ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ & ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ - ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ

"Ψηφιακή Επεξεργασία και Ανάλυση Εικόνας"

Ακαδημαϊκό Έτος 2021-22 (Εαρινό Εξάμηνο)

Εργαστηριακές Ασκήσεις - Μέρος Α

Γεωργακόπουλος Σωτήριος

ΑΜ: 1059310

Έτος: 5ο

1. Φιλτράρισμα στο πεδίο συχνοτήτων

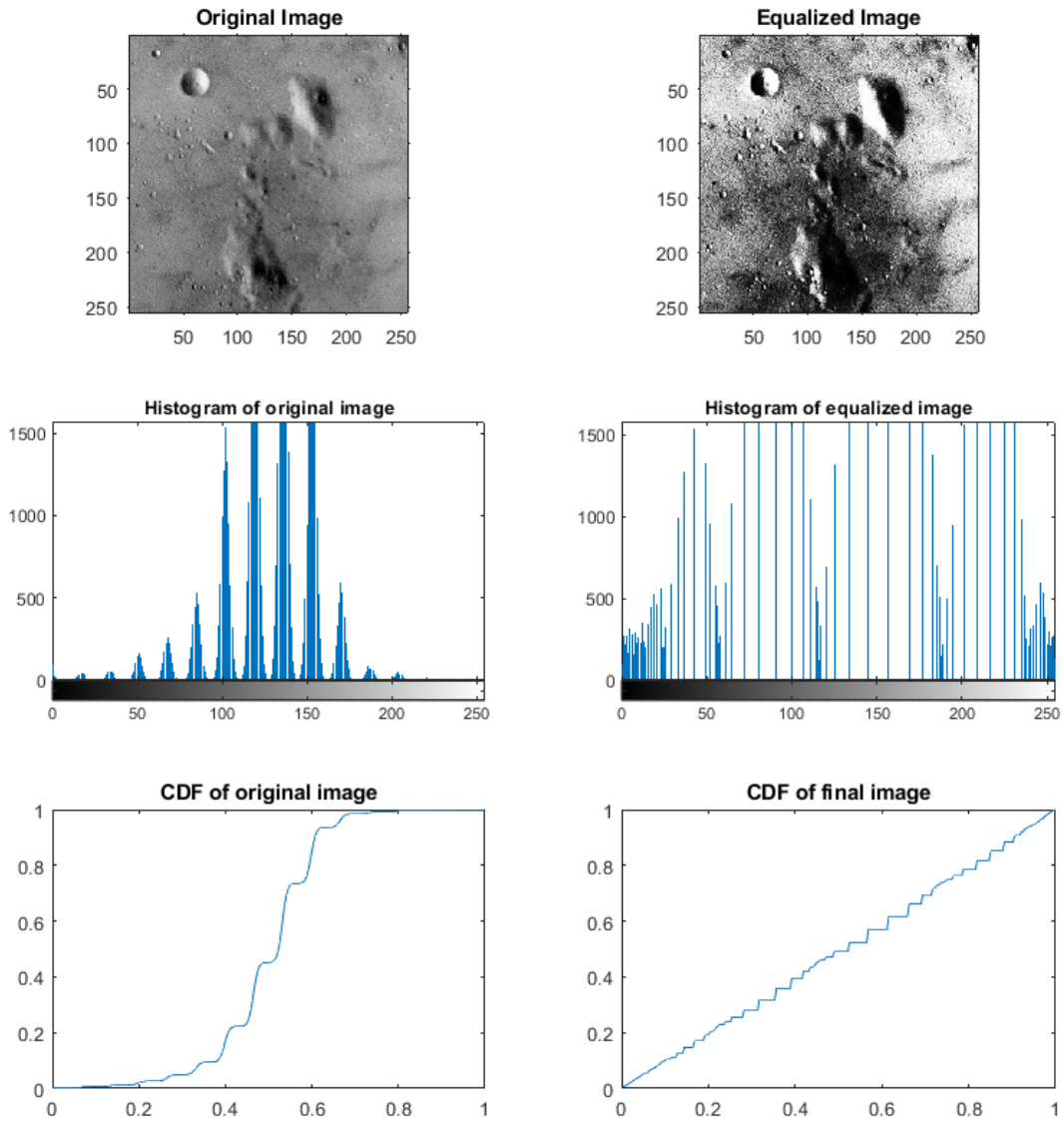
Για την εικόνα **moon.jpg** να εκτελεστούν τα παρακάτω βήματα :

1. ***Προεπεξεργασία*: Αρχικά εφαρμόστε γραμμικό μετασχηματισμό της περιοχής τιμών της εικόνας, ώστε να καλύπτει πλήρως τη δυναμική περιοχή [0: 255]. Στη συνέχεια, με χρήση κατάλληλης ιδιότητας του μετασχηματισμού DFT, μεταφέρετε το συχνοτικό σημείο (0,0) στο κέντρο του πεδίου.**

Φορτώνουμε την εικόνα, την μετατρέπουμε σε grayscale, και με την διαδικασία της εξίσωσης ιστογράμματος, φροντίζουμε να καλύπτει όσο το δυνατόν περισσότερο την δυναμική περιοχή [0:255].

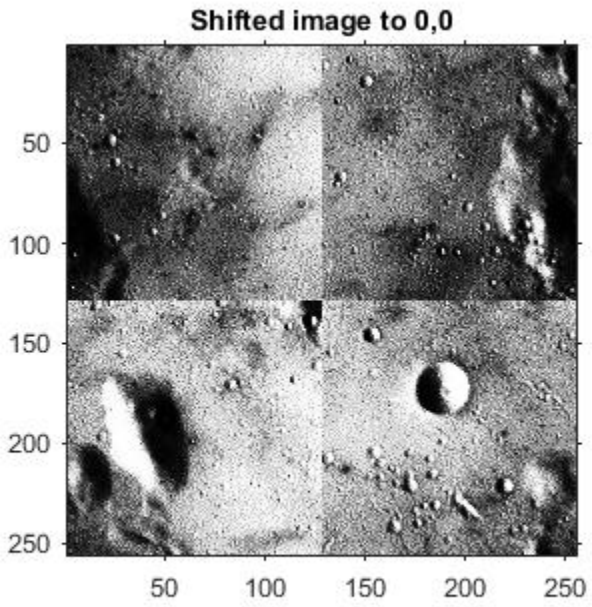
Για να φτιάξουμε το ιστόγραμμα, υπολογίζουμε την συχνότητα με την οποία εμφανίζεται κάθε τιμή έντασης της εικόνας και μπορούμε έτσι να υπολογίζουμε και την αθροιστική συνάρτηση πιθανότητας (cumulative distribution function). Στόχος είναι αυτή να την κάνουμε να προσεγγίσει όσο δυνατόν καλύτερα την y=x συνάρτηση, δηλαδή κάθε τιμή έντασης να εμφανίζεται με την ίδια συχνότητα.

Παρατηρούμε ότι μετά την εξίσωση ιστογράμματος, είναι πολύ πιο ευδιάκριτες στην εικόνα οι εναλλαγές στα χρώματα, άρα μπορεί κανείς να ανιχνεύσει πιο ευκολά τις εναλλαγές στο ύψος της επιφάνειας του φεγγαριού.



Εικόνα 1- Αριστερά η αρχική εικόνα και δεξιά η εικόνα μετά την εξίσωση ιστογράμματος

Και με την συνάρτηση fftshift() της matlab, που εκμεταλλεύεται την περιοδικότητα του DFT, μπορούμε να μετακινήσουμε το συχνοτικό σημείο (0,0) στο κέντρο του πεδίου, ενώ κανονικά θα βρισκόταν στο πρώτο και το τελευταίο στοιχείο του πεδίου.



Εικόνα 2 Παρατηρούμε ότι το πρώτο και το τελευταίο pixel της εικόνας μετακινούνται στο κέντρο του πεδίου

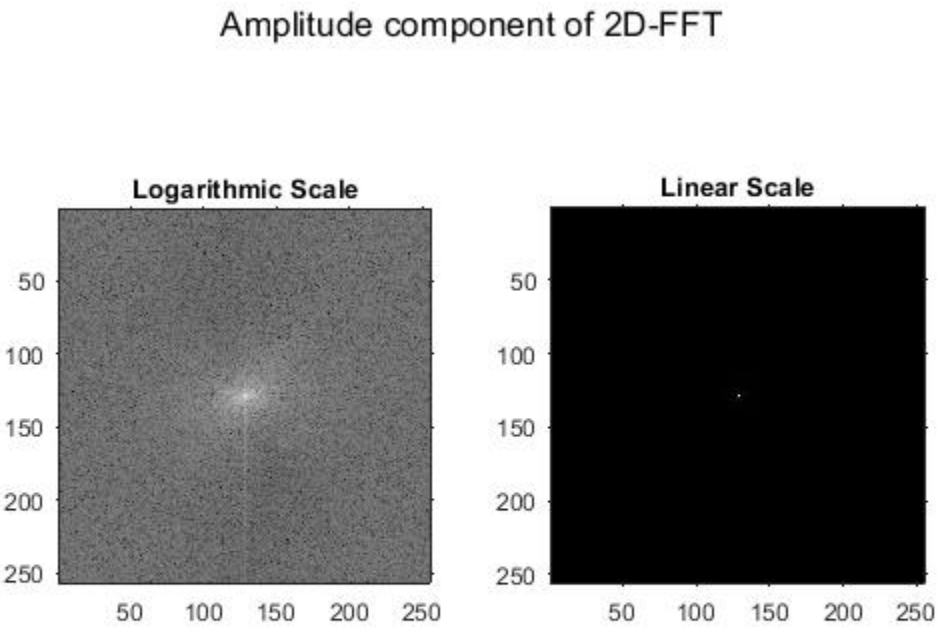
Κώδικας:

```
clear,close all;
moon = imread("moon.jpg");
%Transforming into grayscale

gray_moon = rgb2gray(moon);

freq = zeros(256,1);
L = 0:255;

%Cumulative Distribution Function calculation for histogram equalization

[height, width] = size(gray_moon);

%Find frequency of each intensity and pdf
for i = 1:height
    for j = 1:width
        for k = 1:256
            if gray_moon(i,j) == L(k)
                freq(k) = freq(k) + 1;
            end
        end
    end
end

pdf = freq / (height*width);
```

```
%Find cdf
cdf = cumsum(pdf);


%Histogram Equalization
eq_hist = round(cdf * 255);
equalized_image = zeros(height,width);

%Mapping new gray level
for i = 1:height
    for j = 1:width
        equalized_image(i,j) = eq_hist(gray_moon(i,j)+1);
    end
end

%Final image
final_image = uint8(equalized_image);


[final_counts, final_bins] = imhist(final_image);
final_cdf = cumsum(final_counts/(height*width));


f1 = figure("Name","Histogram Equalization");
f1.Position = [100 100 900 900];


subplot(3,2,1);
imshow(moon);
axis on
title("Original Image")

subplot(3,2,2);
imshow(final_image);
axis on
title("Equalized Image")


subplot(3,2,3);
imhist(gray_moon);
title("Histogram of original image")


subplot(3,2,4);
imhist(final_image);
title("Histogram of equalized image")


subplot(3,2,5);
plot((0:255)/255,cdf)
title("CDF of original image")

subplot(3,2,6);
plot((0:255)/255,final_cdf)
title("CDF of final image")

%Moving the 0 frequency point at the center with fftshift

shifted_moon = fftshift(final_image);
figure("Name","Shifting the 0,0 point");
imshow(shifted_moon);
axis on
title("Shifted image to 0,0");
```

2. **Να υλοποιηθεί ο δισδιάστατος διακριτός μετασχηματισμός Fourier με τη μέθοδο γραμμών-στηλών και χρησιμοποιώντας το μονοδιάστατο μετασχηματισμό DFT. Να γίνει γραμμική και λογαριθμική απεικόνιση του πλάτους του μετασχηματισμού της εικόνας.**

Υλοποιούμε τον δισδιάστατο DFT με τη χρήση του μονοδιάστατου DFT, πρώτα υπολογίζοντας τον 1D-FFT κάθε γραμμής και μετά κάθε στήλης. Με την χρήση του μετασχηματισμού y = log(1 + y) κάνουμε λογαριθμική απεικόνιση του πλάτους.



Εικόνα 3 Λογαριθμική και γραμμική απεικόνιση του πλάτους της εικόνας

Με την λογαριθμική κλίμακα μπορούμε να διακρίνουμε πολύ καλύτερα το συχνοτικό περιεχόμενο της εικόνας μας.

3. **Να φιλτράρετε την εικόνα στο πεδίο συχνοτήτων (u, v) με χρήση ενός κατωπερατού φίλτρου (low-pass filter) $H(u, v)$ με ζώνη διάβασης της επιλογής σας.**

Επιλέξαμε να φιλτράρουμε την εικόνα με ένα Gaussian Low Pass Filter, με συχνότητα αποκοπής fo τα 30 Hz, η συνάρτηση του οποίου είναι:

$$H_{Gaussian}(u, v) = e^{-\frac{D(u,v)^2}{2*f_0{}^2}}$$

Όπου D είναι η απόσταση του κάθε pixel από το συχνοτικό σημείο 0,0.

Κώδικας:

```
% Cut-off Freq
f_cut = 30;

P = round(height/2);
Q = round(width/2);

% Defining the filter kernel
H = zeros(height,width);
for i = 1:height
    for j = 1:width
        D = (i - P).^2 + (j-Q).^2;
        H(i,j) = exp(-D/2/f_cut^2);
    end
end

%Applying the filter

filteredImage = H .* fftshift(FFT_of_moon);
```

4. Να εφαρμοστεί ο δισδιάστατος αντίστροφος διακριτός μετασχηματισμός IDFT για την επαναφορά στο χωρικό πεδίο.

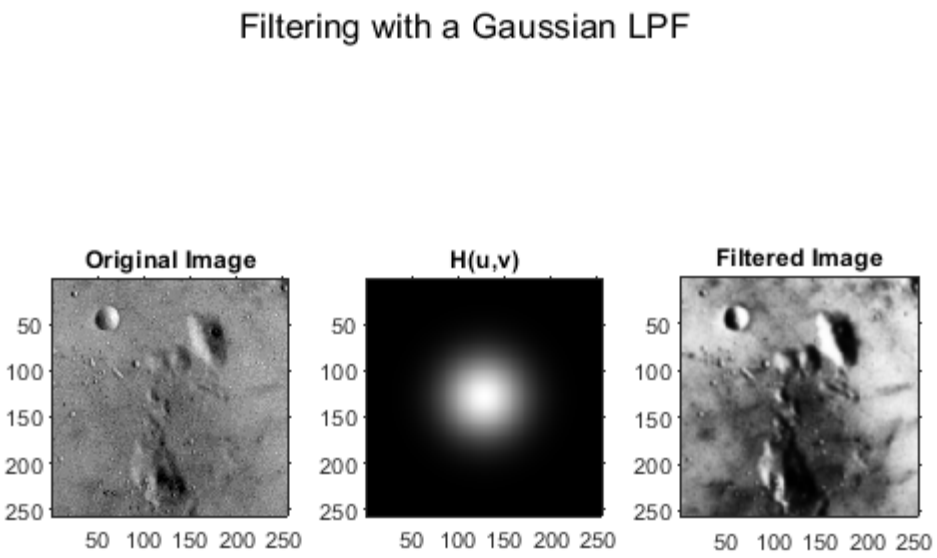
Εφαρμόζουμε τον 2D-IFFT πάλι με την μέθοδο γραμμών-στηλών, υπολογίζοντας τον IFFT για τις γραμμές και έπειτα για τις στήλες.

5. Τέλος, εφαρμόστε την αντίστροφη διαδικασία του Βήματος 1, έτσι ώστε να επαναφέρετε το σημείο (0,0).

Ξαναχρησιμοποιούμε την fftshift() για να επαναφέρουμε το συχνοτικό σημείο (0,0) στην αρχική θέση του.

Κώδικας:

```
%Applying IDFT
final_image=ifft(ifft(fftshift(filteredImage),[],1),[],2);

%Shifting back
final_image = fftshift(final_image);

figure("Name","Filtering with a Gaussian LPF");

subplot(1,3,1)
imshow(gray_moon);
axis on
title("Original Image");

subplot(1,3,2)
imshow(H);
axis on
title("H(u,v)");

subplot(1,3,3)
imshow(uint8(abs(final_image)));
axis on
title("Filtered Image");
sgtitle("Filtering with a Gaussian LPF");
```



Εικόνα 4 Αριστερά: Η αρχική εικόνα, Κέντρο: Η συνάρτηση του Gaussian LPF, Δεξιά: Η φιλτραρισμένη εικόνα

2. Συμπίεση Εικόνας με χρήση μετασχηματισμού DCT

Θεωρήστε την εικόνα barbara.mat και εκτελέστε τις παρακάτω επεξεργασίες:

Συμπίεση με τη χρήση του μετασχηματισμού 2D-DCT.Για να συμπιέσετε την εικόνα, αρχικά τεμαχίστε την σε μη επικαλυπτόμενες περιοχές διαστάσεων 32 × 32, και εφαρμόστε το μετασχηματισμό 2D-DCT σε κάθε μία περιοχή. Στη συνέχεια να επιλέξετε κατάλληλο υποσύνολο των συντελεστών του 2D-DCT της κάθε υποεικόνας, κρατώντας πληροφορία ενός ποσοστού r. Η επιλογή για το υποσύνολο συντελεστών να γίνει με δύο μεθόδους, συγκεκριμένα:

1. *Με τη μέθοδο ζώνης*
2. *Με τη μέθοδο κατωφλιού*

Απεικονίστε το μέσο τετραγωνικό σφάλμα ανάμεσα στην αρχική και την ανακατασκευασμένη εικόνα για τις τιμές του r που ανήκουν στο σύνολο τιμών [5%, 50%].

Για τον 2D-DCT, σπάμε την εικόνα σε μη επικαλυπτόμενες περιοχές διαστάσεων 32 x 32 (blocks), υπολογίζουμε τον DCT Transform Matrix με την συνάρτηση my1D_DCT για μέγεθος 32 * 32 και για να βρούμε τον τελικό μετασχηματισμό πολλαπλασιάζουμε από δεξιά και αριστερά το κάθε block με τον πίνακα αυτόν. Δηλαδή:

Έστω T ο M x M DCT Transform Matrix:

$$T(p,q) = \begin{cases} \frac{1}{\sqrt{M}}, \text{για } p = 0, 0 \le q \le M\text{-}1 \\ \sqrt{\frac{2}{M}} \cos\frac{\pi(2q+1)p}{2M}, \text{για } 1 \le p \le M\text{-}1, 0 \le q \le M\text{-}1 \end{cases}$$

Όπου M η διάσταση του block.

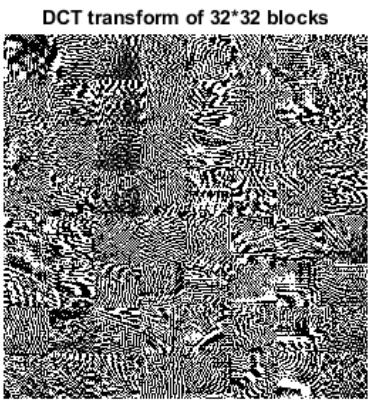
Τότε ο μετασχηματισμός 2D-DCT του κάθε block είναι: T * block * T'

Και ο ανάστροφός μετασχηματισμός 2D-IDCT είναι T / block / T'

Για την εφαρμογή του μετασχηματισμού σε κάθε block, χρησιμοποιούμε την συνάρτηση blockproc της MATLAB, η οποία εφαρμόζει μια συνάρτηση μετασχηματισμού σε μια εικόνα ανά block, για το μέγεθος block που της ορίζουμε και μας επιστρέφει το αποτέλεσμα.

Κώδικας:

```
function dct_1d= my1D_DCT(m,n)
%Calculates DCT Transform Matrix of m*m block


    c(1:m,1) = sqrt(2)/sqrt(m);
    c(1) = c(1) / sqrt(2);
    range = (0:m-1);
    dct_1d = c.*cos(range(:)*pi*(2*range+1)/(2*m));

end
function dct2D = my2D_DCT(inputImage)
%MY2D_DCT Calculates 2D-DCT of image using 1D-DCT

  image = double(inputImage);
  N = size(image,1);
  M = size(image,2);
  A=my1D_DCT(N,M);

  dct2D =A*image*A.';
end


function idct2D = my2D_IDCT(inputImage)
%MY2D_IDCT

  image = double(inputImage);
  N = length(image);
  A=my1D_DCT(N);

  idct2D =A\image/A.';
end


%% Exercise 2 - DCT Compression

close all; clear;
storedStruct = load("barbara.mat");
barbara = storedStruct.barbara;

figure;
imshow(barbara,[]);
title("Original Image");


I = im2gray(barbara);
windowSize = [32 32];

dct = @(block_struct) my2D_DCT(block_struct.data);
dct_transform = blockproc(I,windowSize,dct);

figure;
imshow(dct_transform);
title("DCT transform of 32*32 blocks");
```



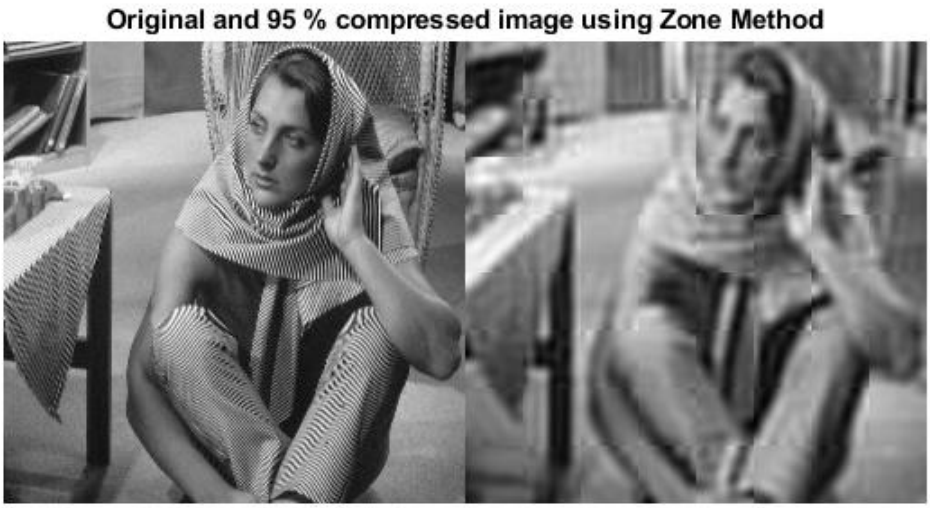
*Εικόνα 5 Αριστερά η αρχική εικόνα και δεξιά ο DCT Transform σε 32*32 blocks της εικόνας*

Για την επιλογή των συντελεστών, χρησιμοποιήθηκε αρχικά η μέθοδος ζώνης, σύμφωνα με την οποία εφαρμόζουμε έναν αναποδογυρισμένο άνω-τριγωνικό πίνακα, με 1 όσους και το πλήθος των συντελεστών που θέλουμε και 0 τα υπόλοιπα, για να κρατήσουμε μόνο τους πάνω αριστερά συντελεστές του κάθε block, οι οποίοι περιέχουν και την περισσότερη πληροφορία. Για διαφορετικό πλήθος συντελεστών υπολογίζουμε το μέσο τετραγωνικό σφάλμα ανάμεσα στην αρχική και στην ανακατασκευασμένη εικόνα.

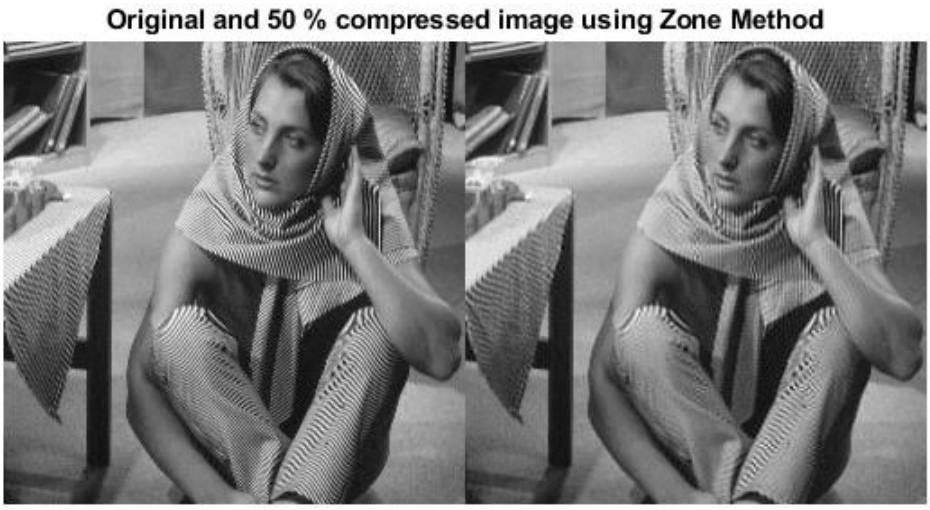
Κώδικας:

```
%% Quantization - Zone Method

% Keeping only the top right coefficients by applying a mask to each block
% r is the percentage of information kept
MSE1 = zeros(0.5/0.05,1);

for r=0.05:0.05:0.5
    no_of_coeffs = round(windowSize(1).^2 * r ) ;
    %Creating the mask by keeping upper diagonal matrix with no_of_coeffs
    %elements
    no_of_elements = 1;
    mask = triu(ones(windowSize),no_of_elements);
    while(no_of_coeffs < nnz(mask))
        mask = triu(ones(windowSize),no_of_elements);
        no_of_elements = no_of_elements + 1;

    end

    final_mask = flip(mask,2);
    final_mask = @(block_struct) (final_mask.*block_struct.data);
    maskedBlocks = blockproc(dct_transform,windowSize,final_mask); %Apply Mask

    IDCT = @(block_struct) my2D_IDCT(block_struct.data); %Inverse DCT
    restoredImage = blockproc(maskedBlocks,windowSize,IDCT); %Apply inverse DCT to
every 32x32 block
```

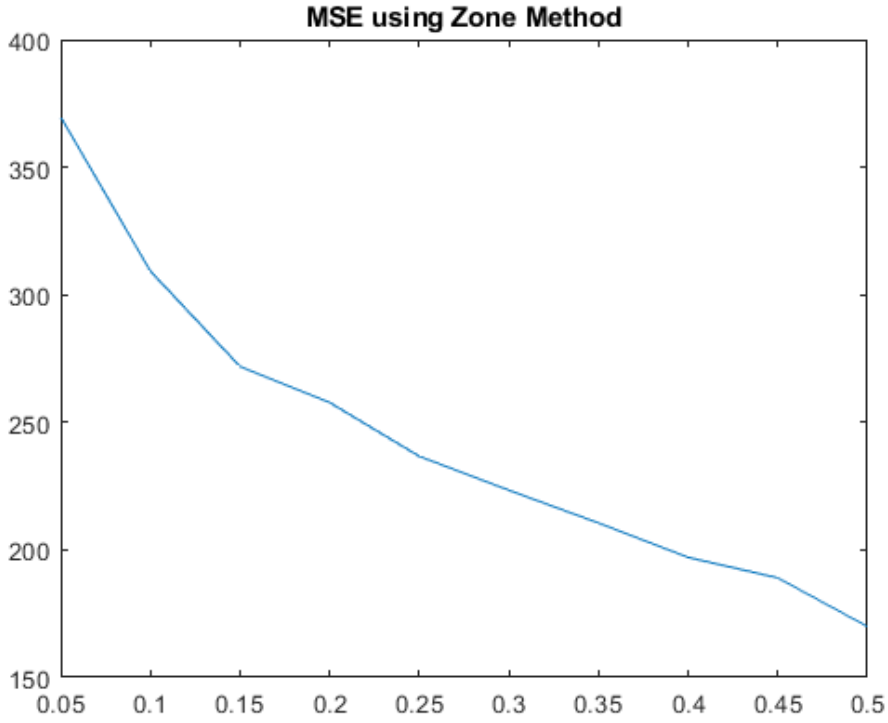
```
     if ((r == 0.05 ) || (r == 0.5))
        figure;
        imshowpair(I,restoredImage,'montage');
        title(sprintf('Original and %d %% compressed image using Zone
Method',uint8(100 - r*100)))
    end

    MSE1(round(r/0.05)) = sum(sum((double(I)-restoredImage).^2)) / (size(I,1) *
size(I,2));

end

%Plotting the Mean Square Error for different information percentage
f2 = figure;
inform_percent = 0.05:0.05:0.5;
plot(inform_percent,MSE1)
title("MSE using Zone Method");
```



Εικόνα 6 - Συμπίεση διώχνοντας το 95% των συντελεστών



Εικόνα 7 Συμπίεση διώχνοντας το 50% των συντελεστών.

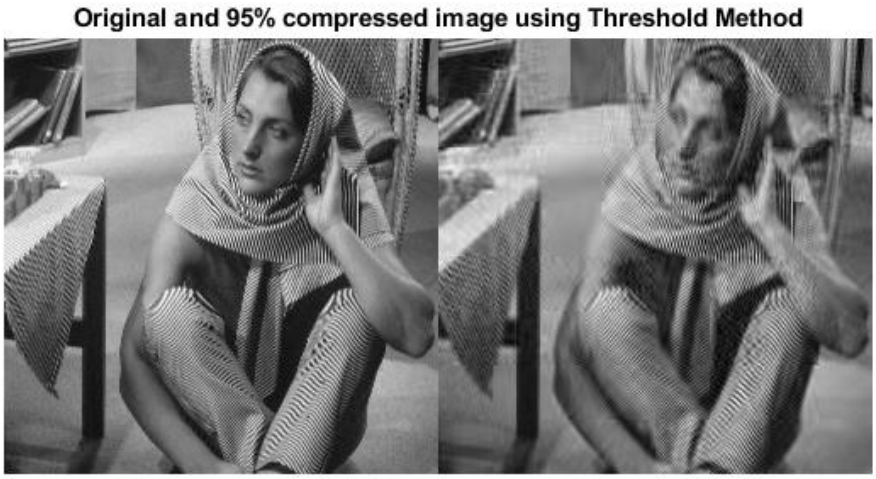


Εικόνα 8 Μέσο Τετραγωνικό Σφάλμα με την μέθοδο Ζώνης

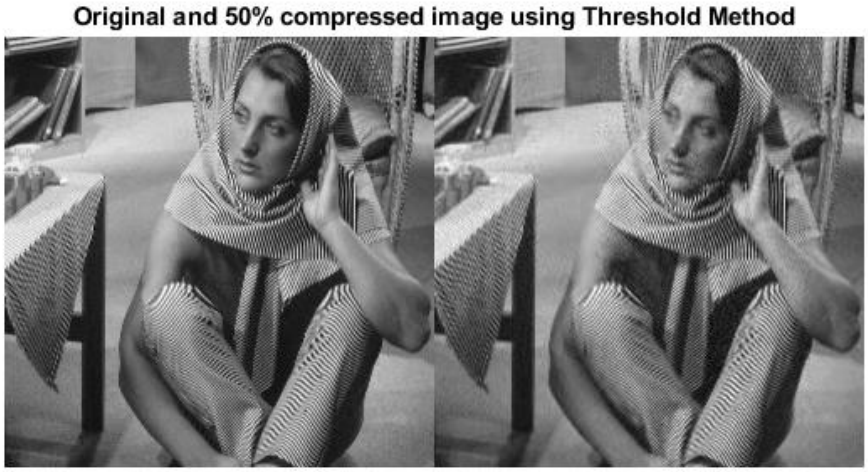
Με την μέθοδο κατωφλιού, για κάθε παράθυρο 32*32 της εικόνας, βρίσκουμε την μέση τιμή των συντελεστών του και κρατάμε μόνο αυτούς που βρίσκονται πάνω από ένα κατώφλι. Το κατώφλι προσαρμόζεται ανάλογα με την τιμή του r = [0.05 – 0.5]. Έτσι, κρατάμε πληροφορία υψηλότερων συχνοτήτων, η οποία με την μέθοδο ζώνης μηδενίζεται.

Κώδικας:

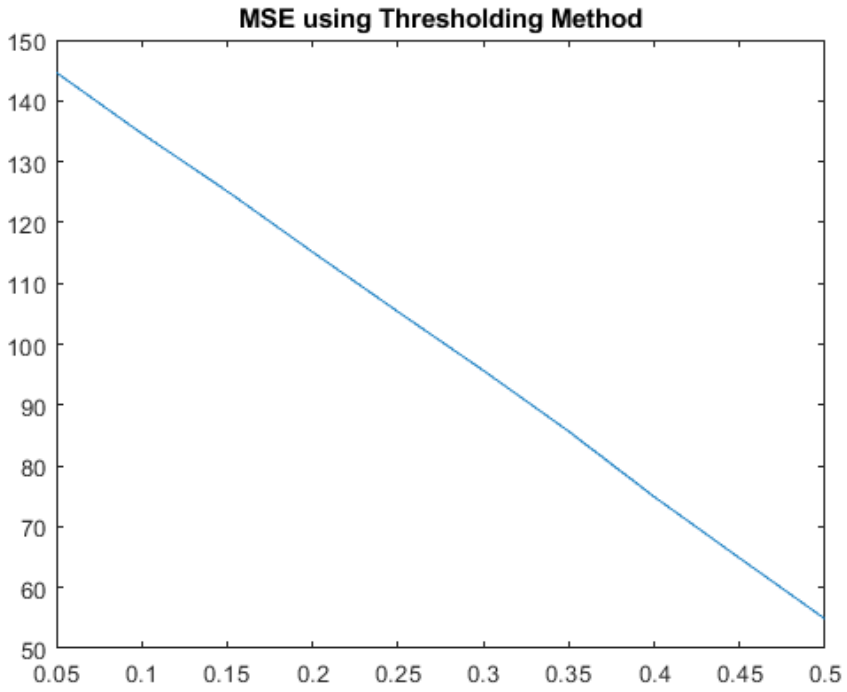
```
MSE2 = zeros(0.5/0.05,1);

for r=0.05:0.05:0.5


    thresholding = @(block_struct) threshold(r,block_struct.data);
    threshBlocks = blockproc(dct_transform,windowSize,thresholding);
    IDCT = @(block_struct) my2D_IDCT(block_struct.data); %Inverse DCT
    restoredImage = blockproc(threshBlocks,windowSize,IDCT); %Apply inverse DCT to
every 32x32 block

    if ((r == 0.05 ) || (r == 0.5))
        figure;
        imshowpair(I,restoredImage,'montage')
        title(sprintf('Original and %d%% compressed image using Threshold
Method',uint8(100 - r*100)))
    end
    MSE2(round(r/0.05)) = sum(sum((double(I)-restoredImage).^2)) / (size(I,1) *
size(I,2));

end


%Plotting the Mean Square Error for different information percentage
f3 = figure;
inform_percent = 0.05:0.05:0.5;
plot(inform_percent,MSE2)
title("MSE using Thresholding Method");
```



Εικόνα 9 95% συμπίεση με την μέθοδο κατωφλιού



Εικόνα 10 50% συμπίεση με την μέθοδο κατωφλιού



Εικόνα 11 Μέσο τετραγωνικό σφάλμα με την μέθοδο κατωφλιού

Παρατηρούμε αρχικά την μεγάλη διαφορά που έχει η συμπίεση για r = 5%, δηλαδή για συμπίεση της εικόνας κατά 95%, με την μέθοδο ζώνης και την μέθοδο κατωφλιού. Η μέθοδος κατωφλιού έχει πολύ καλύτερο αποτέλεσμα, κάτι που φαίνεται και στο ότι για τα ίδια ποσοστά συμπίεσης, έχει αρκετά μικρότερο MSE από την μέθοδο ζώνης.

3. Βελτίωση εικόνας - Φιλτράρισμα Θορύβου

1. **Στην εικόνα tiger.mat προσθέστε λευκό Gaussian θόρυβο μηδενικής μέσης τιμής και τέτοιας διασποράς ώστε η τελική εικόνα να έχει λόγο σήματος προς θόρυβο ίσο με $15dB$. Χρησιμοποιείστε το φίλτρο "κινούμενου μέσου" (moving average filter) και το "φίλτρο διαμέσου" (median filter) για να αφαιρέσετε το θόρυβο από την εικόνα.**

Υπολογίζουμε τον λευκό Gaussian θόρυβο που θα προσθέσουμε στην εικόνα, πολλαπλασιάζοντας ένα τυχαίο πίνακα διαστάσεων ίσων με την εικόνα μας, με την απόκλιση να είναι τέτοια ώστε το SNR (Signal to Noise Ratio) να είναι 15 dB. Η απόκλιση δίνεται από τον τύπο:

$$\sigma = \sqrt{\frac{P_{signal}}{10^{SNR/10}}}$$

όπου $P_{signal} = \sum_{(x,y)=(0,0)}^{(M,N)} \frac{tiger(x,y)^2}{M*N}$ και M,N οι διαστάσεις της εικόνας tiger.mat.

Κώδικας:

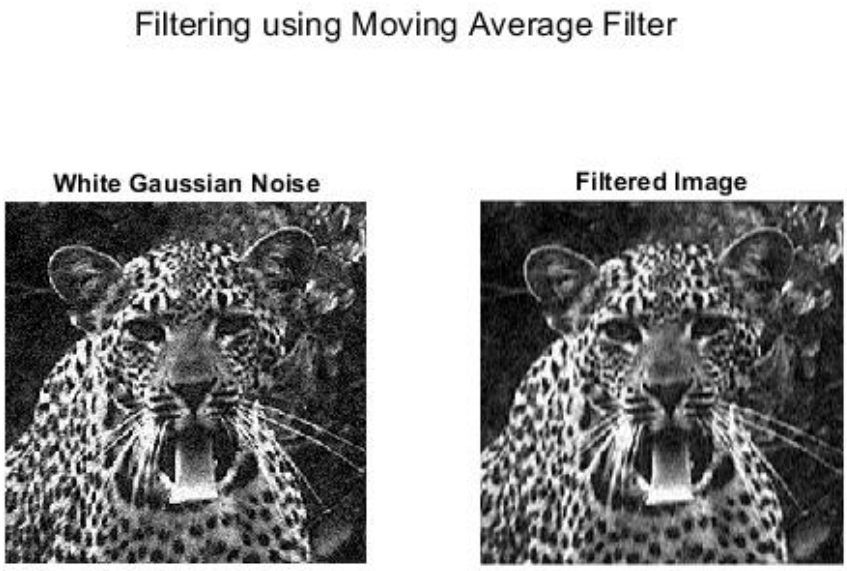
```
%% Exercise 3 - Noise Filtering
%% 1 - White Gaussian Noise
close all; clear;

storedStruct = load("tiger.mat");
tiger = storedStruct.tiger;
figure;
imshow(tiger,[]);
title("Original Image");

% Get size of image
[rows,columns,noOfColorChannels] = size(tiger);

% Adding white Gaussian noise such that SNR = 15 dB.
%Noise Formula
%dNoise = d + sqrt(10^(SNR/10)) * randomPixels = variance * randomPixels
%SNR = Psignal/Pnoise, for Psignal I use the mean intensity of the image.
SNR = 15;

Psignal = sum(sum(tiger(:,:).^ 2)) / (rows * columns);

%Calculating noise variance
variance = sqrt(Psignal / 10 ^(SNR/10));

%Additive White Gaussian Noise calculation and adding to signal
AWGN(:,:) = variance * randn(rows,columns);
noisyTiger = tiger + AWGN;
```

```
[PSNR, actualSNR] = psnr(noisyTiger,tiger);

figure;
imshowpair(tiger,noisyTiger,'montage');
title("Orignal Image and image with White Gaussian Noise")
```

Για να απομακρύνουμε τον θόρυβο από την εικόνα, θα χρησιμοποιήσουμε πρώτα το φίλτρο «κινούμενου μέσου» (moving average filter – MA). Το φίλτρο αυτό είναι μια μάσκα από άσσους, η οποία ολισθαίνεται πάνω στην εικόνα και αντικαθιστά το κεντρικό pixel της μάσκας με την μέση τιμή των γειτονικών pixel. Στη συγκεκριμένη περίπτωση, δημιουργούμε ένα MAF kernel μεγέθους 3 και το εφαρμόζουμε στην εικόνα με την συνάρτηση filter της MATLAB.

Κώδικας:

```
%% Filtering AWGN using Moving Average Filter
%Creating a MAF using fspecial and one using filter
windowSize = 3;
MAF_kernel = ones(windowSize,1)/windowSize;
MAF_filtered = filter(MAF_kernel,1,noisyTiger);

f2 = figure;
subplot(1,2,1)
imshow(noisyTiger);
title('White Gaussian Noise');
subplot(1,2,2);
imshow(MAF_filtered);
title('Filtered Image');
sgtitle('Filtering using Moving Average Filter');
```

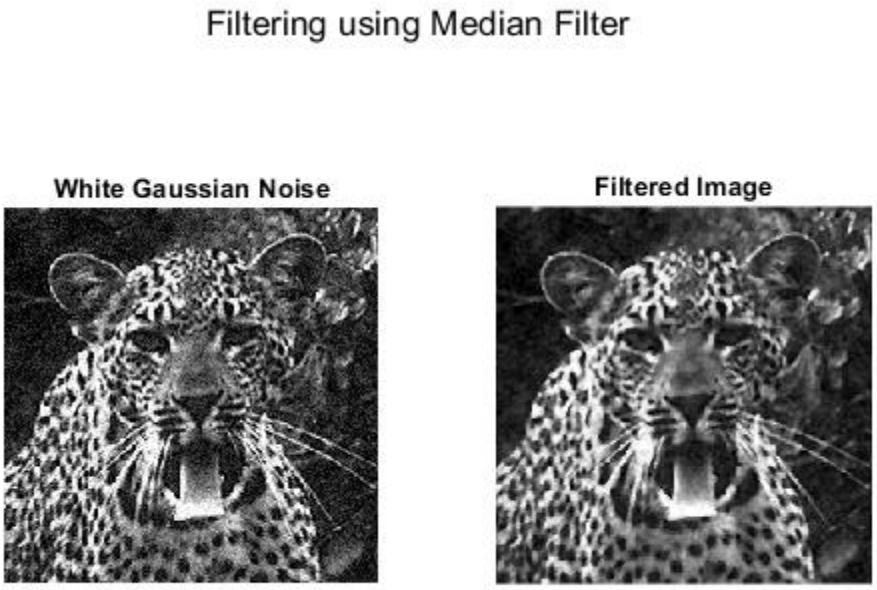


Εικόνα 12 Φιλτράρισμα λευκού θορύβου με Moving Average Filter

Έπειτα, χρησιμοποιούμε ένα άλλο φίλτρο, το φίλτρο διαμέσου, το αντικαθιστά κάθε pixel με την διάμεσο της 3*3 γειτονίας γύρω του. Για την υλοποίηση του φίλτρου χρησιμοποιούμε την συνάρτηση med2filt της MATLAB.

Κώδικας:

```
%% Filtering AWGN using Median Filter
f3 = figure;
Median_Filtered = medfilt2(noisyTiger);
subplot(1,2,1)
imshow(noisyTiger);
title('White Gaussian Noise');
subplot(1,2,2);
imshow(Median_Filtered);
title('Filtered Image');
sgtitle('Filtering using Median Filter');
```



Εικόνα 13 Φιλτράρισμα λευκού θορύβου με Median Filter

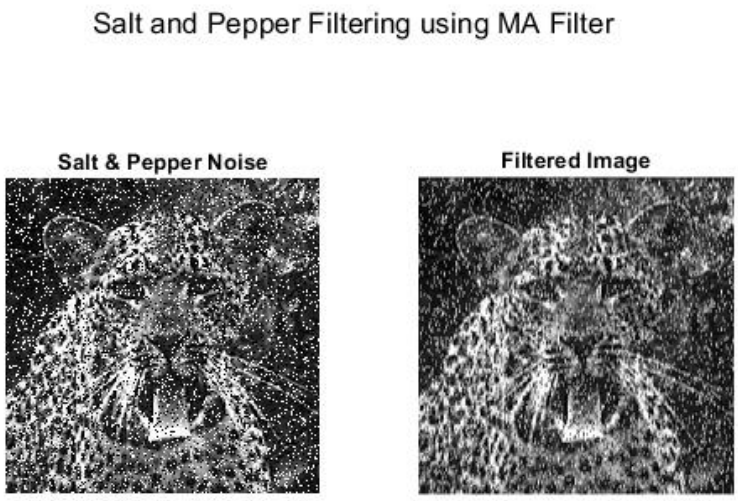
Παρατηρούμε ότι για την αφαίρεση του λευκού θορύβου, το φίλτρο κινούμενου μέσου είναι το καλύτερο καθώς κάνει καλύτερο smoothing σε περιοχές μεγάλης διασποράς, διαχειρίζεται δηλαδή καλύτερα τον λευκό θόρυβο.

2. Στην εικόνα tiger.mat προσθέστε κρουστικό θόρυβο σε ποσοστό 20% και χρησιμοποιείστε τα ίδια φίλτρα για την αφαίρεση του θορύβου.

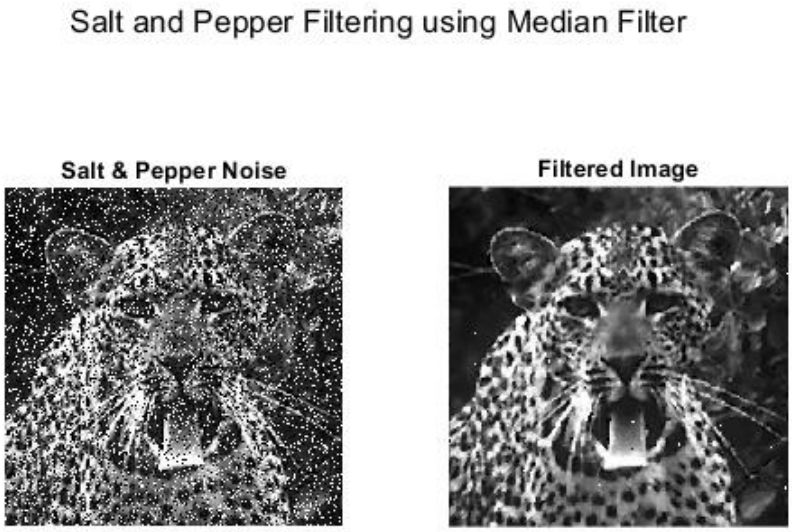
Προσθέτουμε κρουστικό θόρυβο στην εικόνα σε ποσοστό 20% με την συνάρτηση imnoise της MATLAB, με το όρισμα ‘salt & pepper’ που αντικαθιστά τυχαία το 20% των pixel της εικόνας μας είτε με 0 είτε με 255 (λευκό – μαύρο). Έπειτα απομακρύνουμε τον θόρυβο χρησιμοποιώντας τα ίδια φίλτρα με πριν.

Κώδικας:

```
%% 2 Salt and Pepper Noise

p = 0.2; %20 percent of pixels altered

noisyTiger2 = imnoise(tiger,'salt & pepper',0.2);

% Filter using MAF
MAF_filtered2 = filter(MAF_kernel,1,noisyTiger2);

%Filter using Median Filter
Median_Filtered2 = medfilt2(noisyTiger2);

f4 = figure;
subplot(1,2,1)
imshow(noisyTiger2);
title('Salt & Pepper Noise');
subplot(1,2,2);
imshow(MAF_filtered2);
title('Filtered Image');
sgtitle('Salt and Pepper Filtering using MA Filter');


f5 = figure;
subplot(1,2,1)
imshow(noisyTiger2);
title('Salt & Pepper Noise');
subplot(1,2,2);
imshow(Median_Filtered2);
title('Filtered Image');
sgtitle('Salt and Pepper Filtering using Median Filter');
```



Εικόνα 14 Φιλτράρισμα κρουστικού θορύβου με Moving Average Filter



Εικόνα 15 Φιλτράρισμα κρουστικού θορύβου με Median Filter

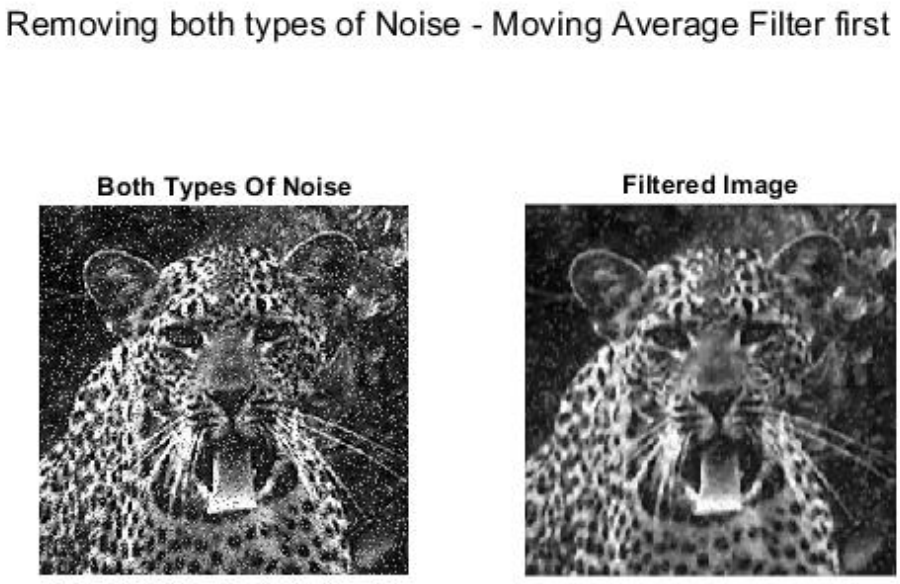
Παρατηρούμε ότι το Median Filter είναι πολύ καλύτερο στο φιλτράρισμα του κρουστικού θορύβου, καθώς διαχειρίζεται πολύ καλά τις απότομες αλλαγές στην ένταση, οι οποίες θα επηρέαζαν πάρα πολύ τις διαμέσους που υπολογίζει το Moving Average Filter.

3. Τέλος στην εικόνα tiger.mat προσθέστε λευκό Gaussian θόρυβο μηδενικής μέσης τιμής και τέτοιας διασποράς ώστε η τελική εικόνα να έχει $15dB$, καθώς και κρουστικό θόρυβο σε ποσοστό 20%. Δοκιμάστε να ανακτήσετε την αρχική εικόνα χρησιμοποιώντας διαδοχική εφαρμογή του φίλτρου κινούμενου μέσου και του φίλτρου μεσαίου, με την κατάλληλη σειρά που εσείς θα επιλέξετε.

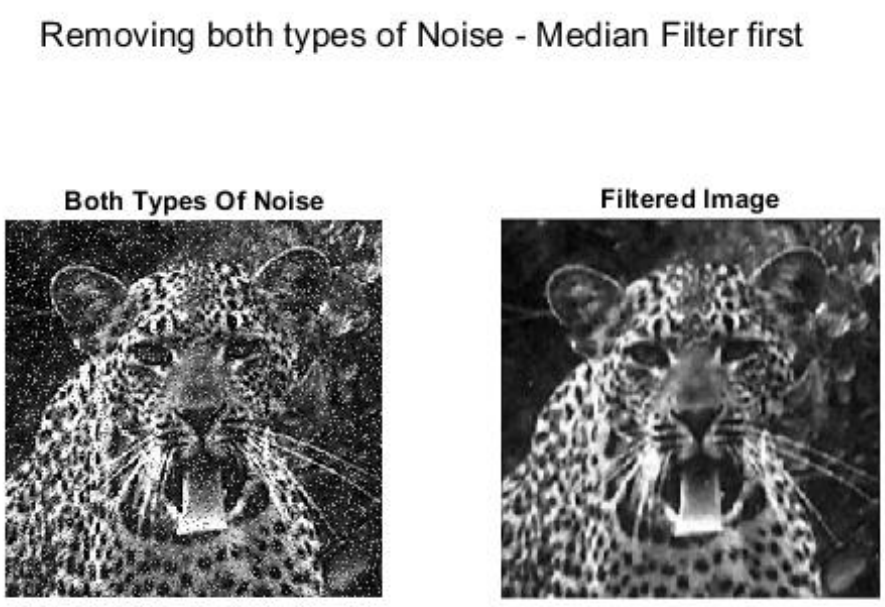
Προσθέτουμε και τους δύο τύπους θορύβου στην εικόνα και φιλτράρουμε πρώτα με το Moving Average φίλτρο και μετά με το Median φίλτρο. Έπειτα κάνουμε το ανάποδο.

Κώδικας:

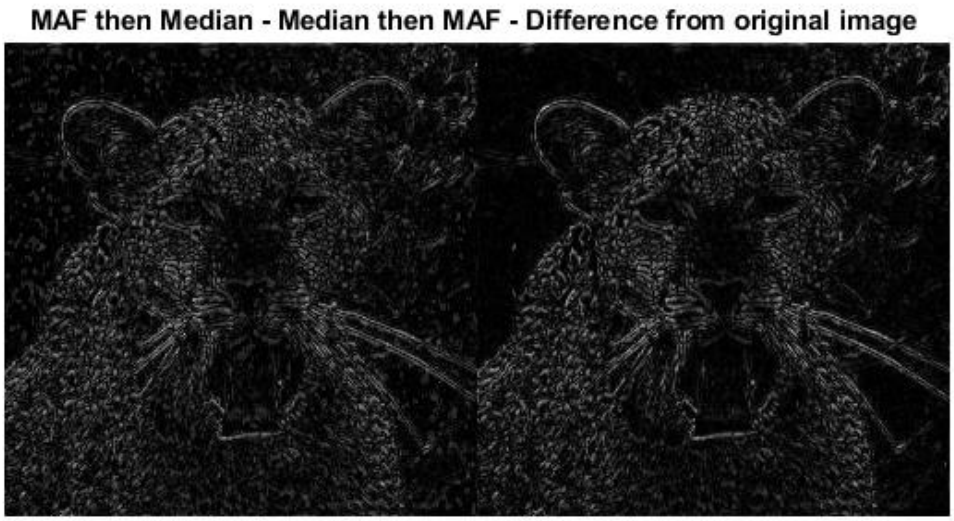
```
%% 3 - Both Noises

BothNoises = (noisyTiger + noisyTiger2) / 2;


%Using Moving Average Filter First
MAF_first = filter(MAF_kernel,1,BothNoises);
MAF_then_Median = medfilt2(MAF_first);

%Using Median Filter First
Median_first = medfilt2(BothNoises);
Median_then_MAF = filter(MAF_kernel,1,Median_first);


f6 = figure;
subplot(1,2,1)
imshow(BothNoises);
title('Both Types Of Noise');
subplot(1,2,2);
imshow(MAF_then_Median);
title('Filtered Image');


sgtitle("Removing both types of Noise - Moving Average Filter first");


f7 = figure;
subplot(1,2,1)
imshow(BothNoises);
title('Both Types Of Noise');
subplot(1,2,2);
imshow(Median_then_MAF);
title('Filtered Image');

sgtitle("Removing both types of Noise - Median Filter first");


%Checking which order of filtering is best by finding out the sum of
%absolute differences between the filtered images and the original one
f8 = figure;
imshow([imabsdiff(tiger,MAF_then_Median) imabsdiff(tiger,Median_then_MAF)]);
title("MAF then Median - Median then MAF - Difference from original image ");
```



Εικόνα 16 Φιλτράρισμα πρώτα με το Moving Average φίλτρο και μετά με το Median



Εικόνα 17 Φιλτράρισμα πρώτα με το Median φίλτρο και μετά με το Moving Average



Εικόνα 18 Διαφορά της αρχικής εικόνας με τα δύο αποτελέσματα φιλτραρίσματος. Όπου η εικόνα είναι μαύρη, υπάρχει διαφορά 0 με την αρχική εικόνα. Παρατηρούμε ότι με το Median πρώτα οι επιφάνειες πίσω από την τίγρη έχουν φιλτραριστεί πολύ καλύτερα.

Παρατηρούμε ότι το καλύτερο αποτέλεσμα το πετυχαίνουμε όταν εφαρμόσουμε το Median πρώτα, το οποίο είναι καλό στο να αφαιρεί τον κρουστικό θόρυβο και στην συνέχεια το Moving Average, γιατί αν εφαρμόσουμε το Moving Average φίλτρο πρώτα θα προσθέσουμε τις υψηλές τιμές των pixel που έχουν κρουστικό θόρυβο στα γειτονικά τους pixel που δεν έχουν.

5. Αποκατάσταση Εικόνας – Αποσυνέλιξη

Θεωρήστε την lenna.jpg και πραγματοποιήστε τα παρακάτω:

Μέρος Α

Αρχικά υποβαθμίστε τη δοθείσα εικόνα με λευκό θόρυβο Gauss ώστε να έχει λόγο σήματος προς θόρυβο (SNR) ίσο με $10dB$. Χρησιμοποιώντας το φίλτρο Wiener απομακρύνετε το θόρυβο με τους δύο παρακάτω διαφορετικούς τρόπους :

1. Αξιοποιώντας τη γνώση που έχετε σχετικά με την ισχύ του θορύβου.
2. Υποθέτοντας ότι δεν γνωρίζετε την ισχύ του θορύβου.

Σε κάθε περίπτωση σχολιάστε και συγκρίνετε τα αποτελέσματα.

Προσθέτουμε λευκό Gaussian θόρυβο τέτοιο ώστε το τελικό SNR να είναι 10 dB, με παρόμοιο τρόπο όπως στην άσκηση 3.

Κώδικας:

```
%% Image Restoration - Deconvolution
%% Part A
%% Adding Noise to Image
clear; close all;
lenna = imread("lenna.jpg");
%Keeping only one channel
lenna =lenna(:,:,1);
f1 = figure;
imshow(lenna);
title("Original Image");

[rows,columns] = size(lenna);

% Adding white Gaussian noise such that SNR = 10 dB.
%Noise Formula
%dNoise = d + sqrt(10^(SNR/10)) * randomPixels = variance * randomPixels
%SNR = Psignal/Pnoise, for Psignal I use the mean intensity of the image.
SNR = 10;

Psignal = sum(sum(double(lenna(:,:)).^ 2)) / (rows * columns);

%Calculating noise variance
variance = sqrt(Psignal / 10 ^(SNR/10));

%Additive White Gaussian Noise calculation and adding to signal
AWGN(:,:) = variance * randn(rows,columns);
noisyLenna = double(lenna) + AWGN;
% g = f + n;
```

Ο θόρυβος είναι προσθετικός, άρα η τελική εικόνα g(x,y) = f(x,y) + n(x,y).

Με την χρήση του φίλτρου Wiener θα απομακρύνουμε τον θόρυβο. Ο υπολογισμός του φίλτρου Wiener δίνεται από τον τύπο:

$$H_{\text{Wiener}}(u,v) = \frac{P_{fg}(u,v)}{P_g(u,v)} = \frac{P_f(u,v)}{P_f(u,v) + P_n(u,v)}$$

Η διαδικασία αποκατάστασης της εικόνας με το φίλτρο Wiener είναι η εξής:

- Zero-padding στην g(x,y) -> g_{ext}(x,y) (π.χ. 2Nx2N)
- Αφαίρεση και υπολογισμός του $G_{\text{ext}}(u,v)$ με εφαρμογή 2D-FFT
- Εύρεση του $H_{\text{Wiener}}(u,v)$στις extended διαστάσεις (π.χ. 2Nx2N)
- Hadamard (pointwise) product των $G_{\text{ext}}(u,v)$ και $H_{\text{WienerExt}}(u,v)$
- Υπολογισμός της $f_{\text{ext}}(x,y)$ με 2D-IFFT στο $G_{\text{ext}}(u,v) \cdot H_{\text{WienerExtended}}(u,v)$
- Κράτημα της περιοχής ενδιαφέροντος

Αρκεί λοιπόν να υπολογίζουμε την $H_{\text{Wiener}}(u,v)$. Γνωρίζουμε ότι $P_g(u,v) = \frac{|G(u,v)|^2}{\text{N*N}}$ και ότι $P_f(u,v) = P_g(u,v) - P_n(u,v)$. Άρα αρκεί να βρούμε το $P_n(u,v)$. Εάν γνωρίζουμε τα χαρακτηριστικά του θορύβου (ισχύς), μπορούμε να υπολογίσουμε το $P_n(u,v)$ και απλά να το αφαιρέσουμε από το $P_g(u,v)$.

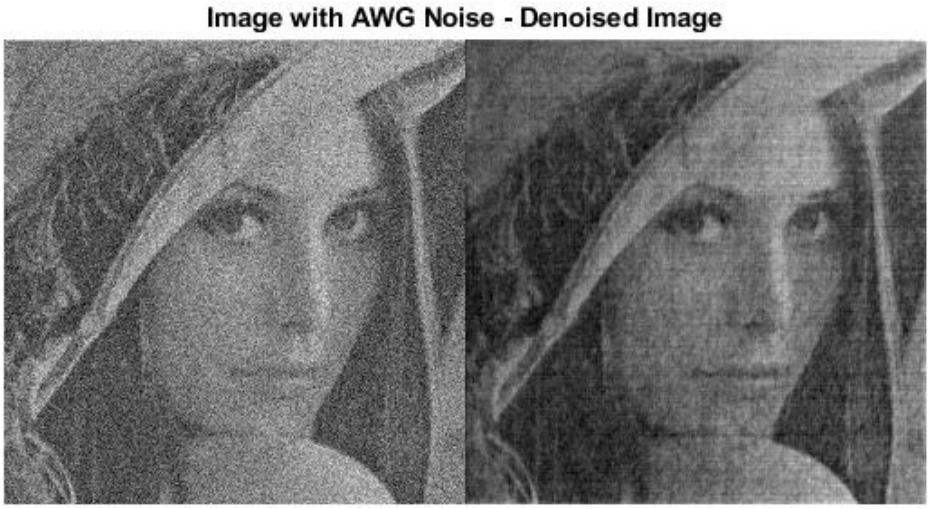
Εάν δεν γνωρίζουμε κάτι για τον θόρυβο, θα υπολογίσουμε το $P_{\text{n}}(u,v)$ από την μέση τιμή ενός παραθύρου που καλύπτει κάποιες από τις υψηλές συχνότητες της εικόνας g. Εδώ επιλέγουμε ένα παράθυρο [200:255,200:255]

<u>Κώδικας:</u>

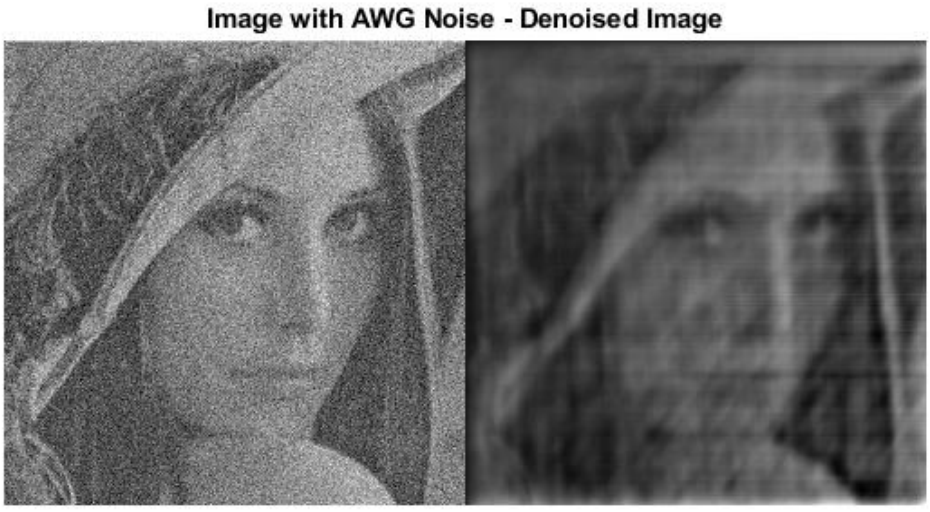
```
%% Denoising with Wiener Filter

% Using the knowledge we have about the noise
% Power of noise signal
Pnoise = abs(fft2(AWGN)).^2 /(rows*columns) ;

%Power of image signal + noise signal
PnoisyImage = abs(fft2(noisyLenna)).^2 /(rows*columns) ;

%Power of image signal without noise
Pimage = PnoisyImage - Pnoise;

%Calculating Wiener Filter
Hwiener = uint8(255 * (Pimage ./ PnoisyImage));
hwiener = ifft2(Hwiener);

%Zero padding for 2N*2N
```

```
ExtendedNoisyImage = zeros(2*rows,2*columns);
ExtendedNoisyImage(1:rows,1:columns) = noisyLenna;

Gextended = fft2(ExtendedNoisyImage);

PaddedFilter = zeros(2*rows,2*columns);
PaddedFilter(1:rows,1:columns) = hwiener;
HwienerExtended = fft2(PaddedFilter);

%Calculating Pointwise product of Gextended and HwienerExtended
ExtendedImage = HwienerExtended .* Gextended;

%IFFT2 for calculating final denoised image
finalExtendedImage = ifft2(ExtendedImage);
finalImage = finalExtendedImage(1:rows,1:columns);

f2 = figure('Name','Using the knowledge about the noise');
imshowpair(noisyLenna,finalImage,'montage');
title("Image with AWG Noise - Denoised Image");
%% Not knowing anything about the noise
% Using the mean value of a window that covers some of the high
% frequencies of the noisy image G


G = fft2(noisyLenna);

hFWindow = G(200:255,200:255);
Pnoise = mean(abs(hFWindow(:)));

%Power of image without noise
Pimage = PnoisyImage - Pnoise;

%Calculating Wiener Filter
Hwiener = uint8(255 * (Pimage ./ PnoisyImage));
hwiener = ifft2(Hwiener);

%Zero padding for 2N*2N

ExtendedNoisyImage = zeros(2*rows,2*columns);
ExtendedNoisyImage(1:rows,1:columns) = noisyLenna;

Gextended = fft2(ExtendedNoisyImage);

PaddedFilter = zeros(2*rows,2*columns);
PaddedFilter(1:rows,1:columns) = hwiener;
HwienerExtended = fft2(PaddedFilter);

%Calculating Pointwise product of Gextended and HwienerExtended
ExtendedImage = HwienerExtended .* Gextended;

%IFFT2 for calculating final denoised image
finalExtendedImage = ifft2(ExtendedImage);
finalImage = finalExtendedImage(1:rows,1:columns);

f2 = figure('Name','Not knowing the noise');
```

```
imshowpair(noisyLenna,finalImage,'montage');
title("Image with AWG Noise - Denoised Image");
```



Εικόνα 19 Φιλτράρισμα εικόνας γνωρίζοντας την ισχύ του θορύβου



Εικόνα 20 Φιλτράρισμα εικόνας με άγνωστη την ισχύ του θορύβου

Παρατηρούμε ότι έχουμε πολύ καλύτερο αποτέλεσμα όταν γνωρίζουμε την ισχύ του θορύβου.

Μέρος Β

Εφαρμόστε το μετασχηματισμό που υλοποιείται στο αρχείο psf.p, το οποίο υλοποιεί την point spread function (PSF) ενός άγνωστου συστήματος καταγραφής εικόνων. Ο μετασχηματισμός εφαρμόζεται με την εντολή $Y = psf(X)$, όπου X η εικόνα εισόδου και Y η εικόνα εξόδου. Στη συνέχεια πραγματοποιήστε την ακόλουθη επεξεργασία:

- Χρησιμοποιώντας κατάλληλη τεχνική υπολογίστε την κρουστική απόκριση του αγνώστου συστήματος και απεικονίστε την απόκριση συχνότητας αυτής.

- Εφαρμόστε την τεχνική του αντίστροφου φίλτρου στο πεδίο της συχνότητας με χρήση κατωφλιού, ώστε να αντιμετωπίσετε το θόλωμα που προκύπτει. Απεικονίστε το μέσο τετραγωνικό σφάλμα (Mean Squared Error, MSE) μεταξύ της αρχικής εικόνας (χωρίς θόρυβο) και του αποτελέσματος για διάφορες τιμές κατωφλιού.

- Σχολιάστε τι συμβαίνει όταν δεν γίνεται χρήση κατωφλιού.

Στην συγκεκριμένη περίπτωση έχουμε υποβάθμιση λόγω συνέλιξης, που συχνά συναντάται στα συστήματα καταγραφής εικόνων, όπου το σύστημα υποβάθμισης b θεωρείται γνωστό.

$$g(x,y) = f(x,y) ** b(x,y),$$

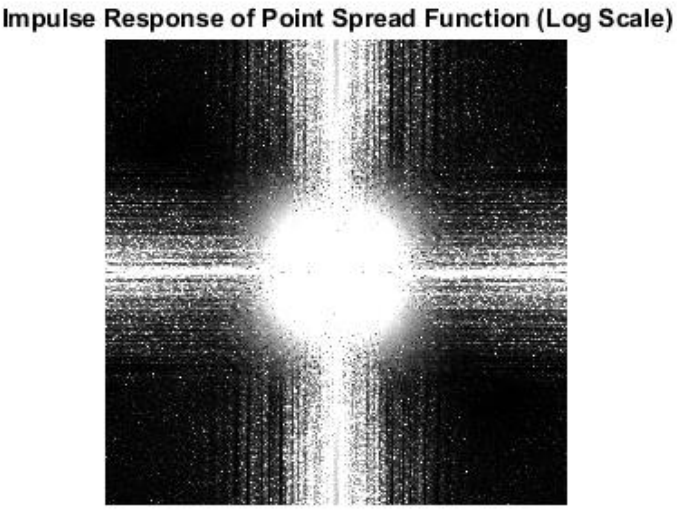
$$\mathrm{G(u,v)} = \mathrm{F(u,v)} * \mathrm{B(u,v)}$$

Εφαρμόζουμε τον θόρυβο στην εικόνα και πηγαίνουμε στο πεδίο της συχνότητάς με τον fft2 έτσι ώστε η απόκριση συχνότητας B(u,v) να δίνεται από τον τύπο $\mathrm{B(u,v)} = \mathrm{G(u,v)} * \mathrm{F(u,v)}$

Κώδικας:

```
%% Part B - Denoising random noise

%Applying Point Spread Function to image
psfImage = psf(lenna);

f3 = figure('Name','Image after PSF');
imshow(psfImage,[]);
title("Image after PSF");
freqPSF = fft2(psfImage);
freqLenna = fft2(lenna);
PSFimpulseResponse = freqPSF ./ freqLenna;


f4 = figure('Name','Impulse response of PSF');
imshow(log(1+abs(fftshift(PSFimpulseResponse))));
title('Impulse Response of Point Spread Function (Log Scale)');
```

Εικόνα 21 Η εικόνα μετά την υλοποίηση του μετασχηματισμού PSF



Εικόνα 22 Η απόκριση συχνότητας του μετασχηματισμού PSF

Για να απομακρύνουμε τον θόρυβο, εφαρμόζουμε την τεχνική του αντίστροφου φίλτρου $\mathrm{H}(u,v) = 1/\mathrm{B}(u,v)$ με χρήση κατωφλιού για να αντιμετωπίσουμε το θόλωμα που προκύπτει.

$$F(u,v) = \frac{1}{B(u,v)} * G(u,v)$$

Εάν έχουμε και θόρυβο τότε έχουμε:

$$\mathrm{H}(u,v) * \mathrm{G}(u,v) \;=\; F(u,v) \;+\; \mathrm{N}(u,v) \,/\, \mathrm{B}(u,v)$$

Επειδή το $\mathrm{B}(u,v)$ μπορεί να πάρει τιμές που προσεγγίζουν το 0, σε κάποια σημεία ο θόρυβος θα προσεγγίσει το άπειρο, ενισχύοντας την επίδραση του στην εικόνα. Για να το αποφύγουμε αυτό, θα ορίσουμε ένα κατώφλι γ, πάνω από το οποίο το φίλτρο μας θα ισούται με το αντίστροφο φίλτρο.

$$\mathrm{H}(u,v) = \begin{cases} 1/B(u,v), & 1 \,/\, |B(u,v)| < \gamma \\ \gamma\,|B(u,v)|/B(u,v), & 1 \,/\, |B(u,v)| \geq \gamma \end{cases}$$

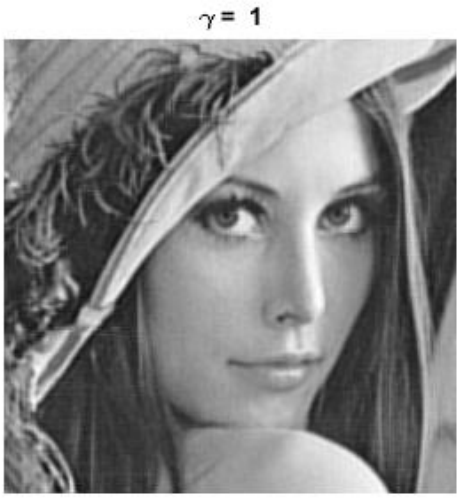
Όσο αυξάνεται το κατώφλι τόσο περισσότερο θα προσεγγίζει το φίλτρο H το αντίστροφο φίλτρο.

Για διαφορετικές τιμές του κατωφλιού γ, υπολογίζουμε το Mean Squared Error μεταξύ της αρχικής εικόνας και του αποτελέσματος.

Κώδικας:

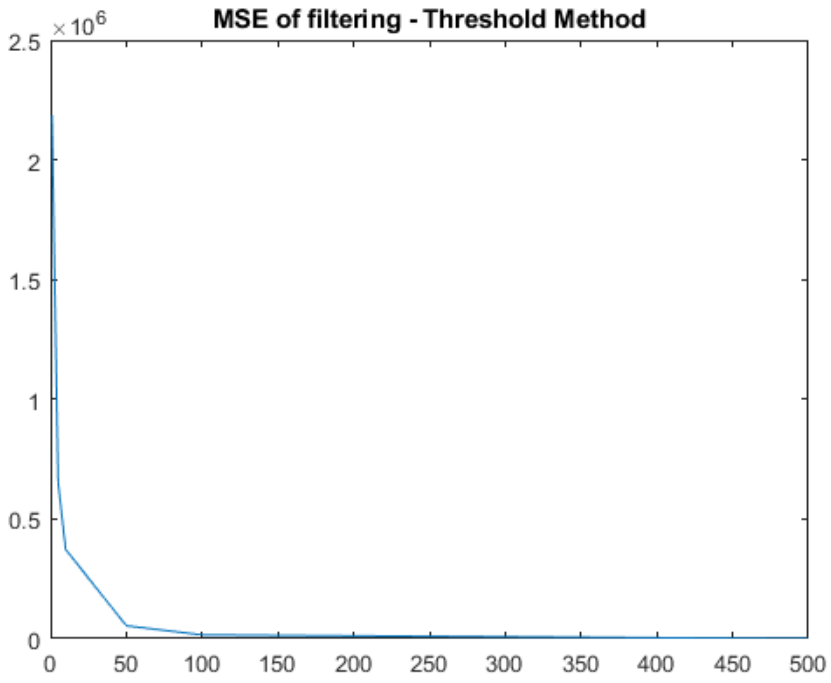
```
%% Inverse Filtering with threshold

thresholdValues=[1 5 10 50 100 500];
inverseFilter=zeros(rows,columns);
MSE = zeros(size(thresholdValues));
cnt = 0;
%For different γ values
for g = thresholdValues
    cnt = cnt + 1;
    for u=1:rows
        for v=1:columns
            if (1/abs(PSFimpulseResponse(u,v)) < g)
                inverseFilter(u,v) = 1/PSFimpulseResponse(u,v);
            else

inverseFilter(u,v)=g*abs(PSFimpulseResponse(u,v))/PSFimpulseResponse(u,v);
            end
        end
    end

    restoredImageFreq = inverseFilter.* freqPSF ;
    restoredImage = ifft2(restoredImageFreq);

    figure;
    imshow(uint8(restoredImage))
    title("\gamma =  " + g);
    MSE(cnt) = sum(sum((double(lenna)-double(restoredImage)).^2)) / rows * columns;
```

```
end

%Plotting the Mean Square Error for different threshold values
f3 = figure;
plot(thresholdValues,MSE)
title("MSE of filtering - Threshold Method");
```



Εικόνα 23 Αποτέλεσμα για κατώφλι = 1

Εικόνα 24 Αποτέλεσμα για κατώφλι = 500



Εικόνα 25 MSE για γ = [1 5 10 50 100 500]

Παρατηρούμε ότι όσο αυξάνεται το κατώφλι, τόσο μικρότερο MSE έχουμε, άρα τόσο καλύτερο αποτέλεσμα αποκατάστασης της εικόνας.

Όταν δεν χρησιμοποιούμε καθόλου κατώφλι έχουμε:

Κώδικας:

```
%% Inverse Filtering Without Threshold

inverseFilter = 1 ./ PSFimpulseResponse;

restoredImageFreq2 = inverseFilter.* freqPSF ;
restoredImage2 = ifft2(restoredImageFreq2);

f4 = figure;
imshow(restoredImage2,[]);
MSE_final = sum(sum((double(lenna)-double(restoredImage2)).^2)) / rows * columns;
title("Inverse Filtering - No Threshold");
```

Εικόνα 26 Αποτέλεσμα χωρίς κατώφλι

Παρατηρούμε ότι χωρίς κατώφλι έχουμε το καλύτερο αποτέλεσμα καθώς προσεγγίζουμε τέλεια το αντίστροφο φίλτρο.

6. Ανάκτηση εικόνας από βάση δεδομένων

Στόχος αυτού του θέματος είναι η εξοικείωση σας με τη διαδικασία της ανάκτησης εικόνων από μία βάση

δεδομένων. Για τον σκοπό αυτό, σας δίνεται μια βάση εικόνων (φάκελος: Database), που είναι ουσιαστικά ένας φάκελος με έναν αριθμό από εικόνες, από τις οποίες ορισμένες μοιάζουν μεταξύ τους και θα πρέπει να αναπτύξετε κατάλληλες ρουτίνες για την ανάκτησή τους, εφαρμόζοντας τεχνικές αναπαράστασης εικόνων.

Ουσιαστικά, το πρόγραμμα που θα υλοποιήσετε πρέπει να επιτελεί την εξής βασική λειτουργία: Δέχεται ως είσοδο μία εικόνα, υπολογίζει (σύμφωνα με κάποια μετρική που έχετε επιλέξει) το βαθμό ομοιότητας των υπολοίπων εικόνων ως προς αυτήν και επιστρέφει την εγγύτερη ως προς αυτήν εικόνα.

Στο φάκελο **test** δίνονται 10 εικόνες με τις οποίες θα πειραματιστείτε για τον έλεγχο των συστημάτων ανάκτησης που θα κατασκευάσετε. Δίνοντας μία εικόνα από τον φάκελο test ως είσοδο σε αυτό το σύστημα θα πρέπει αυτό να την εντοπίζει στην βάση δεδομένων.

Ζητούμενα:

1. Σύστημα ανάκτησης εφαρμόζοντας ως τεχνική αναπαράστασης των εικόνων τα ιστογράμματα. Συγκεκριμένα τόσο για τις εικόνες στην βάση δεδομένων, όσο και για τις εικόνες στον test φάκελο, θα πρέπει να υπολογιστούν τα ιστογράμματα τους και η σύγκριση θα πρέπει να πραγματοποιηθεί σύμφωνα με αυτά. Ως έξοδο του συστήματος θεωρείται η εικόνα με το εγγύτερο ιστόγραμμα ως προς το ιστόγραμμα της εικόνας εισόδου. Εξάγετε το ποσοστό επιτυχούς ανάκτησης.

2. Σύστημα ανάκτησης εφαρμόζοντας ως τεχνική αναπαράστασης των εικόνων τον μετασχηματισμό DCT. Για τις δοθείσες εικόνες (εικόνες στον test φάκελο και στην βάση δεδομένων) θα πρέπει να υπολογιστεί ο 2D-DCT μετασχηματισμός τους και η σύγκριση των εικόνων να πραγματοποιηθεί στο πεδίο DCT. Χρησιμοποιώντας την μέθοδο κατωφλίου διατηρείστε ένα συγκεκριμένο ποσοστό των συντελεστών του DCT και εξάγεται τα ποσοστά επιτυχούς ανάκτησης. Ενδεικτικά παρουσιάστε τα αποτελέσματα για το 10%, 50% και το 100% του συνόλου των συντελεστών DCT.

Για την αναπαράσταση των εικόνων μας, χρησιμοποιούμε την συνάρτηση imageDatastore της MATLAB, η οποία χρησιμοποιείται για την αναπαράσταση βάσεων δεδομένων με εικόνες. Με τον ίδιο τρόπο αναπαριστάμε και τις εικόνες test, για ευκολία ανάκτησης καθεμίας για την δοκιμή των τεχνικών ανάκτησης.

Στο πρώτο σύστημα ανάκτησης, αναπαριστούμε κάθε εικόνα ως το ιστόγραμμα της, με την συνάρτηση imhist της MATLAB. Χρειάζεται να κρατήσουμε πόσα counts έχει το κάθε χρώμα της εικόνας και τα αποθηκεύουμε σε μια μεταβλητή, έτσι ώστε να μη χρειάζεται να ξανακάνουμε την αναπαράσταση σε ιστόγραμμα όλης της βάσης εικόνων κάθε φορά που αναζητούμε μια εικόνα.

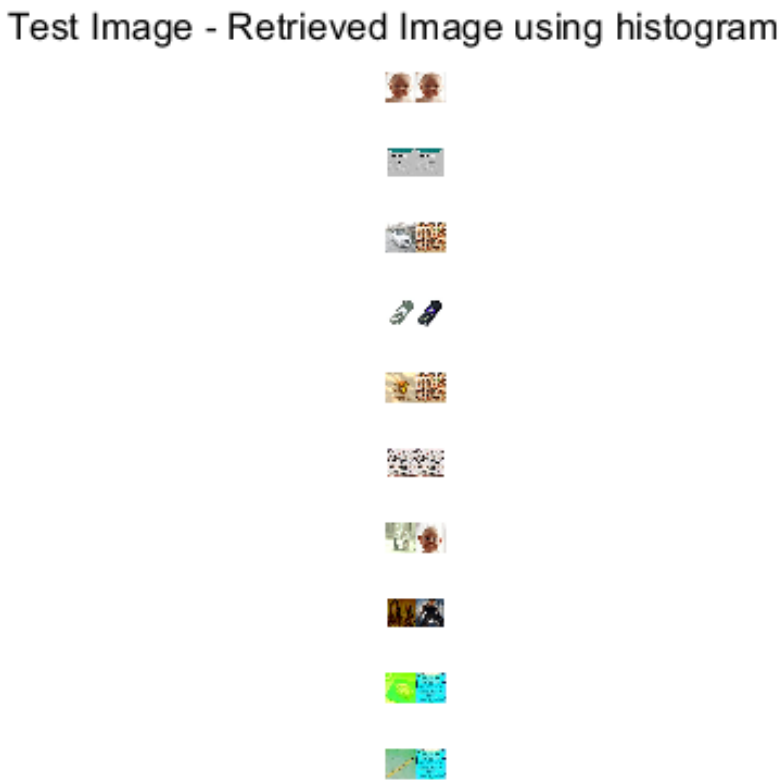
Έπειτα, πάμε για κάθε εικόνα test, υπολογίζουμε το ιστόγραμμα της και υπολογίζουμε το μέσο τετραγωνικό σφάλμα της από κάθε ιστόγραμμα της βάσης. Κρατάμε το ιστόγραμμα με την μικρότερη απόσταση και αποθηκεύουμε το index του στη βάση. Δημιουργούμε έτσι τη μεταβλητή matchedImagePairs(x,y) όπου x οι εικόνες test, και y οι αντίστοιχες εικόνες με τα εγγύτερα ιστογράμματα.

Κώδικας:

```
%% Image retrieval from database
close all; clear;
unzip("DataBase.zip");
%% Representing Images as Histograms

imgDatabase = imageDatastore("DataBase");
testImages = imageDatastore("test");
histDB = zeros(length(imgDatabase.Files),256);

% Saving image database as image histograms
for i = 1:length(imgDatabase.Files)
    currImg = readimage(imgDatabase,i);

    %Convert to grayscale
    currImg = rgb2gray(currImg);

    %Calculate histogram
    [currCounts,currBins] = imhist(currImg);
    histDB(i,:) = transpose(currCounts);

end

%For each image in the test file, calculate its histogram and
%check if it matches any histograms in the database.
matchedImageIndexPairs = zeros(length(testImages.Files),2);
successes = zeros(1,10);
for testIndex = 1:length(testImages.Files)
    closestDistance = Inf;
    currImg = readimage(testImages,testIndex);

    %Convert to grayscale
    currImg = rgb2gray(currImg);


    %Calculate histogram
    [currCounts,currBins] = imhist(currImg);
    %For each histogram in the database, find the closest ones index

    for dBIndex = 1:size(histDB,1)
        %Calculate difference between histogram counts using immse with
        %MSE as the metric
        currentDistance = immse(histDB(dBIndex,:),transpose(currCounts));
        if (currentDistance < closestDistance)
            closestDistance = currentDistance;
            matchedImageIndexPairs(testIndex,:) = [testIndex dBIndex];
        end
    end

    %Check if retrieval was successful

    [testImg,~] = readimage(testImages,testIndex);
    [retrievedImg,~] = readimage(imgDatabase,matchedImageIndexPairs(testIndex,2));
    successes(testIndex) = isequal(testImg,retrievedImg);
```

```
end

%Plotting test images and retrieved images
succPercentage = sum(successes) / length(successes);
f3 = figure;
for i = 1:10
    [testImg,~] = readimage(testImages,i);
    [retrievedImg,~] = readimage(imgDatabase,matchedImageIndexPairs(i,2));
    subplot(10,1,i), imshowpair(testImg,retrievedImg,'montage')
end
sgtitle("Test Image - Retrieved Image using histogram");
```

Το αποτέλεσμα είναι το εξής:



Παρατηρούμε ότι έχουμε μόνο 4 επιτυχείς ανακτήσεις εικόνας (1,2,4,6)

Στη συνέχεια εφαρμόζουμε την ίδια λογική, αλλά αυτή τη φορά αναπαριστούμε τις εικόνες με τον μετασχηματισμο DCT τους, με διαφορετικό πλήθος συντελεστών (10%, 50%, 100%).

Κώδικας:

```
%% Representing Images using DCT

dctDB_r_0_1 = zeros(length(imgDatabase.Files),100,100); % 10 % of coefficients kept
dctDB_r_0_5 = zeros(length(imgDatabase.Files),100,100); % 50 % of coefficients kept
dctDB_r_0_0 = zeros(length(imgDatabase.Files),100,100); % all coefficients kept

windowSize = [32 32];
r_values = [0.1 0.5 1];

% Saving image database as dct transforms
for dBIndex = 1:length(imgDatabase.Files)

    %Current Image
    currImg = readimage(imgDatabase,dBIndex);
    currImg = rgb2gray(currImg);

    %Implementing 2D-DCT with 1D-DCT
    for r = r_values
        Q = dct(double(currImg),[],1);
        R = dct(Q,[],2);
        X = R(:);

        %Keeping only values above the threshhold
        current_threshold = sum(abs(X)) / (size(currImg,1) * size(currImg,2)) * ((1 -
r)/0.5);
        R(abs(R) < current_threshold) = 0;

        if (r == 0.1)
            dctDB_r_0_1(dBIndex,:,:) = R;

        elseif (r == 0.5)
            dctDB_r_0_5(dBIndex,:,:) = R;

        else
            dctDB_r_0_0(dBIndex,:,:) = R;
        end
    end

end


% For each test image, find its DCT transform with different coefficient
% percentage and try to find its match in the database
matchedImageIndexPairs_r_0_1 = zeros(length(testImages.Files),2);
matchedImageIndexPairs_r_0_5 = zeros(length(testImages.Files),2);
matchedImageIndexPairs_r_0_0 = zeros(length(testImages.Files),2);

successesDCT_r_0_1 = zeros(1,10);
successesDCT_r_0_5 = zeros(1,10);
successesDCT_r_0_0 = zeros(1,10);
```

```
testImageDCT_r_0_1 = zeros(100,100);
testImageDCT_r_0_5 = zeros(100,100);
testImageDCT_r_0_0 = zeros(100,100);

for testIndex = 1:length(testImages.Files)
    %For each image in the test database
    currImg = readimage(testImages,testIndex);

    %Convert to grayscale
    currImg = rgb2gray(currImg);

    %Find DCT for different r_values
    for r = r_values
        Q = dct(double(currImg),[],1);
        R = dct(Q,[],2);
        X = R(:);

        %Keeping only values above the threshhold
        current_threshold = sum(abs(X)) / (size(currImg,1) * size(currImg,2)) * ((1 -
r)/0.5);
        R(abs(R) < current_threshold) = 0;
        if (r == 0.1)
           testImageDCT_r_0_1 = R;

        elseif (r == 0.5)
           testImageDCT_r_0_5 = R;

        else
           testImageDCT_r_0_0 = R;
        end
    end

    %After we have calculated the DCT for different r, parse the img database
    %and find the closest match for each test image
    closestDistance_0_1 = Inf;
    closestDistance_0_5 = Inf;
    closestDistance_0_0 = Inf;
    for dBIndex = 1:length(imgDatabase.Files)

        %First for r = 0.1
        currentDistance =
immse(testImageDCT_r_0_1,squeeze(dctDB_r_0_1(dBIndex,:,:)));
        if (currentDistance < closestDistance_0_1)
            closestDistance_0_1 = currentDistance;
            matchedImageIndexPairs_r_0_1(testIndex,:) = [testIndex dBIndex];
        end

        %Then for r = 0.5
        currentDistance =
immse(testImageDCT_r_0_5,squeeze(dctDB_r_0_5(dBIndex,:,:)));
        if (currentDistance < closestDistance_0_5)
            closestDistance_0_5 = currentDistance;
            matchedImageIndexPairs_r_0_5(testIndex,:) = [testIndex dBIndex];
        end
```

```
        %Finally r = 0 (100% of coeffs kept
        currentDistance =
immse(testImageDCT_r_0_0,squeeze(dctDB_r_0_0(dBIndex,:,:)));
        if (currentDistance < closestDistance_0_0)
            closestDistance_0_0 = currentDistance;
            matchedImageIndexPairs_r_0_0(testIndex,:) = [testIndex dBIndex];
        end
    end
end


%Find success percentage for each r
for i = 1:10
    [testImg,~] = readimage(testImages,i);


    %r = 0.1
    [retrievedImg1,~] = readimage(imgDatabase,matchedImageIndexPairs_r_0_1(i,2));
    successesDCT_r_0_1(i) = isequal(testImg,retrievedImg1);


    %r = 0.5
    [retrievedImg2,~] = readimage(imgDatabase,matchedImageIndexPairs_r_0_5(i,2));
    successesDCT_r_0_5(i) = isequal(testImg,retrievedImg2);

    %r = 0 (100% coeffs)
    [retrievedImg3,~] = readimage(imgDatabase,matchedImageIndexPairs_r_0_0(i,2));
    successesDCT_r_0_0(i) = isequal(testImg,retrievedImg3);


end

f4 = figure;
for i = 1:10
    [testImg,~] = readimage(testImages,i);
    [retrievedImg,~] = readimage(imgDatabase,matchedImageIndexPairs_r_0_1(i,2));
    subplot(10,1,i), imshowpair(testImg,retrievedImg,'montage')
end
sgtitle("Test Image - Retrieved Image using DCT - 10% of coeffs");

f5 = figure;
for i = 1:10
    [testImg,~] = readimage(testImages,i);
    [retrievedImg,~] = readimage(imgDatabase,matchedImageIndexPairs_r_0_5(i,2));
    subplot(10,1,i), imshowpair(testImg,retrievedImg,'montage')
end
sgtitle("Test Image - Retrieved Image using DCT - 50% of coeffs");

f6 = figure;
for i = 1:10
    [testImg,~] = readimage(testImages,i);
    [retrievedImg,~] = readimage(imgDatabase,matchedImageIndexPairs_r_0_0(i,2));
    subplot(10,1,i), imshowpair(testImg,retrievedImg,'montage')
end
sgtitle("Test Image - Retrieved Image using DCT - 100% of coeffs");
```

Για τα τρία διαφορετικά ποσοστά συντελεστών έχουμε:

Παρατηρούμε ότι και για τις τρείς περιπτώσεις συντελεστών έχουμε 6/10 επιτυχίες (1,2,4,5,6,7). Άρα η σύγκριση των εικόνων με βάση το συχνοτικό περιεχόμενο του DCT τους αποδεικνύεται καλύτερος τρόπος για την ανάκτηση μιας εικόνας από μια βάση δεδομένων.