7\ 9\ 7\ 4\ -2\ 3]$

$$\begin{cases} c_{m-1,n} = \sqrt{2} \sum_{\tilde{v}} h_0[\tilde{v}-2n]\, c_{m,\tilde{v}} \\ d_{m-1,n} = \sqrt{2} \sum_{\tilde{v}} h_1[\tilde{v}-2n]\, d_{m,\tilde{v}} \end{cases}$$

first-level:

$c = \sqrt{2}\left[\frac{1+6}{2}, \frac{7+9}{2}, \frac{7+4}{2}, \frac{-2+3}{2}\right]$

$d = \sqrt{2}\left[\frac{1-6}{2}, \frac{7-9}{2}, \frac{7-4}{2}, \frac{-2-3}{2}\right]$

$[4.9\ \ 11.3\ \ 7.8\ \ 0.7\ \ -3.5\ \ -1.4\ \ 2.1\ \ -3.5]$

The results are matching.

Check the following (by recording values as necessary):

(a) The "length" (Matlab: **norm**) of the DWT of **s**, and check it is the same as for **s** itself.

(b) The first half of the DWT of s contains (scaled) sums of pairs of elements of s, while the second half contains scaled differences of pairs of elements of s.

(c) The one-step DWT of **s** is the same as multiplying **s** (actually the column vector **s′**) by the following matrix:

$$\mathbf{W}_1 = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{pmatrix}$$

(a)

Calculating the norm for sequence s and 1-step DWT

```
% (a)result
s_n = norm(s)
dwt1_n = norm(dwt_1)
```

The result show that they are equal

```
s_n = 15.6525
dwt1_n = 15.6525
```

(b)

```
% (b)result
co_1 = (s(1)+s(2))/(2*dwt_1(1))
co_2 = (s(3)+s(4))/(2*dwt_1(2))
co_3 = (s(5)+s(6))/(2*dwt_1(3))
co_4 = (s(7)+s(8))/(2*dwt_1(4))
co_5 = (s(1)-s(2))/(2*dwt_1(5))
co_6 = (s(3)-s(4))/(2*dwt_1(6))
co_7 = (s(5)-s(6))/(2*dwt_1(7))
co_8 = (s(7)-s(8))/(2*dwt_1(8))
```

```
co_1 = 0.7071
co_2 = 0.7071
co_3 = 0.7071
co_4 = 0.7071
co_5 = 0.7071
co_6 = 0.7071
co_7 = 0.7071
co_8 = 0.7071
```

The result shows that, The first half of the DWT of s contains scaled sums of pairs of elements of s, while the second half contains scaled differences of pairs of elements of s. The scaling factor is just the normalization factor $\sqrt{2}$

(c)

```
% (c)result
% define the Haar matrix
w = (1/sqrt(2)) * [1 1 0 0 0 0 0 0; 0 0 1 1 0 0 0 0;
    0 0 0 0 1 1 0 0; 0 0 0 0 0 0 1 1;
    1 -1 0 0 0 0 0 0; 0 0 1 -1 0 0 0 0;
    0 0 0 0 1 -1 0 0; 0 0 0 0 0 0 1 -1]
% The result by using bankfilter step 1
dwt_1
% The result by using Haar matrix step 1
dwt_1_Haar = w*s'
% The result by using bankfilter step 2
dwt_2
% The result by using Haar matrix step 2
dwt_2_Haar = (w*w)*s'
% The result by using bankfilter step 3
dwt_3
% The result by using Haar matrix step 3
dwt_3_Haar = (w*w*w)*s'
```

The haar matrix (8×8) :

```
W = 8×8
    0.7071    0.7071         0         0         0         0         0         0
         0         0    0.7071    0.7071         0         0         0         0
         0         0         0         0    0.7071    0.7071         0         0
         0         0         0         0         0         0    0.7071    0.7071
    0.7071   -0.7071         0         0         0         0         0         0
         0         0    0.7071   -0.7071         0         0         0         0
         0         0         0         0    0.7071   -0.7071         0         0
         0         0         0         0         0         0    0.7071   -0.7071

dwt_1 = 1×8
    4.9497   11.3137    7.7782    0.7071   -3.5355   -1.4142    2.1213   -3.5355

dwt_1_Haar = 8×1
    4.9497
   11.3137
    7.7782
    0.7071
   -3.5355
   -1.4142
    2.1213
   -3.5355
```

As the result, both the filter-bank method and the haar matrix method have the same outputs.

The Haar DWT transform in other levels (At most 3 levels since length of sequence is 8, levels $= \log_2(8) = 3$)

```
dwt_2 = 1×8
   11.5000    6.0000   -4.5000    5.0000   -3.5355   -1.4142    2.1213   -3.5355

dwt_2_Haar = 8×1
   11.5000
    6.0000
   -3.5000
   -1.0000
   -4.5000
    5.0000
   -1.5000
    4.0000

dwt_3 = 1×8
   12.3744    3.8891   -4.5000    5.0000   -3.5355   -1.4142    2.1213   -3.5355

dwt_3_Haar = 8×1
   12.3744
   -3.1820
    0.3536
    1.7678
    3.8891
   -1.7678
   -6.7175
   -3.8891
```

3.2 Inverse DWT Filter Bank

Apply this to the signals you transformed in the previous section. Explain what happens.
(For your report after the lab): Explain how this Matlab function works, and check that it implements a recursion formula similar to: [illegible].

Apply idwt_haar to the 1-step haar DWT sequence dwt_1:

```
s_recover = idwt_haar(dwt_1, 1);
s_recover
```

Output:

s_recover = 1×8

	1	2	3	4	5	6	7	8
1	1.0000	6.0000	7.0000	9.0000	7.0000	4.0000	-2.0000	3.0000

The sequence transformed back to the given sequence s.

The recursive formula:

$$c_{m,n} = \sqrt{2}\left(\sum_i h_0[n-2i]c_{m-1,i} + \sum_i h_1[n-2i]d_{m-1,i} \right)$$

This formula also combines the up-sampling process and the filtering process into one step.
In the idwt_haar function, this has been split into two individual steps.

First, split the DWT sequence into c part and d part. The c part is the first half of the sequence, while the d part is the latter half of the sequence.

```
% Split signal into the two halves
N2 = (N+1)/2;
c1 = c(1:N2);          % First half of signal
d1 = c(N2+1:end);    % Second half of signal
```

Then, up-sample the c1 and d1 sequence. The up-sample method is to add zeros to the even samples. In this way, the sequence length of c1 and d1 is 2 times larger.

```
% Upsample the signals: first c ...
for i=1:N2
    up_c(2*i-1) = idwtc1(i); % Transfer odd samples
    up_c(2*i)   = 0;             % Even samples set to zero
end
% ... then d
for i=1:N2
    up_d(2*i-1) = d1(i);
    up_d(2*i)   = 0;
end
```

After that, filtering the c1 and d1 sequence respectively by convoluting with low-pass filter and high-pass filter. The difference is that the filters are flipped, since the re-synthesis filters need to be time-reversed.

```
% Set the Haar analysis filters
h0 = [1/2  1/2];        % Haar Low-pass filter
h1 = [-1/2 1/2];        % Haar High-pass filter

% Resynthesis filters are flip (time-reverse) of analysis filters
g0 = h0(end:-1:1);  % This will be [1/2  1/2] for Haar
g1 = h1(end:-1:1);  % This will be [1/2 -1/2] for Haar
```

```
% Filter the signals
filt_c = conv(g0, up_c);
filt_d = conv(g1, up_d);
```

Add up and scale (just like the adding up the two parts in the formula and multiplied by a normalization factor). After this, cut off the surplus elements (introduced by convolution).

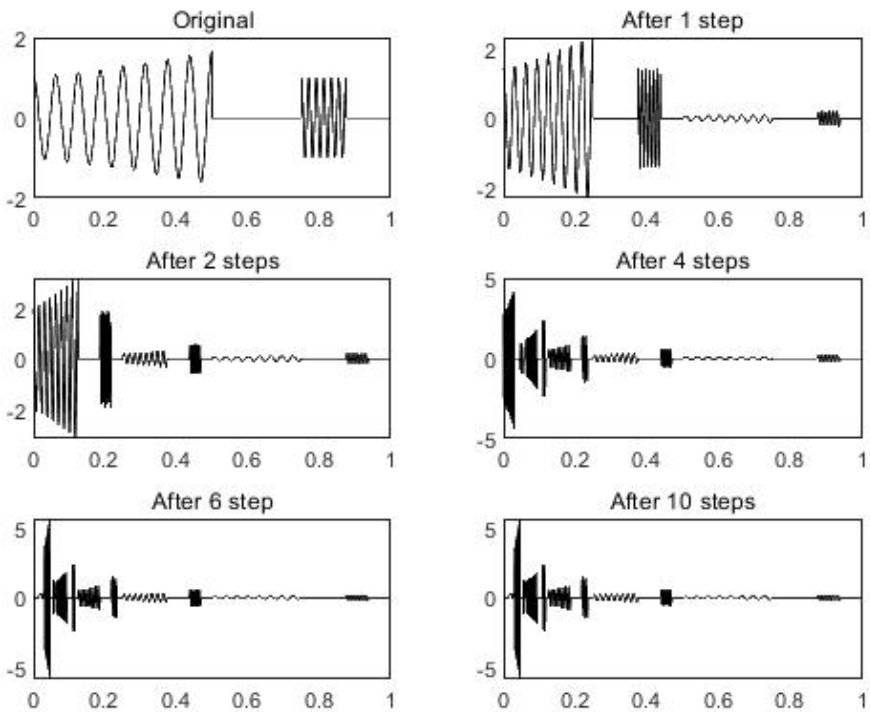
```
% Add and scale
idwtc = sqrt(2)*(filt_c + filt_d);

% Answer is defined for n=0..N (in Matlab, 1 to N)
idwtc = idwtc(1:N+1);
```

3.3 Signal Compression using the DWT

Download the following files: **`testsig.m`**, **`compress.m`**, **`uncompress.m`**, **`comp_ratio.m`**, **`distortion.m`**, **`ratedistortion.m`**. The file testsig.m generates the following signal, which we will use for testing:

$$s(t) = \begin{cases} e^{-t}\cos(32\pi) & 0 \le t < 1/2 \\ 0 & 1/2 \le t < 3/4 \\ \cos(96\pi) & 3/4 \le t < 7/8 \\ 0 & 7/8 \le t < 1 \end{cases}$$

Plot the DWT of this signal for a few different levels.



Modify the files compress.m and uncompress.m to include your transform of the signal where indicated.

compress.m

```
function y = compress(x, threshold, level)
% Compress signal using some function
% transformed signal values below threshold will be removed
%
% DO SOMETHING TO THE SIGNAL HERE
fx = dwt_haar(x, level);
% Keep only sufficiently large values of fx
y = fx .* (abs(fx)>=threshold);
% Done
```

uncompress.m

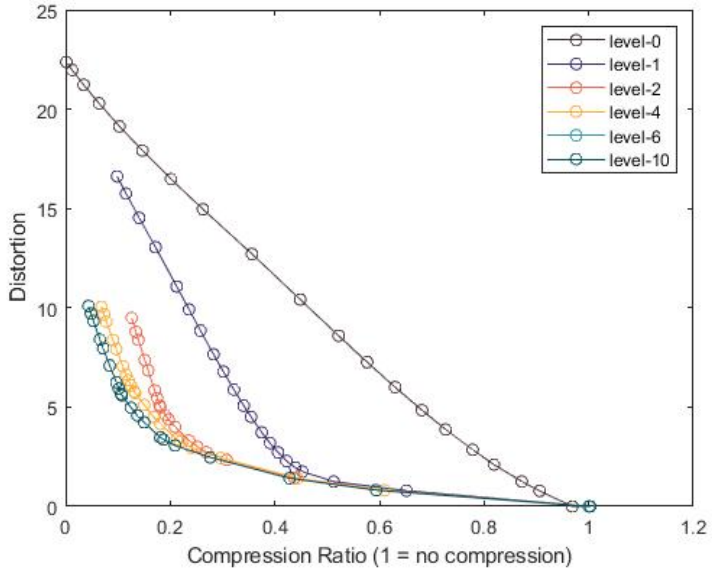
```
function x = uncompress(y, level)
% Compress signal using some function
% transformed signal values below threshold will be removed
%
% DO OPPPOSITE OF "COMPRESS" TRANSFORM HERE
x = idwt_haar(y, level);
% Done
```

Corresponding codes are as following:

```
%% Signal Compression using the DWT
% Generate the signal
sample = 1024;
dt = 1/sample;
t = 0:dt:1-dt;
s = testsig(t);
% Calculate dwt in different levels
s_1 = dwt_haar(s, 1);
s_2 = dwt_haar(s, 2);
s_4 = dwt_haar(s, 4);
s_6 = dwt_haar(s, 6);
s_10 = dwt_haar(s, 10);
```

```
% plot the signals in different levels
figure(1),
subplot(3,2,1),
plot(t,s,'k','linewidth',0.8); title("Original");
subplot(3,2,2),
plot(t,s_1,'k','linewidth',0.8); title("After 1 step");
subplot(3,2,3),
plot(t,s_2,'k','linewidth',0.8); title("After 2 steps");
subplot(3,2,4),
plot(t,s_4,'k','linewidth',0.8); title("After 4 steps");
subplot(3,2,5),
plot(t,s_6,'k','linewidth',0.8); title("After 6 step");
subplot(3,2,6),
plot(t,s_10,'k','linewidth',0.8); title("After 10 steps");
```

Once you have done this, you can use **`ratedistortion(maxt)`** to plot a rate-distortion curve up to a threshold **`maxt`** equal to the maximum signal value**.** Plot and compare the rate-distortion curves for DWT at a few different levels, as well as for the original signal.



The result of rate-distortion curves for DWT at different levels

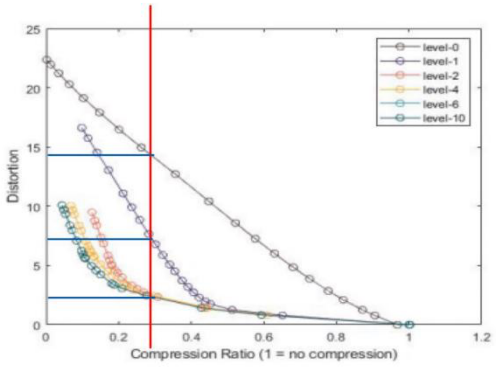
```
% check the ratedistortion of the signal
maxthreshold = max(s); % maximum signal value
figure(3),
% Plot the curves ('-o' means line with a circle at each plot point)
[ratevec, distvec] = ratedistortion(maxthreshold, 0);
plot(ratevec,distvec,'-o','Color',[71,51,53]/255); hold on;
[ratevec, distvec] = ratedistortion(maxthreshold, 1);
plot(ratevec,distvec,'-o','Color',[62,43,109]/255); hold on;
[ratevec, distvec] = ratedistortion(maxthreshold, 2);
plot(ratevec,distvec,'-o','Color',[240,100,73]/255); hold on;
[ratevec, distvec] = ratedistortion(maxthreshold, 4);
plot(ratevec,distvec,'-o','Color',[255,170,50]/255); hold on;
[ratevec, distvec] = ratedistortion(maxthreshold, 6);
plot(ratevec,distvec,'-o','Color',[48,151,164]/255); hold on;
[ratevec, distvec] = ratedistortion(maxthreshold, 10);
plot(ratevec,distvec,'-o','Color',[5,80,91]/255); hold on;
legend('level-0', 'level-1', 'level-2', 'level-4', 'level-6', 'level-10')
% Label the graph
xlabel('Compression Ratio (1 = no compression)');
ylabel('Distortion');
```

The corresponding code for drawing the curves

As shown from the above graph, there is a trade-off between compression rate and the distortion. At level-0, which means we only remain the maximum value of the original sequence, the curve is approximately linear. As the number of levels increase, the curves appear to be non-linear. Further more, at the same compression rate, the distortion seems better in higher levels. Besides, it shows that when levels are greater than 4, the curves seem to be near to each other which means, when the levels are quite large, the effect of reducing distortion at same compression rate is weak (not that strong).

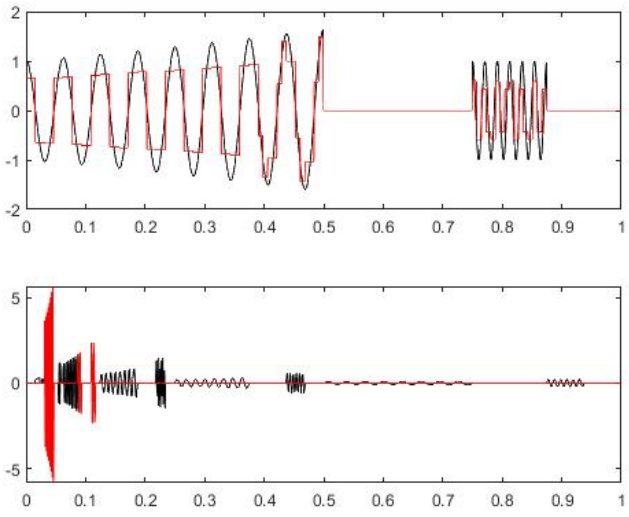
Discuss: Does the DWT helps to compress the signal? How can you tell?

In the above parts, we have noticed that at the same compression rate, if we apply DWT first, we will get less distortion. Further more, when we increase the levels, it shows that this distortion-reduction effect becomes better. The information can be deduced by the following graph.



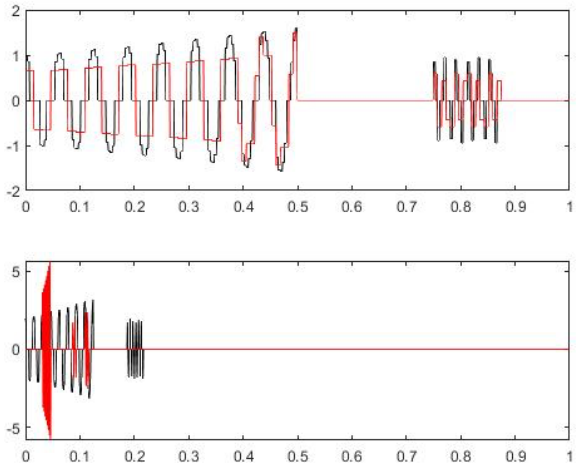
Red line is the fixed compression rate, blue lines show different distortion for different values

Perceiving the signals after DWT, we can find that after removing some samples below threshold, some of the fine details disappears. In this case, compression happens, since some of the signals are lost, however, just the fine details. Convert the signals back to time domain, we can achieve a compression signal with low distortion.



The above figure is the signal in time domain, the figure below is the signal after applying DWT. Black line represent original signal, red line represent signal after compression

As for compression in different levels, we know, in deeper levels, more information for details has been separated from the signal (approximation component). This explain why higher layers provide better reduction in distortion when at same compression rate.



Black line represent signal (after level-2 compression), red line represent signal after (after level-10 compression)

3.4 Comparison with FFT

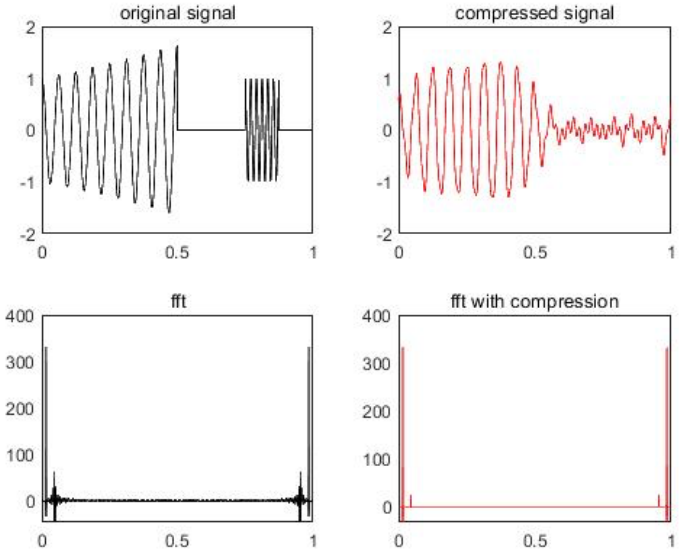
Repeat the experiment in section 3.3, but instead **use the FFT** in your compression system in place of the DWT.

```
function y = compress_fft(x, threshold)
% Compress signal using some function
% transformed signal values below threshold will be removed

fx = fft(x);
y = fx .* (abs(fx)>=threshold);
% Done
```

compress_fft.m

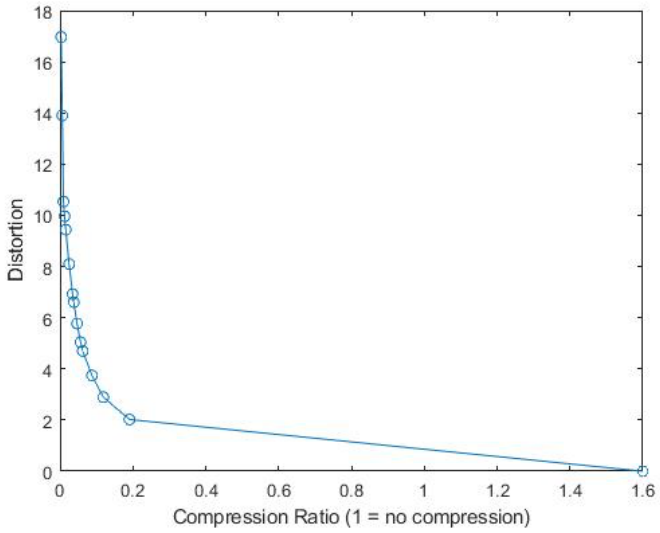
```
function x = uncompress_fft(y)
% Compress signal using some function
% transformed signal values below threshold will be removed
%
% DO OPPPOSITE OF "COMPRESS" TRANSFORM HERE
x = real(ifft(y));
% Done
```

uncompress_fft.m (only retain the real part)



The threshold is chosen to be $\frac{1}{5}\cdot\max(real(fft(s)))$

The compression by using FFT is similar to DWT. First transform the signal to its frequency domain, then removed samples below the threshold (here we choose th = $\frac{1}{5}\cdot\max(real(fft(s)))$), then applying the inverse transform IFFT and retain the real part.

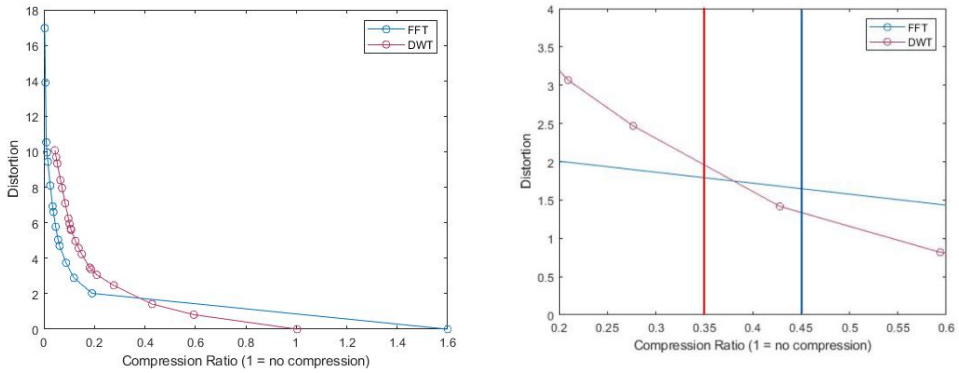


The compression-distortion curve using FFT compression (max=threshold = $\max(real(fft(s)))$)

Corresponding code for drawing two above graphs:

```
% Define the threshold
th = max(real(s_fft))/5;
s_fft = fft(s);
s_fft_com = compress_fft(s, th);
s_com = uncompress_fft(s_fft_com);
figure(1),
subplot(2,2,1),
plot(t,s,'k','linewidth',1.0);
axis([0,1,-2,2]); title("original signal");
subplot(2,2,2),
plot(t,s_com,'r','linewidth',1.0);
axis([0,1,-2,2]); title("compressed signal");
subplot(2,2,3),
plot(t,real(s_fft),'k','linewidth',1.0);
title("fft");
subplot(2,2,4),
plot(t,real(s_fft_com),'r','linewidth',1.0);
title("fft with compression")
```

Compare your results with the FFT to the results you got for the DWT.



DWT compression (level = 10) curve and FFT compression curve, fft_threshold = $\max(real(fft(s)))$, dwt_threshold = max(s)

From the curve above, we notice that, FFT compression yet retain the distortion-reduction effect. Besides, if we compare two curves, we find that when compression ratio $\approx$ 0.37, they have the same distortion. When compression ratio is less than 0.37, FFT compression has a better performance, whereas when compression ratio is greater than 0.37, DWT compression has a better performance, since, at this situation, less distortion is obtained during different types of compression.

Corresponding codes are as following:

```
% check the ratedistortion of the signal
dwt_threshold = max(s);
fft_threshold = max(real(s_fft));
figure(3),
ratedistortion_fft(fft_threshold), hold on;
[ratevec, distvec] = ratedistortion(dwt_threshold, 10);
plot(ratevec,distvec,'-o','Color',[0.6902, 0.18824,0.37647]);
legend('FFT','DWT');
% axis([0.2,0.6,0,4]);
```

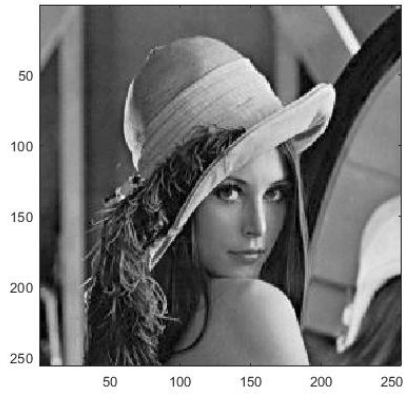
Exercises 4

4.1 2-D Wavelet Transform

Read and plot the test image, using

```
im = imreadreal('lena.bmp');
```

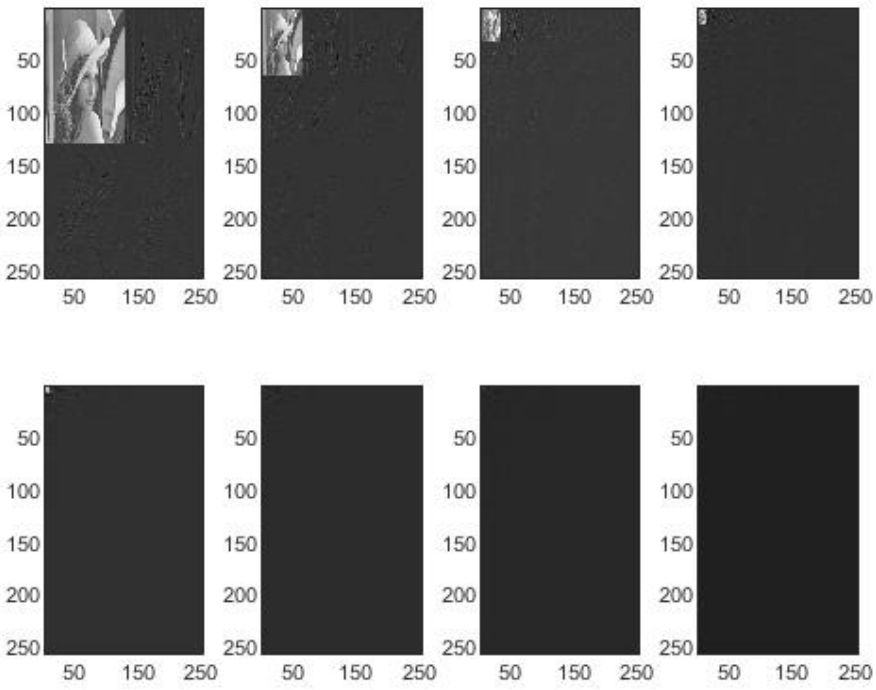
```
% read and plot the image
im = imreadreal('lena.bmp');
```



Take a 2D DWT of the test image at different number of steps, using `dwt2_haar(im, steps)`.

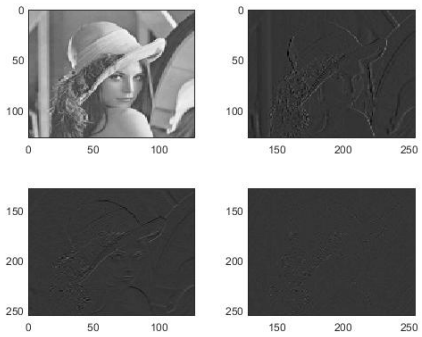
Describe qualitatively what happens to the image.

Since the *"lena"* is a 256 by 256 pixel image, the total level is 8 ($\log_2 256$)

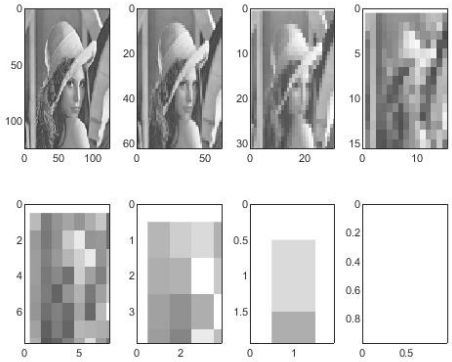


From the top left to the bottom right, DWT outputs in all 8 steps

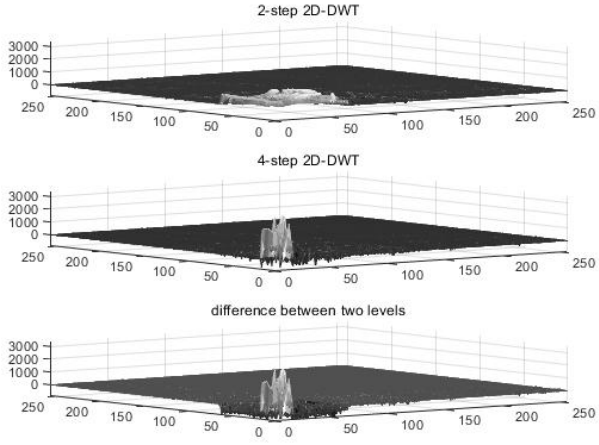
From the consecutive images above, we noticed that when DWT goes deeper, the main part (approximation part) of the image is always at the left-top position. Besides the size of the approximation part becomes smaller and smaller, whereas the size (or the pixels) for fine details increases step by step.
The main part and fine detail parts in level-2 DWT are separated as the following graph:



For the main part, if we only stretch out this part in each level, we find that the resolution dropped markedly.



Besides, if we compare the main part between a shallow layer and a deep layer, we found that, most of the main part values are larger in deep layer than in shallow layer (compare the z-axis height, and come to this conclusion).



To find out whether this is a general trend, and the reason for this trend. I calculate the average number of main part in each levels. The result shows that the average number will be doubled as the level goes deeper. Reminding the DWT transform in 1-D, there is a normalization factor $\sqrt{2}$ in the recursive formula when we calculate the deeper layer coefficients. In 2-D DWT, we apply 1-D DWT for both rows and vectors, so in this case, the factor changes to $2 = \sqrt{2} \cdot \sqrt{2}$.

The corresponding codes for calculating the mean values in each level:

```
size_im = size(im);
len  = size_im(1);
%Calculate the average of the main part of each DWT levels
average_calc = [0 0 0 0 0 0 0 0];
for steps_1 = 1 : log2(len)
    dwt_im = dwt2_haar(im, steps_1);
    scal = power(2,steps_1);
    arr = dwt_im(1:len/scal,1:len/scal);
    average_calc(steps_1) = mean(arr(:));
end
average_calc
```

The results are as follow:

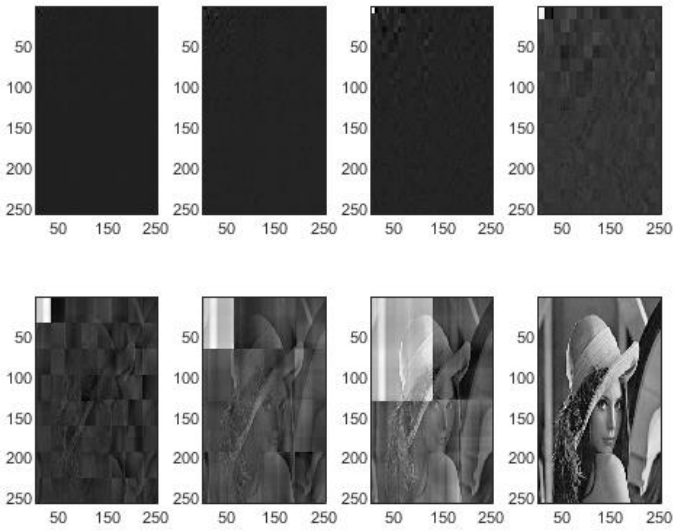
```
average_calc = 1×8
10^4 ×
    0.0208    0.0416    0.0833    0.1666    0.3331    0.6662    1.3324    2.6648
```

It shows that the mean value of each level doubles as level goes deeper.

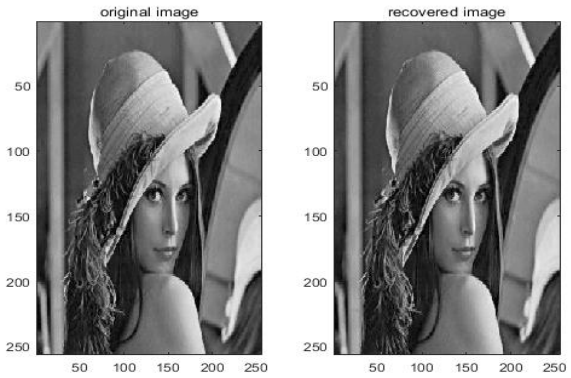
Write a Matlab function in the file **`idwt2_haar.m`** that will apply the 2D *Inverse* DWT for a given number of steps, and check that you can recover the original image.

The idwt2_haar,m function:

```
function y = idwt2_haar(x, steps)
% 2D Haar Discrete Wavelet Transform

% Get the size of x as (n x m)
[n, m] = size(x);
% Create matrix to hold the answer
y = zeros(n,m);
% First apply the IDWT to the rows of x
for i=1:n
   row = x(i,:);                      % Get the i-th row
   row = idwt_haar(row, steps); % Transform it
   y(i,:) = row;                      % Put into i-th column of y
end
% Now apply IDWT to columns
for j=1:n
   col = y(:,j)';                     % Get the i-th column (as a row vector)
   col = idwt_haar(col, steps); % Transform it
   y(:,j) = col';                     % Put back (as column) into i-th column of y
end
```



8 levels step by step idwt2_haar recover



The original image and the recovered image are the same

```
% dwt of the final version
size_dwt = size(dwt_result);
figure(2),
for steps_2 = 1 : log2(size_dwt(1))
    im_recover = idwt2_haar(dwt_result, steps_2);
    subplot(2,4,steps_2)
    colormap(gray);
    imagesc(im_recover)
end
im_rec_result = im_recover;
```

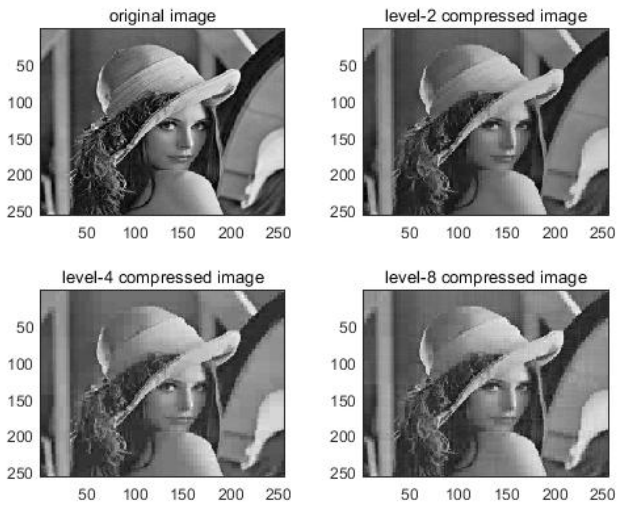
Corresponding codes for recovering the image

4.2 Image Compression using Wavelets

Modify the functions **`compress.m`** and **`uncompress.m`** to work with the 2D transform and inverse transform.

compress.m and uncompress.m to work with the 2D transform

```
function y = compress(x, threshold, level)
% Compress signal using some function
% transformed signal values below threshold will be removed
%
fx = dwt2_haar(x, level);
% Keep only sufficiently large values of fx
y = fx .* (abs(fx)>=threshold);
% Done

function x = uncompress(y, level)
% Compress signal using some function
% transformed signal values below threshold will be removed
%
% DO OPPPOSITE OF "COMPRESS" TRANSFORM HERE
x = idwt2_haar(y, level);
% Done
```



Different level 2-D DWT compression (threshold = 50)

Corresponding codes:

```
% Define the threshold
% th = max(max(im)); % maximum signal value
th = 50; % a chosen threshold number
level1 = 2;
level2 = 4;
level3 = 8;
im_dwt2 = compress(im, th, level1);
im_comp2 = uncompress(im_dwt2, level1);
im_dwt4 = compress(im, th, level2);
im_comp4 = uncompress(im_dwt4, level2);
im_dwt8 = compress(im, th, level3);
im_comp8 = uncompress(im_dwt8, level3);
```

Calculate the cosine similarity between the original image and each compressed image, it shows that the image compressed with higher-level appears more differences from the original image, though it is not obvious.
The cosine similarity outputs:

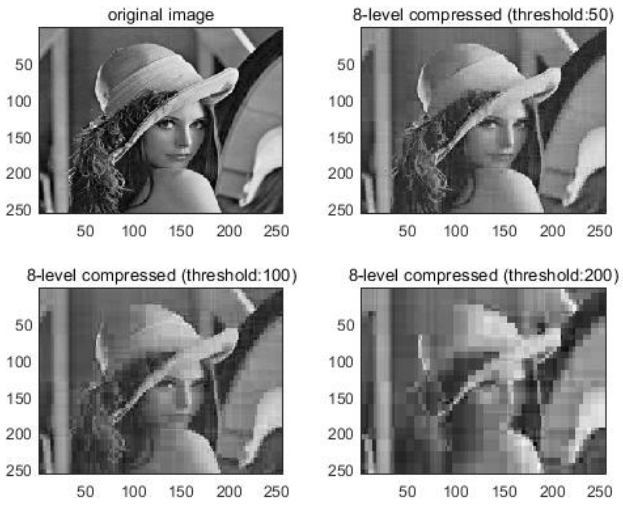
```
cosine_2 = 0.9963
cosine_4 = 0.9955
cosine_8 = 0.9953
```

The corresponding codes and functions:

```
cosine_2 = similarity(im, im_comp2)
cosine_4 = similarity(im, im_comp4)
cosine_8 = similarity(im, im_comp8)
```

The similarity.m function:

```
function [cosine] = similarity(inputArray1, inputArray2)
%SIMILARITY describe the similarity between two array(cos similarity)
%[cosine] = similarity(inputArray1, inputArray2)
vector1 = inputArray1(:);
vector2 = inputArray2(:);
cosine = dot(vector1,vector2)/( sqrt( sum( vector1.*vector1 ) ) * sqrt( sum( vector2.*vector2 ) ) );
end
```



Level 8 2-D DWT compression (with threshold = 50, 100, 200)

Corresponding codes:

```
th1 = 50; th2 = 100; th3 = 200;
im_dwt = compress(im, th1, 8);
im_comp_th1 = uncompress(im_dwt, 8);
im_dwt = compress(im, th2, 8);
im_comp_th2 = uncompress(im_dwt, 8);
im_dwt = compress(im, th3, 8);
im_comp_th3 = uncompress(im_dwt, 8);
```

Calculate the cosine similarity between the original image and each compressed image, it shows that the image compressed with larger threshold appears more differences from the original image (more distorted). It is obvious as we can see from the above graphs.

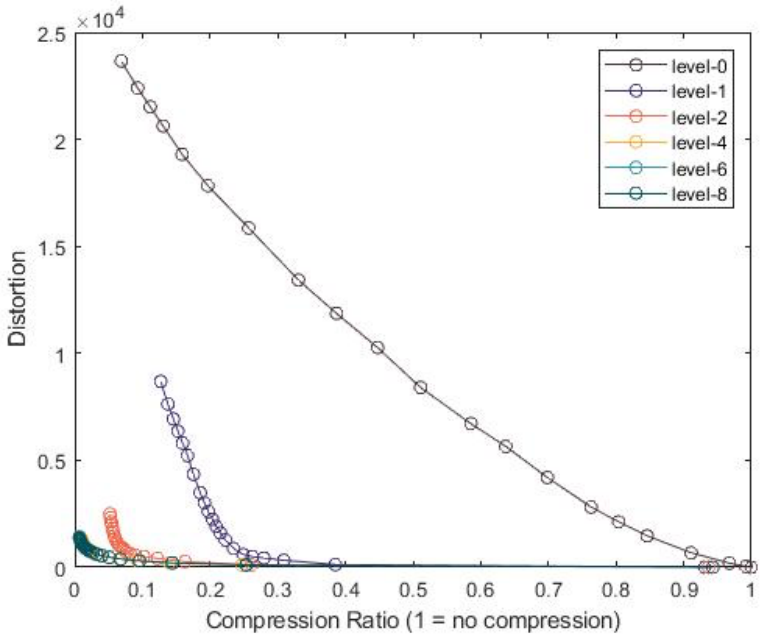
The cosine similarity outputs:

```
cosine_th1 = 0.9953
cosine_th2 = 0.9897
cosine_th3 = 0.9807
```

Corresponding codes:

```
cosine_th1 = similarity(im, im_comp_th1)
cosine_th2 = similarity(im, im_comp_th2)
cosine_th3 = similarity(im, im_comp_th3)
```

Calculate rate-distortion curves for 2D wavelet transforms at various levels, including none (no transform, or DWT with 0 steps), using the functions you have just modified and discuss these results.

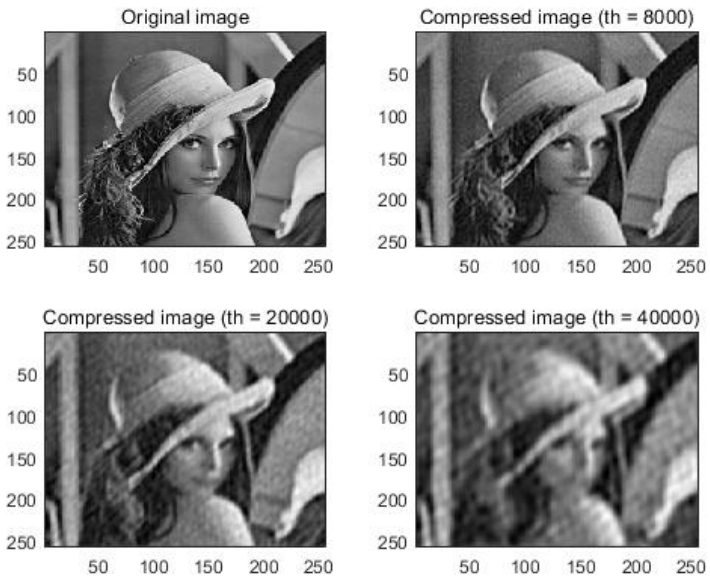


The result of rate-distortion curves for 2D-DWT at different levels (threshold = 200)

As for the result, it is similar to 1-D dimensional case. We can conclude that the compression ratio is negative correlated with distortion no matter in which level. Besides, at the same compression ratio, deeper levels have a lower distortion value. When level is larger than 4, the curves overlapped, which means the effect of reducing distortion at same compression rate is weak (not that strong).

4.3 Comparison with FFT

Repeat the 2D Image compression experiment using a 2D FFT in place of the DWT.



2-D FFT compression (with threshold = 8000, 20000, 40000)

Corresponding codes:

```
th1 = 8000;
th2 = 20000;
th3 = 40000;
im_fft = compress_fft(im, th1);
im_comp_th1 = uncompress_fft(im_fft);
im_fft = compress_fft(im, th2);
im_comp_th2 = uncompress_fft(im_fft);
im_fft = compress_fft(im, th3);
im_comp_th3 = uncompress_fft(im_fft);
```

Calculate the cosine similarity for FFT compression with different threshold. image compressed with larger threshold appears more differences from the original image (more distortion).

The cosine similarity outputs:

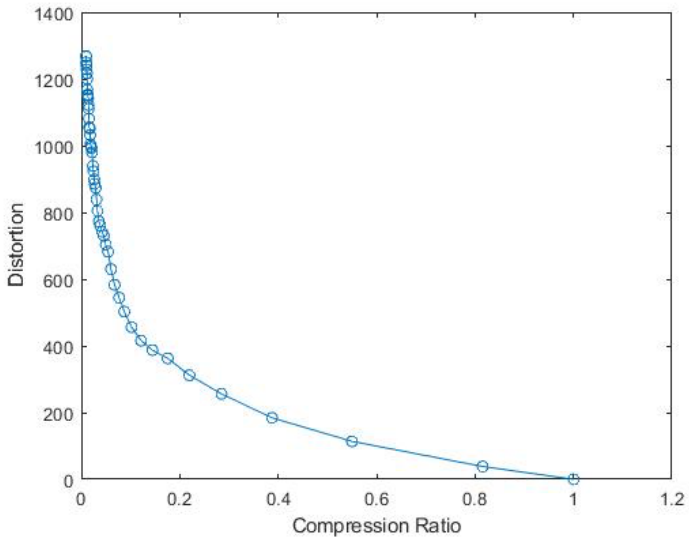
```
cosine_th1 = 0.9960
cosine_th2 = 0.9906
cosine_th3 = 0.9840
```

Corresponding codes:

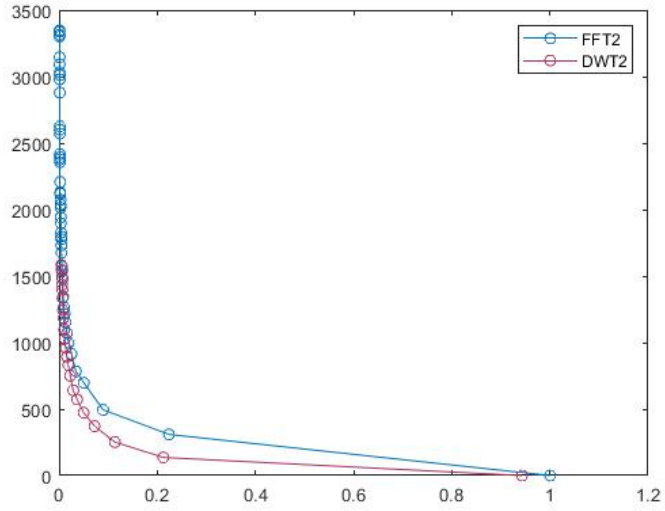
```
cosine_th1 = similarity(im, im_comp_th1)
cosine_th2 = similarity(im, im_comp_th2)
cosine_th3 = similarity(im, im_comp_th3)
```

Calculate the rate-distortion curves for a 2D FFT, and **compare** these results to the DWT.

Describe any differences you observe.



Compression-distortion curve for 2D-FFT



Compression-distortion curve for 2D-FFT and 2D-DWT

The result shows that 2D-FFT also have the distortion-reduction effect. Compared with 2D-DWT on image compression, 2D-FFT perform worse than 2D-DWT in general. If we draw a vertical line on a discretionary compression ratio, the distortion caused by 2D-DWT compression is always less than the distortion caused by 2D-FFT compression. Therefore, 2D-DWT perform better in image compression case.