Rosie Manoli

Course: Digital Image Processing and Applications

Assignment #1: Introduction, Transformations, Optimization, and Segmentation

Introduction: For code design I used the online Octave environment and the book “Ψηφιακή επεξεργασία & ανάλυση εικόνας” by Papamarkos Nikolaos H.. In the repository you will find all my scripts, providing the code for solving the following questions.

Exercise 1 - Quantization: Consider the grayscale image "barbara.bmp".

<u>Questions</u>:
A. Design and implement (without using built-in quantization functions) a uniform mid-rise scalar quantizer with 8, 12, 16, and 20 levels.
B. Plot the transformation function of each quantizer in a single figure.
C. Apply each quantizer to the given image and display the results in a single figure. What do you observe?
D. Calculate the mean squared quantization error for each case.

<u>Answers</u>:
A. The design and implementation of the code for the uniform mid-rise scalar quantizer are provided in the repository. To design the mid-rise function, I wrote its mathematical formula from the textbook into a script, which was then applied in the code to display the images in the command window.
B - C. The result after applying the quantizers to the original image is shown in the following image for each quantization level, along with the corresponding mean squared quantization error. From the images, we observe that as the quantization levels increase, the image becomes smoother but loses detail, as information is removed from the image.

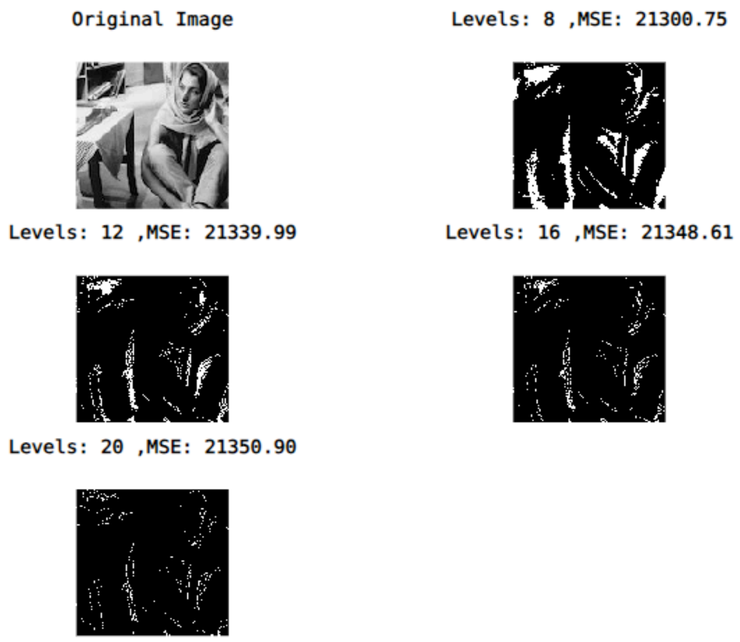

Original Image
Levels: 8 ,MSE: 21300.75
Levels: 12 ,MSE: 21339.99
Levels: 16 ,MSE: 21348.61
Levels: 20 ,MSE: 21350.90

Image 1: The original image and the images after applying each quantizer.

D. The mean squared error is calculated using the function implemented in a script, which was then called—similar to the mid-rise function—within the code for image representation in the command window. The code will be provided in the repository.

Exercise 2 -Fourier Transformation : Consider the grayscale image "barbara.bmp".

<u>Questions</u>:
A. Compute the 2D Fourier Transform (DFT) of the image.
B. Compute and display the amplitude and phase spectra of the transform.
C. Consider only 20% and 40% of the coefficients (low frequencies) of the DFT along each axis and reconstruct the original image using the inverse Fourier transform. Display the reconstructed images in a single figure. What do you observe?
D. Compute the mean squared reconstruction error in each case.

<u>Answers</u>:
A - B. For the calculation of the 2D Fourier transform (DFT) of the image, the function fft2() was used, as suggested in the textbook. To calculate the amplitude and phase spectra, the functions abs() and angle() were used, respectively. By plotting the amplitude and phase spectra, we observe that the amplitude spectrum shows the content of the image in the frequency domain, while the phase spectrum provides information about the position of the main information and the variation of the frequencies. This process allows us to analyze how the image's frequency components are distributed and the role of phase in capturing the spatial structure of the image.

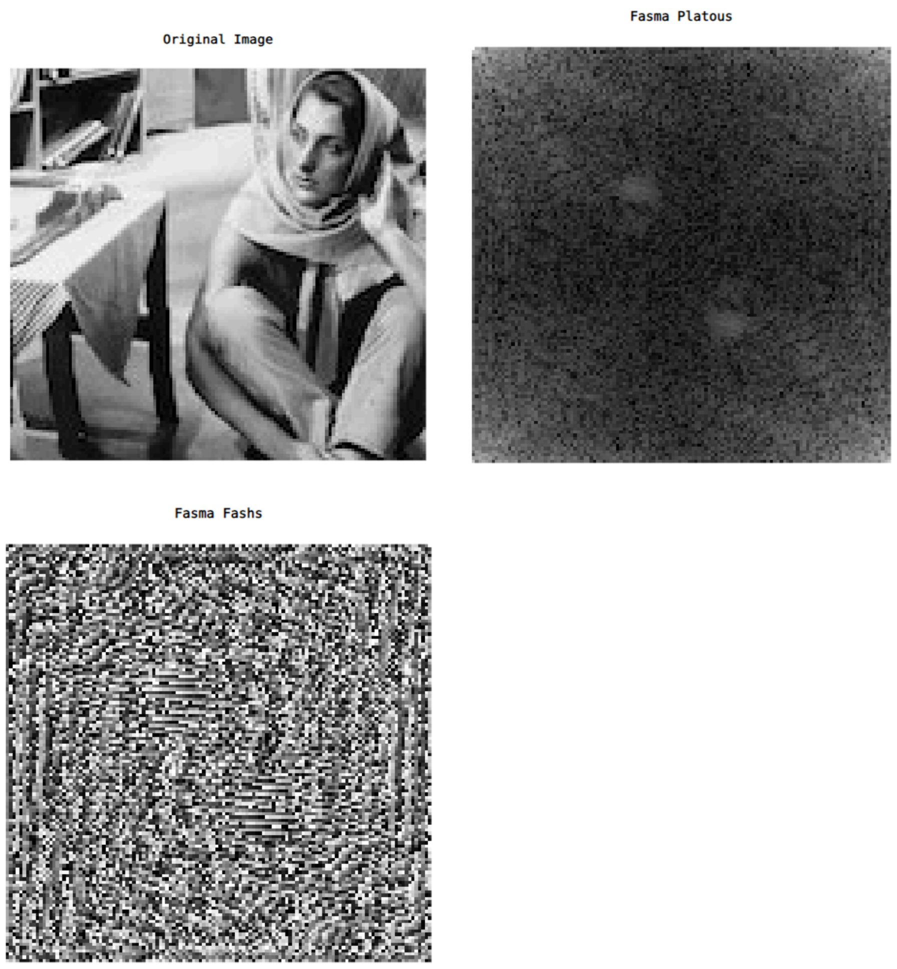


Image 2: The original image, the amplitude spectrum, and the phase spectrum.

C - D. For the calculation of the inverse Fourier transform for the original image, the function ifft2() was used, as suggested in the textbook. The code written for the reconstruction of the images in the two cases and for calculating the mean squared reconstruction error in each case is provided in the repository.

When plotting the reconstructed images using only a portion of the DFT coefficients (20% and 40% along each axis), we observe that with the lower frequencies, details are lost. This is why the image with 20% of the coefficients is quite blurry, while the image with 40%, although still blurry, is closer to the original. Finally, from the squared error, we can see how well the reconstruction was done, which is why the error for 40% is smaller than the error for 20%.

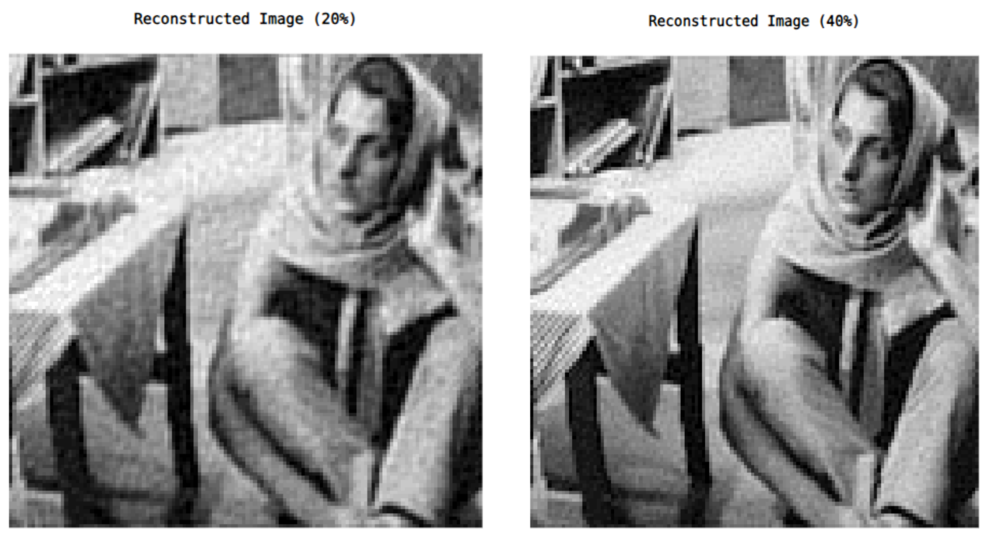
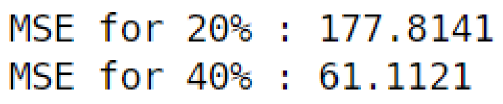


Image 3: The reconstructed images with only 20% and 40% of the coefficients, respectively.



```
MSE for 20% : 177.8141
MSE for 40% : 61.1121
```

Image 4: The mean squared reconstruction error for each case.

Exercise 3 - Cosine Transform: Consider the grayscale image "barbara.bmp".

Questions:
A. Implement the forward and inverse 2D Discrete Cosine Transform (DCT - IDCT) of the image (without using built-in DCT - IDCT functions).
B. Compute and display the spectrum of the transform.

Answers:
A - B. For the computation of the Discrete Cosine Transform (DCT) and its inverse (IDCT) of the image, the function provided in the textbook was implemented in a script. This script was then applied in the code for displaying the images in the command window. The corresponding code will be included in the repository. By printing the transform spectrum, we can observe how the image information is distributed. **Note:** The execution of this code was slower compared to the other functions, taking longer to run.

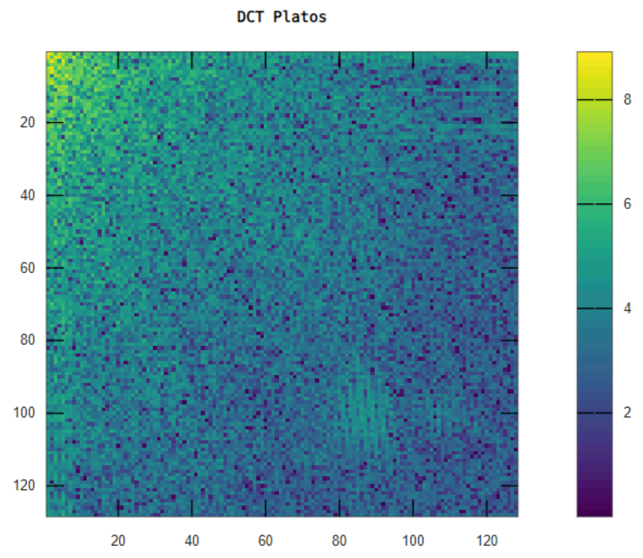


Image 5: The transform spectrum.

Exercise 4 - Filtering in the Spatial Domain: Consider the grayscale image "lenna.bmp". Add salt & pepper noise with a density of 0.05 (i.e., 5% coverage of the image surface).

Questions:
A. Display the image before and after adding the noise.
B. Apply mean filters of sizes 3×3, 5×5, and 7×7 separately to the noisy image using a convolution operator.
C. Display all filtered images in a single figure and compare them with the original image. What do you observe?

Answers:
A. To add salt & pepper noise to the original image with a density of 0.05, the function imnoise(I, 'salt & pepper', D) was used, where D represents the noise density, as recommended by the textbook. The code for displaying the images before and after adding noise will be provided in the repository. After printing the noisy image, we observe many scattered dots across the image surface, which is a characteristic of salt & pepper noise.

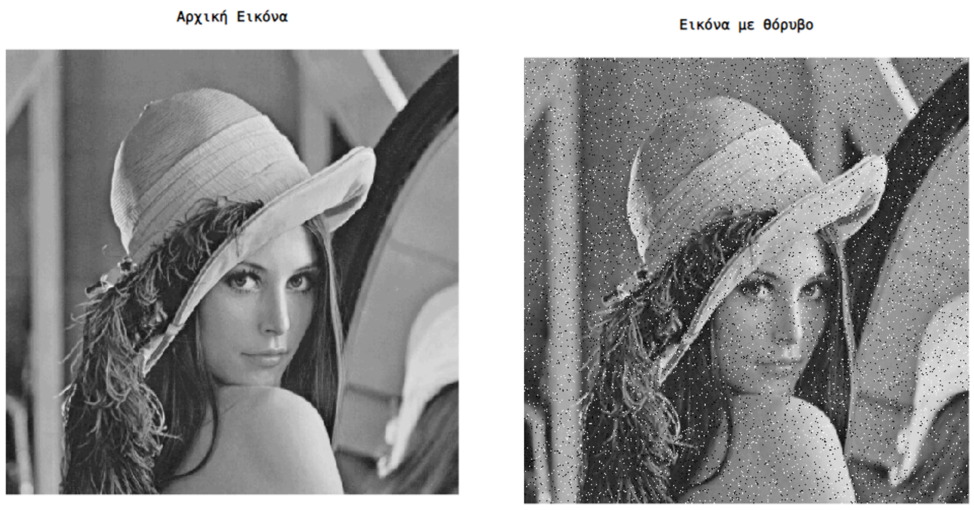


Image 6: The original image and the image with added salt & pepper noise.

B -C. For the design of the mean filters, I used functions that were provided in the lab of the course. Applying the mean filters with different sizes to the noisy image, we observe that the image becomes smoother, and the salt & pepper noise is removed. Specifically, for the 3x3 filter, we see that the noise is gone, and the image is quite close to the original. However, as the filter size increases, we notice that although the noise is removed, the image becomes blurry, and details are gradually lost (especially with the 7x7 filter). Therefore, we conclude that the correct filter size should be chosen when trying to remove noise, so that we don't lose details from the image.

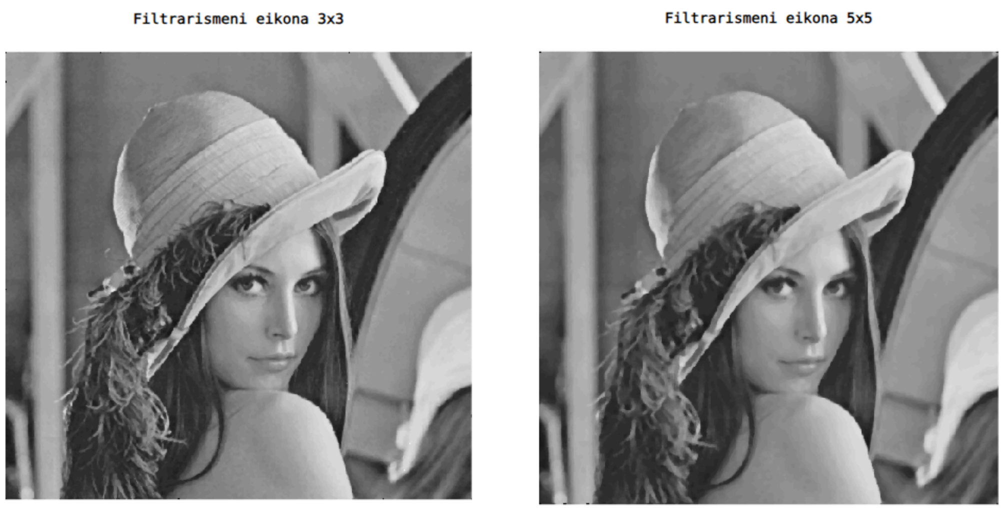


Image 7: The filtered image with 3x3 and 5x5 filter sizes.

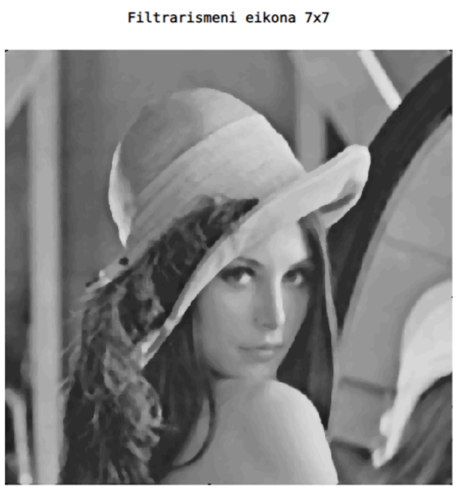


Image 8: The filtered image with a 7x7 filter size.

Exercise 5 - Frequency domain filtering: Consider the grayscale image "lenna.bmp". Add Gaussian noise to the image with a mean of zero and a variance of 0.01.

<u>Questions</u>:
A. Print the image before and after the addition of noise.
B. Consider successive low-pass Butterworth filters of 3rd, 5th, and 7th order in the frequency domain. Apply each filter separately to the noisy image (using the forward and inverse Fourier transform).
C. Print in a single figure the resulting filtered image for each case of the applied filter and compare with the original image. What do you observe? It is reminded that the imaginary part of the response resulting from the application of the inverse Fourier transform can be ignored.

<u>Answers</u>:
A. To add Gaussian noise to the original image, with a mean value of 0 and a variance of 0.01, the function imnoise(I, 'gaussian', m, s) was used, where m is the mean and s is the

variance, as suggested by the book. The code for representing the images before and after adding noise will be provided in the repository. After printing the image following the addition of noise, we observe uniform noise across the entire surface of the image. This is characteristic of Gaussian noise.

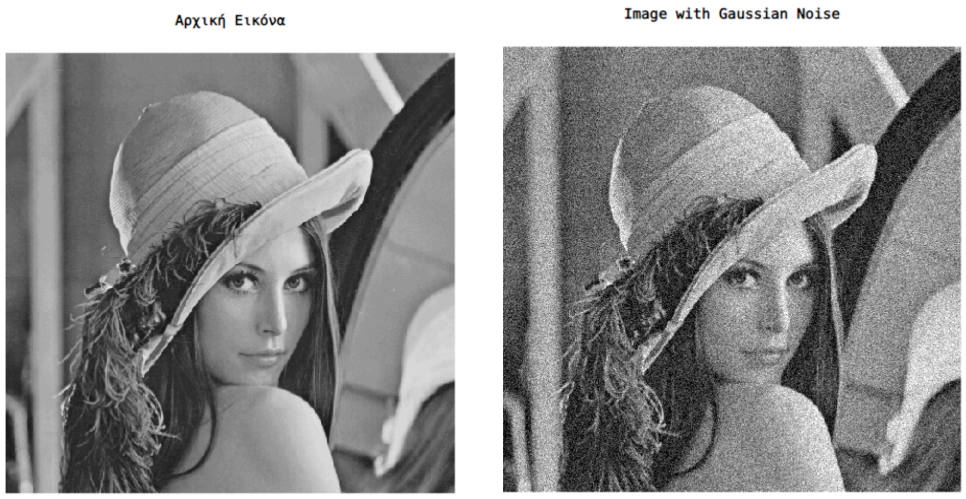


Image 9: The original image and the image with added Gaussian noise

B -C. For the design of the low-pass Butterworth filters of the 3rd, 5th, and 7th order, I used the functions provided in the lab of the course. By applying the Butterworth filters with various orders to the noisy image, we observe that the image is smoothed, and the Gaussian noise is removed. However, since the filter is low-pass, the high frequencies are cut off, which is why, although the images no longer have noise, they have lost details and are not as sharp. Specifically, for the 3rd order filter, we see that the noise has been significantly removed, and the image is quite close to the original, though still somewhat blurry. However, by increasing the order of the filters, we observe that while the noise is removed, the image becomes blurrier, and many details are gradually lost as more high frequencies are cut off (especially in the 7th-order filter). Particularly in areas with many details, such as the feathers on the hat, the details are lost, and the points are quite blurry in the filtered images.

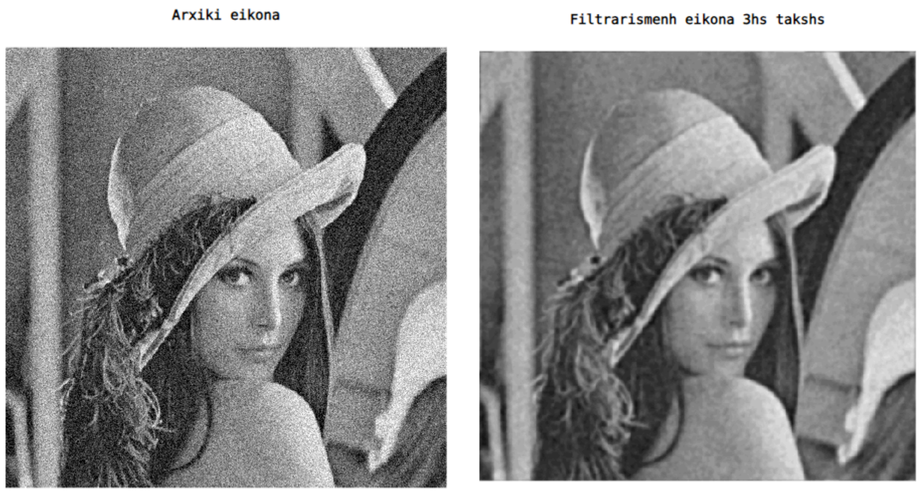


Image 10: The filtered image with the 3rd-order Butterworth filter

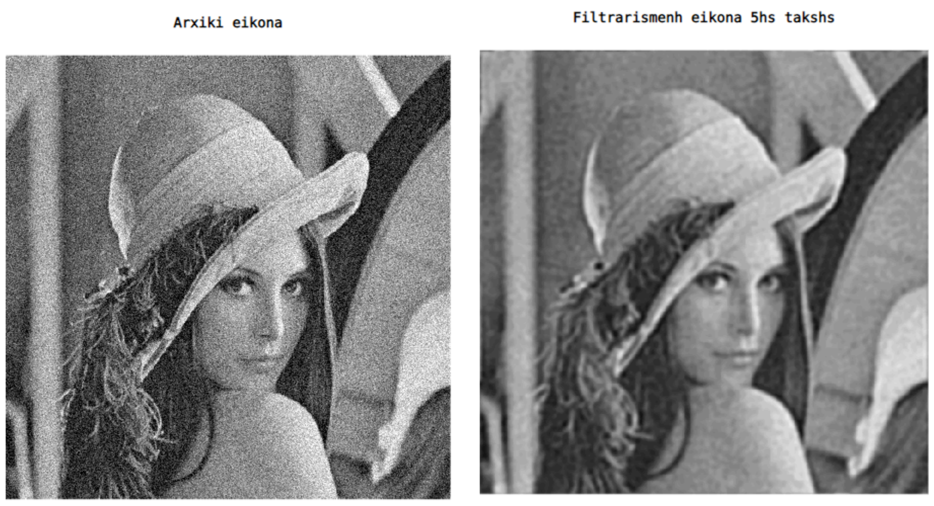


Image 11: The filtered image with the 5th-order Butterworth filter

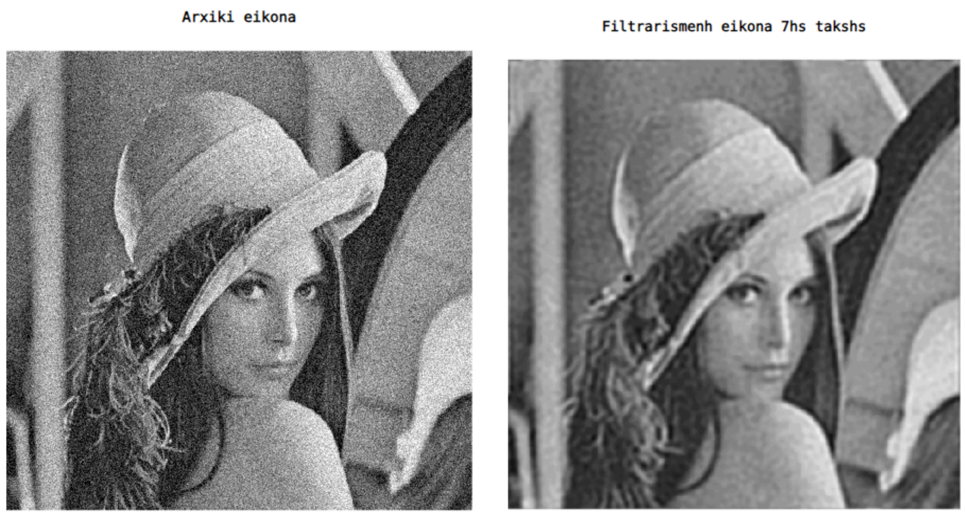


Image 12: The filtered image with the 7th-order Butterworth filter

Exercise 6 - Consider the color image "butterfly.jpg". Consider a segmentation method based on the clustering algorithm 'K-means'. The goal of the K-means algorithm is to identify K (K< N) groups-classes for a set of N data samples. Each sample is assigned to only one class, that is, the class for which the sample has the minimum distance from the calculated center.

Specifically, for the case of color images, each sample n corresponds to a pixel (xn, yn) of the examined image fn(xn, yn) = [Rn, Gn, Bn], where (xn, yn) are the pixel coordinates in the image and [Rn, Gn, Bn] is the vector of the color components. The goal of K-means is to compute the K class centers Ck[Rk, Gk, Bk] through an iterative optimization process, where the parameter K is predefined. Implement the following version of the K-means algorithm (do not use pre-built library functions):

- Transform the color image into the RGB color space, meaning each pixel fn(xn, yn) is represented by a 3D vector [Rn, Gn, Bn] of the basic red, green, and blue colors.
- Algorithm initialization: randomly select pixels from the image as initial class centers Ck[Rk, Gk, Bk].
- Consider the Euclidean distance dk = || [Rn, Gn, Bn] - Ck[Rk, Gk, Bk] || as the distance-similarity function of a point to each class.
- The K-means algorithm should run until it converges (i.e., there is no change in the classification of the pixels between two consecutive steps) or until T = 50 steps of the algorithm are reached.

Questions:
A. Implement the above version of the 'K-means' algorithm.
B. Apply the segmentation algorithm to the given color image and print the result (segmentation mask) as a grayscale image (each object should correspond to a different shade) for K = 5, 10, and 15. Comment on the segmentation results.

Απαντήσεις:
A. For the implementation of the K-means algorithm, I used both the provided sources and other sources from the internet. The code for the design will be provided in the repository and the sources will be referenced in the appendix. In the code, I performed the color transformation of the image into the RGB color space, then defined the clusters and randomly initialized the centers. Afterward, I implemented the 50 steps as described in the problem statement for the algorithm to run. For the Euclidean distance, I used the provided sources. Then I find the closest center and check that it is not the same as the previous one. Finally, I assign each pixel to the clustered center, convert the image to grayscale, and print the images.

B. Printing the images after applying the algorithm, we observe that as K gets smaller, the segmentation is more coarse. Specifically, we can see that the sky, wings, and the flower are where segmentation is more prominent. As K increases, the details become more precise, and more shades are used to better distinguish them.

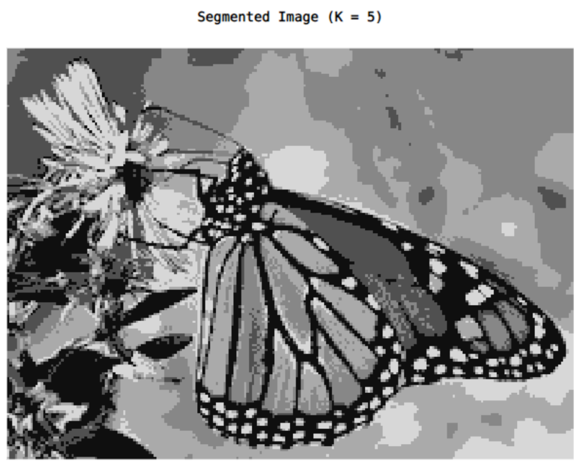


Image 13: Application of the algorithm to the image with K = 5

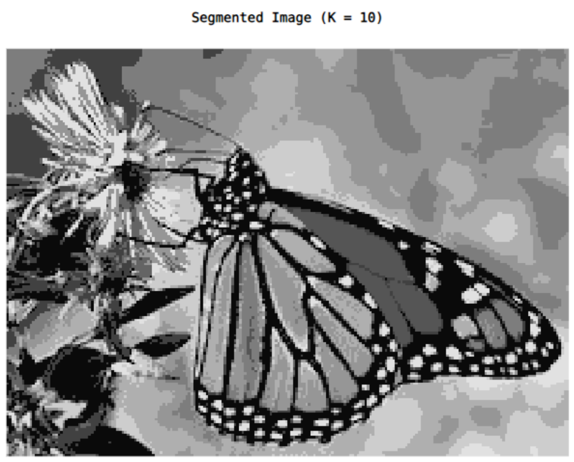


Image 14: Application of the algorithm to the image with K = 10

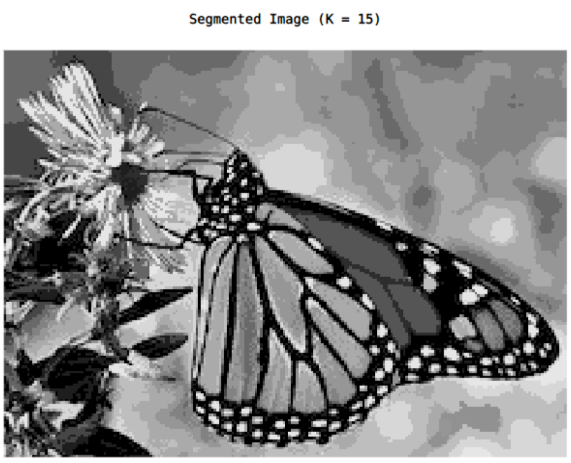


Image 15: Application of the algorithm to the image with K = 15

-For the design of the K-means algorithm, I used the following sources::

1. https://towardsdatascience.com/create-your-own-k-means-clustering-algorithm-in-python-d7d4c9077670
2. https://www.mathworks.com/help/stats/kmeans.html
3. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9072123
4. https://www.slideshare.net/slideshow/k-meanclustering-algorithm/51056901