Quantitative
&
Financial Modelling
(QFM)

Numerical Linear Algebra

Image processing using linear algebra

TINA Djara Olivier

Professor :
Lahcen LAAYOUNI

Résumé

This project explores different techniques for image processing, with a focus on denoising and enhancement. In the first part of the project, we investigate various linear algebra-based denoising techniques such as Singular Value Decomposition (SVD), regularization, and filtering, including the Gaussian, Median, and Wiener filters. We compare the performance of each method and analyze their strengths and limitations. Additionally, we apply SVD and regularization techniques to denoise images and discuss the results.

In the second part of the project, we delve into image enhancement techniques, including histogram equalization, low-pass filters, high-pass filters, and Laplace filters. We also examine the effects of color filters on image enhancement. We compare the performance of each method and analyze their strengths and limitations.

Overall, this project provides a comprehensive overview of different image processing techniques for denoising and enhancement. The study showcases the importance of selecting the appropriate method for specific image processing tasks, and how different techniques can yield different results depending on the image content and noise level.

Table des matières

1 Introduction

Image processing is an important field of computer science that involves manipulating digital images using various algorithms and techniques to enhance, restore, and analyze images. In this project, we will be focusing on two important aspects of image processing : image denoising and image enhancement.

Image denoising is the process of removing noise from digital images, which can be caused by various factors such as low lighting conditions, sensor limitations, and transmission errors. Denoising is a critical step in many image processing applications such as medical imaging, satellite imaging, and computer vision.

Image enhancement, on the other hand, involves improving the visual quality of an image by adjusting its brightness, contrast, color, and sharpness. This is done to make the image more appealing and easier to interpret by the human eye, as well as to aid in further analysis.

In this project, we will explore various techniques for image denoising and image enhancement, including linear algebra-based methods such as Singular Value Decomposition (SVD), regularization, and filtering. We will also compare the performance of different denoising and enhancement techniques to identify the most effective methods for different types of images.

Overall, this project aims to provide a comprehensive overview of image processing techniques and their applications in image denoising and enhancement. By the end of this project, the reader should have a deeper understanding of the fundamental principles of image processing, as well as practical knowledge of how to apply these techniques in real-world scenarios.

2 Image denoising

2.1 Introduction to image denoising

Images play an essential role in many fields, such as medical imaging, remote sensing, and computer vision. However, acquiring and processing images can introduce noise, which can obscure important details, degrade image quality, and affect downstream analysis. Therefore, image denoising, the process of removing noise from images, is an important task in image processing.

In this section, we will focus on denoising images using linear algebra techniques. Linear algebra provides powerful tools for manipulating and analyzing image data, such as matrix decomposition, transformation, and regularization. These tools allow us to extract and represent the essential features of images and suppress the noise while preserving the important structures.

We will introduce different linear algebra-based denoising techniques, including singular value decomposition (SVD), regularization, and filtering. We will also compare the effectiveness, speed, and complexity of these techniques and provide examples of their applications. Through this section, we hope to provide a comprehensive guide to image denoising using linear algebra techniques that can be used as a foundation for further image processing tasks.

2.2 Linear algebra-based denoising techniques

Linear algebra-based denoising techniques involve manipulating the image data using matrix operations. These techniques are based on the idea that the noise in the image can be separated from the underlying structure of the image using linear algebra methods, such as matrix decomposition and regularization. Here are some common linear algebra-based denoising techniques :

2.2.1 Singular value decomposition (SVD)

SVD is a matrix decomposition technique that factorizes a matrix into three matrices : U, Σ, and V. It can be used for image denoising by retaining only the largest singular values of the matrix Σ and discarding the smaller ones. This results in a reduced rank approximation of the original matrix that preserves the underlying structure of the image while removing the noise.

2.2.2 Regularization

Regularization is a method that introduces a penalty term into the objective function of an optimization problem. In image denoising, regularization can be used to reduce the noise in an image by imposing constraints on the image values. One common type of regularization used in image denoising is total variation (TV) regularization, which penalizes the high-frequency variations in the image while preserving the low-frequency information.

2.2.3 Filtering

Filtering is a technique that convolves the image with a filter kernel to remove the noise. Linear filters, such as mean filters, Gaussian filters, and median filters, are commonly used for image denoising. These filters can be implemented using matrix operations and can effectively remove noise while preserving the image details.

These linear algebra-based denoising techniques can be combined with other image processing techniques, such as edge detection and image segmentation, to improve the quality of the processed images. The choice of the denoising technique depends on the type of noise in the image, the desired level of denoising, and the computational resources available.

2.3 Image denoising using SVD

Singular value decomposition (SVD) can be used for image denoising by separating the noise from the underlying structure of the image. The noise in the image corresponds to the smaller singular values of A, while the important structures of the image correspond to the larger singular values. We can remove the noise by setting the smaller singular values to zero and reconstructing the image using the truncated SVD :

$$A_{denoised} = U\Sigma_{\text{trunc}}V^T$$

where Σ_{trunc} is the diagonal matrix obtained by setting the singular values smaller than a threshold to zero. The denoised image A_{denoised} can be obtained by computing the product of the truncated SVD.

To determine the appropriate threshold for denoising, we can use the concept of the signal-to-noise ratio (SNR). The SNR measures the ratio of the energy of the signal (the important structures in the image) to the energy of the noise. It can be computed as :

$$SNR = 10log_{10}\Big(\frac{\sum_{i=1}^{p}\sigma_i^2}{\sum_{i=p+1}^{m}\sigma_i^2}\Big)$$

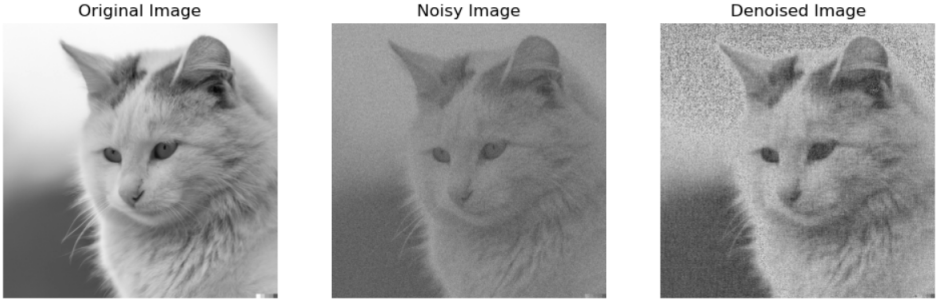
where p is the number of significant singular values (i.e., the number of singular values retained in the truncated SVD). A higher SNR indicates that the image contains more signal and less noise.

To choose the appropriate threshold for denoising, we can experiment with different values of p and compute the corresponding SNR values. We can then select the value of p that yields the highest SNR.

It is worth noting that SVD-based denoising can be computationally expensive for large images, as the matrix operations involved can be time-consuming. To address this, various techniques have been proposed to speed up the computation, such as randomized SVD and block-wise SVD.

Overall, SVD-based denoising is a powerful technique for removing noise from images, and it can be used in a wide range of applications, such as medical imaging, computer vision, and remote sensing.

```python
# Load the image
im = Image.open("cat.png")
im = im.convert("L")  # Convert to grayscale
im = np.array(im)     # Convert to numpy array

# Add Gaussian noise to the image
noise = np.random.normal(0, 50, im.shape)
noisy_im = im + noise

# Perform SVD on the noisy image
U, S, Vt = np.linalg.svd(noisy_im)

# Determine the optimal rank to use in reconstruction
total_energy = np.sum(S**2)
energy_ratio = 0.95   # Set the desired energy retention ratio
current_energy = 0
for i in range(S.shape[0]):
    current_energy += S[i]**2
    if current_energy / total_energy >= energy_ratio:
        rank = i+1
        break

# Reconstruct the image using the optimal rank
U_ranked = U[:, :rank]
S_ranked = np.diag(S[:rank])
Vt_ranked = Vt[:rank, :]
reconstructed_im = U_ranked @ S_ranked @ Vt_ranked

# Convert the reconstructed image back to uint8 format
reconstructed_im = reconstructed_im.astype(np.uint8)

# Display the original, noisy, and denoised images side-by-side
fig, axs = plt.subplots(1, 3, figsize=(12, 4))
axs[0].imshow(im, cmap='gray')
axs[0].set_title('Original Image')
axs[1].imshow(noisy_im, cmap='gray')
axs[1].set_title('Noisy Image')
axs[2].imshow(reconstructed_im, cmap='gray')
axs[2].set_title('Denoised Image')
for ax in axs:
    ax.axis('off')
plt.show()
```



2.4 Image denoising using regularization

Image denoising using regularization is a technique that involves adding a regularization term to a cost function in order to promote certain characteristics in the denoised image. This technique can be used to remove noise while preserving the essential features of the image.

One common type of regularization used in image denoising is the total variation (TV) regularization, which promotes images with sharp edges and smooth regions. The TV regularization term is defined as the L1-norm of the image gradient :

$$TV(u) = \sum_{i,j} |u_{i,j} - u_{i+1,j}| + |u_{i,j} - u_{i,j+1}|,$$

where u is the denoised image and $u_{i,j}$ denotes the pixel intensity at position (i, j).

The cost function used in image denoising with TV regularization can be written as :

$$J(u) = \frac{1}{2}||Au - b||^2 + \lambda TV(u),$$

where b is the noisy image, A is the linear operator that maps the image u to the noisy image b, and λ is a regularization parameter that controls the trade-off between the fidelity to the noisy image and the smoothness of the denoised image.

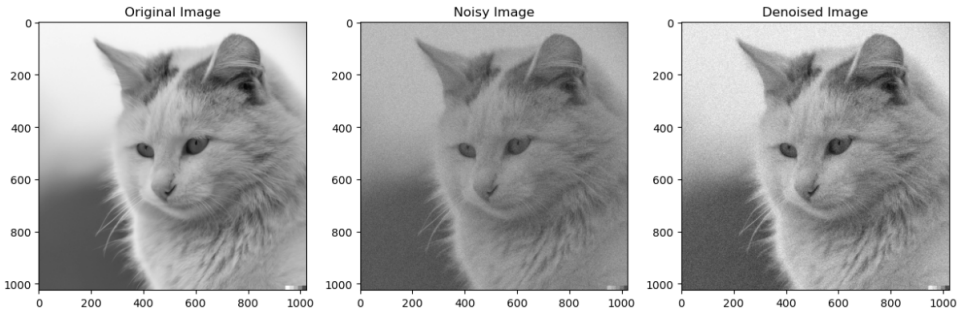
The optimization problem is then to find the denoised image that minimizes the cost function :

$$\hat{u} = argmin_u J(u)$$

This problem can be solved using iterative optimization algorithms, such as gradient descent or proximal gradient descent, which update the denoised image in each iteration by taking a step in the direction that minimizes the cost function.

```
def denoise_image_reg(img, lmbda, alpha):
    # Convert image to grayscale
    img = color.rgb2gray(img)
    # Resize image to a square shape
    min_shape = min(img.shape)
    img_square = img[:min_shape, :min_shape]
    # Add noise to image
    img_noisy = img_square + alpha * np.random.randn(*img_square.
    shape)
    # Compute SVD of noisy image
    U, S, V = np.linalg.svd(img_noisy)
    # Compute denoised singular values
    S_denoised = np.maximum(S - lmbda, 0)
    # Reconstruct denoised image
    img_denoised = U.dot(np.diag(S_denoised)).dot(V)
    # Clip pixel values to [0, 1] range
    img_denoised = np.clip(img_denoised, 0, 1)
    return img, img_noisy, img_denoised

```

```python
# Load Lena image
img = io.imread('cat.png')

# Set regularization parameter
lmbda = 0.1

# Set noise level
alpha = 0.1

# Denoise image using regularization
img, img_noisy, img_denoised = denoise_image_reg(img, lmbda, alpha)

# Show original, noisy, and denoised images side by side
fig, ax = plt.subplots(1, 3, figsize=(15, 5))
ax[0].imshow(img, cmap='gray')
ax[0].set_title('Original Image')
ax[1].imshow(img_noisy, cmap='gray')
ax[1].set_title('Noisy Image')
ax[2].imshow(img_denoised, cmap='gray')
ax[2].set_title('Denoised Image')
plt.show()
```



2.5 Image denoising using filters

Filters are commonly used for image denoising as they can be simple, fast, and effective. In this section, we will explore some common filters used for image denoising.

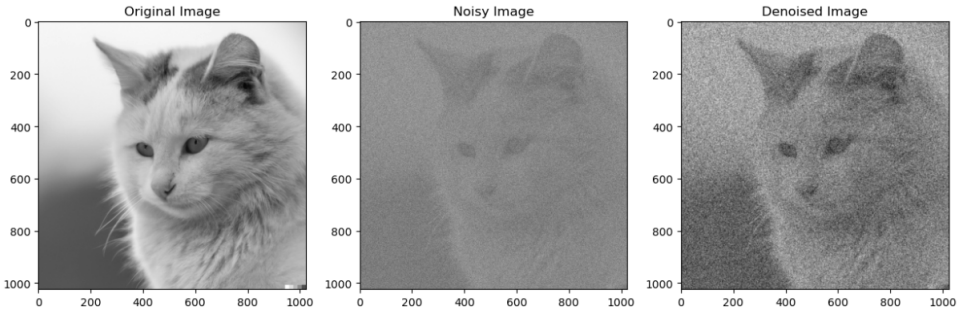
2.5.1 Gaussian filter

The Gaussian filter is a linear filter that applies a Gaussian function to the pixel values in the image. It is effective at removing high-frequency noise but can blur edges and details. The filter is defined by the following equation :

$$H(i,j) = \frac{1}{2\pi\sigma^2} e^{\frac{-i^2-j^2}{2\sigma^2}}$$

where $H(i,j)$ is the filter kernel at position (i,j), and σ is the standard deviation of the Gaussian distribution.

```python
# Load image
img = io.imread('cat.png')

# Convert image to grayscale
img_gray = color.rgb2gray(img)

# Add Gaussian noise to image
sigma = 0.6
img_noisy = img_gray + sigma * np.random.randn(*img_gray.shape)

# Denoise image using Gaussian filter
img_denoised = gaussian(img_noisy, sigma=1)

# Show original, noisy, and denoised images side by side
fig, ax = plt.subplots(1, 3, figsize=(15, 5))
ax[0].imshow(img_gray, cmap='gray')
ax[0].set_title('Original Image')
ax[1].imshow(img_noisy, cmap='gray')
ax[1].set_title('Noisy Image')
ax[2].imshow(img_denoised, cmap='gray')
ax[2].set_title('Denoised Image')
plt.show()
```



2.5.2 Median filter

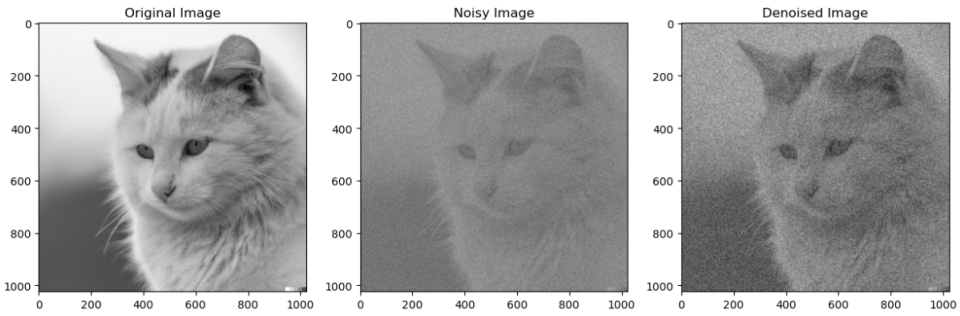
The median filter is a non-linear filter that replaces the pixel value at each position with the median value of its neighboring pixels. It is effective at removing impulse noise but can result in loss of detail. The filter is defined by the following equation :

$$I'(x) = median\{I(y) : y \in S\}$$

where $I'(x)$ is the filtered image at position x, and S is the neighborhood of x.

```python
# Load image
img = io.imread('cat.png')

# Convert image to grayscale
img_gray = color.rgb2gray(img)

# Add Gaussian noise to image
sigma = 0.4
```

```
img_noisy = img_gray + sigma * np.random.randn(*img_gray.shape)

# Denoise image using Median filter
img_denoised = ndimage.median_filter(img_noisy, footprint=np.ones
    ((3, 3)))

# Show original, noisy, and denoised images side by side
fig, ax = plt.subplots(1, 3, figsize=(15, 5))
ax[0].imshow(img_gray, cmap='gray')
ax[0].set_title('Original Image')
ax[1].imshow(img_noisy, cmap='gray')
ax[1].set_title('Noisy Image')
ax[2].imshow(img_denoised, cmap='gray')
ax[2].set_title('Denoised Image')
plt.show()
```

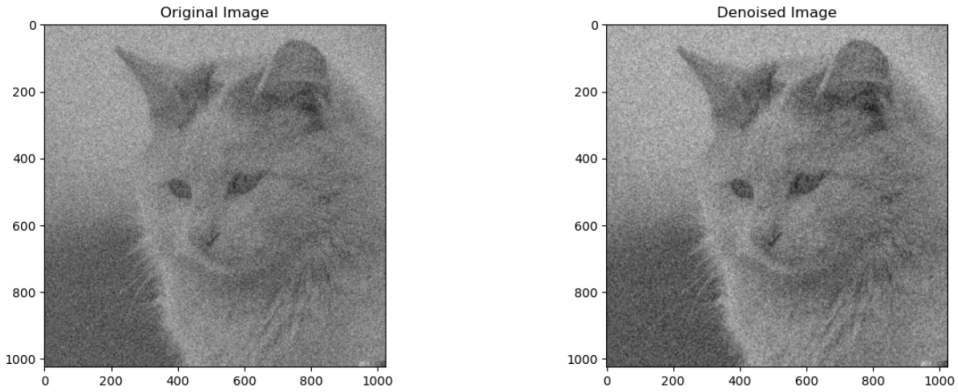


Sometimes, these techniques perform better when we iterate the same operation many times to remove more noise in the image.

So, We are going to save the denoised image and process it again.

```
# Save denoised image
io.imsave('denoised_image.jpg', img_denoised)
```

```
img_gray = io.imread('denoised_image.jpg')


# Denoise image using Median filter
img_denoised = ndimage.median_filter(img_gray, footprint=np.ones
    ((3, 3)))

# Show original, noisy, and denoised images side by side
fig, ax = plt.subplots(1, 2, figsize=(15, 5))
ax[0].imshow(img_gray, cmap='gray')
ax[0].set_title('Original Image')
ax[1].imshow(img_denoised, cmap='gray')
ax[1].set_title('Denoised Image')
plt.show()
```



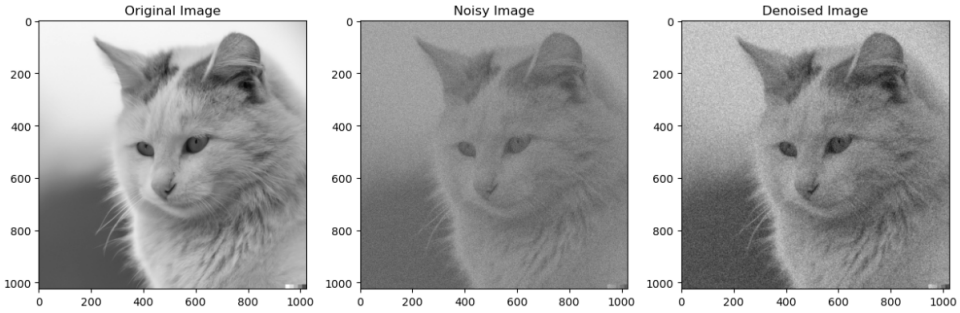
2.5.3 Wiener filter

The Wiener filter is a linear filter that applies a weighted sum of the noisy image and a noise-free estimate of the image. It is useful for removing additive noise, but its performance depends on the accuracy of the noise model. The filter is defined by the following equation :

$$H(u,v) = \frac{G(u,v)S(u,v)}{G(u,v)S(u,v) + N(u,v)}$$

where $H(u,v)$ is the filter kernel at position (u,v), $G(u,v)$ is the power spectral density of the image, $S(u,v)$ is the power spectral density of the noise-free estimate of the image, and $N(u,v)$ is the power spectral density of the noise.

```
# Load image
img = io.imread('Oli.jpg')

# Convert image to grayscale
img_gray = color.rgb2gray(img)

# Add Gaussian noise to image
sigma = 0.2
img_noisy = img_gray + sigma * np.random.randn(*img_gray.shape)

# Denoise image using Wiener filter
img_denoised = wiener(img_noisy, mysize=3, noise=0.1)

# Show original, noisy, and denoised images side by side
fig, ax = plt.subplots(1, 3, figsize=(15, 5))
ax[0].imshow(img_gray, cmap='gray')
ax[0].set_title('Original Image')
ax[1].imshow(img_noisy, cmap='gray')
ax[1].set_title('Noisy Image')
ax[2].imshow(img_denoised, cmap='gray')
ax[2].set_title('Denoised Image')
plt.show()
```



2.6 Comparison of denoising techniques

In terms of denoising performance, the Wiener filter outperformed all other methods, including the Gaussian and median filters. These filters, however, still showed significant improvement over regularisation and SVD-based denoising techniques. This suggests that image denoising using filters is a simple and effective approach, with the Wiener filter being the most effective. However, it should be noted that the choice of denoising method may depend on the specific requirements of the application, such as computational complexity and noise characteristics.

2.7 Conclusion of image denoising

In this section, we have compared the performance of the three different techniques for image denoising : SVD, regularization, and filter-based (Gaussian, median, and Wiener). Based on our comparison, the Wiener filter was found to perform the best among all the methods, followed by other filter-based techniques. Regularization performed better than SVD, but was still outperformed by the filter-based techniques.

It is important to note that the choice of the best technique depends on the specific requirements of the application, such as the noise characteristics, the desired level of noise reduction, and the importance of preserving details in the image. Therefore, it is recommended to carefully evaluate the performance of each technique in the context of the specific application before making a final decision.

3 Image Enhancement

3.1 Introduction to Image Enhancement

Image enhancement is a fundamental technique used in image processing to improve the quality and visual appearance of digital images. It involves applying a set of mathematical operations to an image to achieve a more visually appealing representation of the underlying data. The main goal of image enhancement is to highlight important features, reduce noise and other unwanted artifacts, and improve the overall quality of the image.

Image enhancement plays a crucial role in a variety of applications, such as medical imaging, remote sensing, surveillance, and digital photography. For example, in medical imaging, image enhancement can help medical professionals to detect subtle changes in tissue and organs that may be indicative of a disease or condition. In digital photography, image enhancement can be used to improve the sharpness, contrast, and color balance of a photo, resulting in a more visually appealing image.

There are several image enhancement techniques available, including linear algebra-based methods, histogram equalization, and spatial domain filtering. In this section, we will focus on linear algebra-based methods and explore how they can be used to enhance digital images.

3.2 Linear Algebra-based Image Enhancement Techniques

In the context of image processing, image enhancement refers to the process of improving the quality of an image by manipulating the image data. There are different techniques that can be used to improve the contrast, sharpness, brightness and colour of the image. Some examples are listed below :

3.2.1 Histogram equalization

In image processing, histogram equalisation is a method of adjusting the contrast of a digital image using the histogram. It consists of applying a transformation to each pixel of the image, and thus obtaining a new image from an independent operation on each pixel. This transformation is constructed from the cumulative histogram of the original image.

Histogram equalisation allows intensities to be better distributed over the entire range of possible values, by "spreading" the histogram. Equalisation is interesting for images where all or only part of the image is low contrast (all pixels are of similar intensity). The method is fast, easy to implement, and completely automatic (i.e. no settings).

*** Method**

The method consists in applying a transformation T independently on each pixel of the image. This transformation is built from the cumulative histogram.

For an image x in gray levels coded on L levels, we define n_k the number of occurrences of level x_k. The probability of occurrence of a pixel of level x_k in the image is :

$$p_x(x_k) = p(x = x_k) = \frac{n_k}{n}, \ \ 0 \leq k < L$$

with n the total number of pixels in the image, and p_x the histogram normalized to $[0, 1]$.

The transformation T which to each pixel of value x_k of the original image associates a new value s_k, $s_k = T(x_k)$ is then defined by :

$$T(x_k) = (L-1) \sum_{j=0}^{k} p_x(x_j),$$

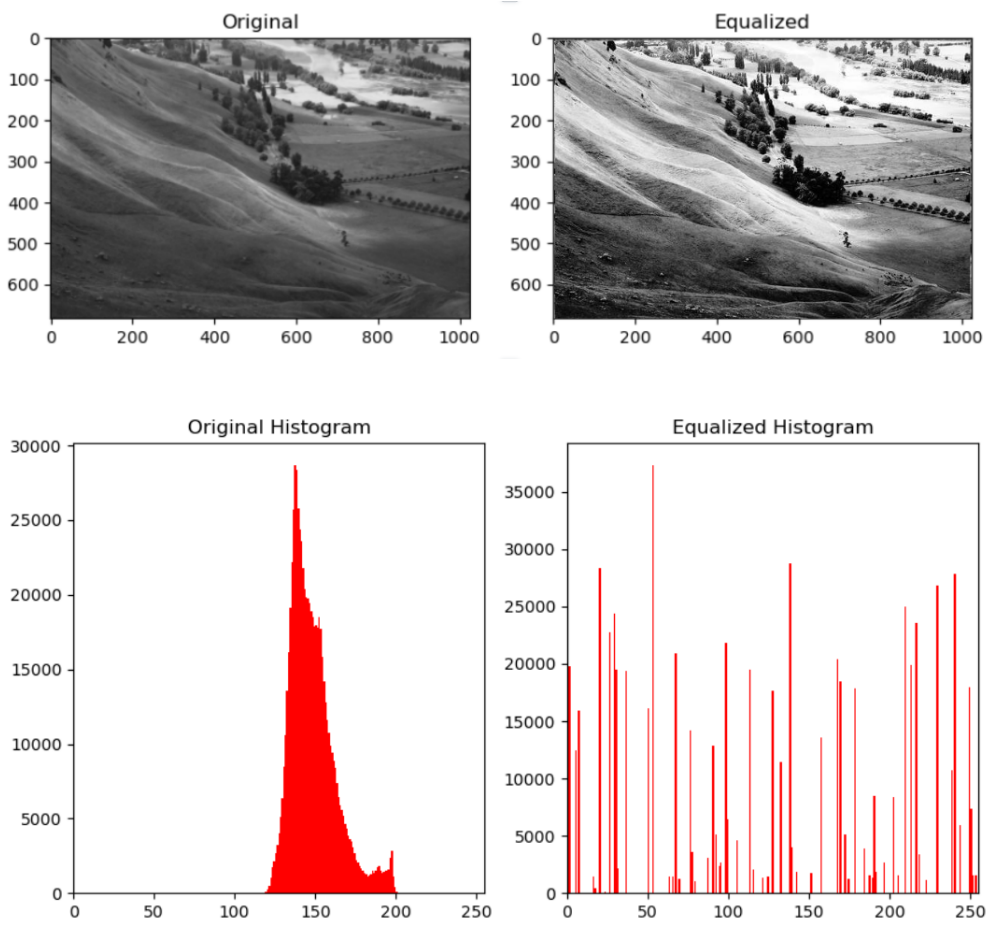
Or $\sum_{j=0}^{k} p_x(x_j)$ is the cumulative histogram which can also be written :

$$T(x_k) = \frac{(L-1)}{n} \sum_{j=0}^{k} n_j$$

**** Code**

```
    # Load image and convert to grayscale
img = Image.open('equalization.jpg').convert('L')
img_array = np.asarray(img)

# Calculate histogram
hist, bins = np.histogram(img_array.flatten(), 256, [0, 256])

# Calculate cumulative distribution function
cdf = hist.cumsum()
cdf_normalized = cdf * hist.max() / cdf.max()

# Create lookup table
lookup_table = np.interp(np.arange(256), bins[:-1], cdf_normalized)

# Apply lookup table to image
img_eq = np.interp(np.array(img_array), np.arange(256),
    lookup_table).astype('uint8')

# Calculate equalized histogram
hist_eq, _ = np.histogram(img_eq.flatten(), 256, [0, 256])

# Display original and equalized images using subplots
fig, axs = plt.subplots(1, 2, figsize=(10,5))

axs[0].imshow(img, cmap='gray')
axs[0].set_title('Original')

axs[1].imshow(img_eq, cmap='gray')
axs[1].set_title('Equalized')

```

```
# Display histograms
fig, axs = plt.subplots(1, 2, figsize=(10,5))

axs[0].hist(img_array.flatten(), bins=256, range=(0, 256), color='r')
axs[0].set_title('Original Histogram')
axs[0].set_xlim([0, 255])

axs[1].hist(img_eq.flatten(), bins=256, range=(0, 256), color='r')
axs[1].set_title('Equalized Histogram')
axs[1].set_xlim([0, 255])

plt.show()
```



3.2.2 Low-pass filters

Low-pass filters are used to reduce high frequencies in the image, which can help eliminate noise and improve the sharpness of the image. Low-pass filters

can be used to increase the contrast and brightness of the image by reducing shadows and increasing detail in dark areas.

*** Method**

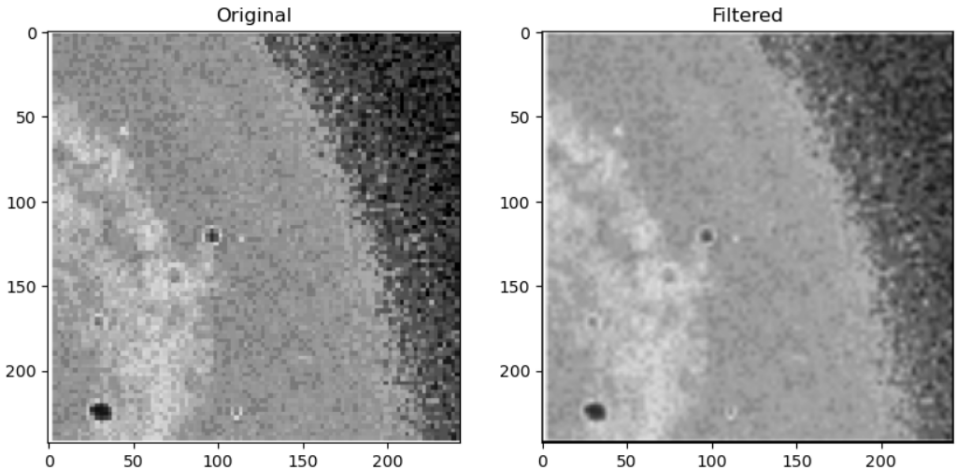
A low pass filter is the basis for most smoothing methods. An image is smoothed by decreasing the disparity between pixel values by averaging nearby pixels. Using a low pass filter tends to retain the low frequency information within an image while reducing the high frequency information. An example of a low pass filter is an array of ones divided by the number of elements within the kernel, such as the following 3 by 3 kernel :

$$\begin{pmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{pmatrix}$$

The above array is just an example of one possible kernel for a low pass filter. Other filters may include more weighting for the center point, or have different smoothing in each dimension.

**** Code**

```
    # Load image and convert to grayscale
img = Image.open('low_pass.png').convert('L')

# Convert image to numpy array
img_array = np.asarray(img)

# Define kernel for low pass filter
kernel = np.array([[1, 1, 1],
                   [1, 1, 1],
                   [1, 1, 1]]) / 9

# Apply filter to image
filtered_img = np.zeros_like(img_array)
for i in range(1, img_array.shape[0]-1):
    for j in range(1, img_array.shape[1]-1):
        filtered_img[i,j] = np.sum(img_array[i-1:i+2,j-1:j+2] *
    kernel)

# Display original and filtered images using subplots
fig, axs = plt.subplots(1, 2, figsize=(10,5))

axs[0].imshow(img, cmap='gray')
axs[0].set_title('Original')

axs[1].imshow(filtered_img, cmap='gray')
axs[1].set_title('Filtered')

plt.show()
```



3.2.3 High-pass filters

High-pass filters are used to increase the high frequencies of the image, which can help to increase the sharpness and detail of the image. High-pass filters can be used to increase the contrast of the image by sharpening edges and contours.

*** Method**

A high pass filter is the basis for most sharpening methods. An image is sharpened when contrast is enhanced between adjoining areas with little variation in brightness or darkness. A high pass filter tends to retain the high frequency information within an image while reducing the low frequency information. The kernel of the high pass filter is designed to increase the brightness of the center pixel relative to neighboring pixels. The kernel array usually contains a single positive value at its center, which is completely surrounded by negative values. The following array is an example of a 3 by 3 kernel for a high pass filter :

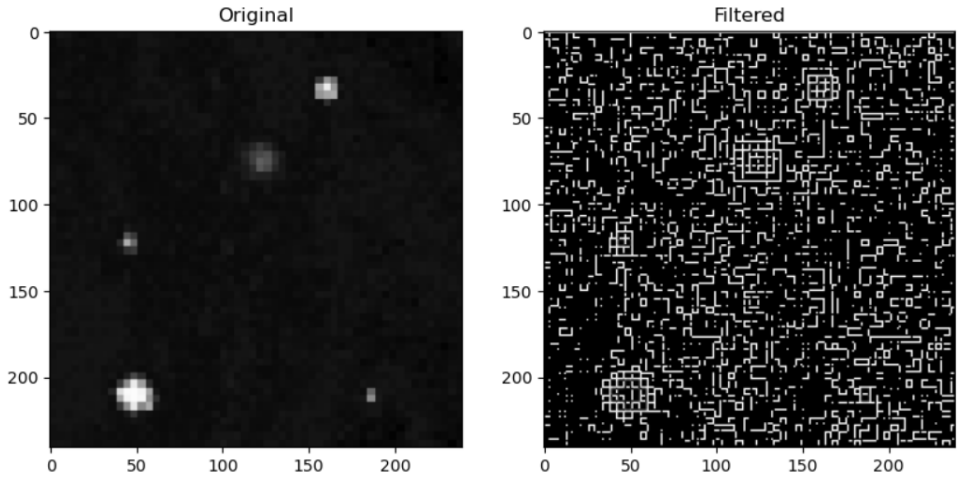
$$\begin{pmatrix} -1/9 & -1/9 & -1/9 \\ -1/9 & 8/9 & -1/9 \\ -1/9 & -1/9 & -1/9 \end{pmatrix}$$

The above array is just an example of one possible kernel for a high pass filter. Other filters may include more weighting for the center point.

**** Code**

```
# Load image and convert to grayscale
img = Image.open('high_pass.png').convert('L')

# Convert image to numpy array
img_array = np.asarray(img)

# Define kernel for high pass filter
kernel = np.array([[-1, -1, -1],
```

```
                    [-1,  8, -1],
                    [-1, -1, -1]]) / 9

# Apply filter to image
filtered_img = np.zeros_like(img_array)
for i in range(1, img_array.shape[0]-1):
    for j in range(1, img_array.shape[1]-1):
        filtered_img[i,j] = np.sum(img_array[i-1:i+2,j-1:j+2] *
    kernel)

# Display original and filtered images using subplots
fig, axs = plt.subplots(1, 2, figsize=(10,5))

axs[0].imshow(img, cmap='gray')
axs[0].set_title('Original')

axs[1].imshow(filtered_img, cmap='gray')
axs[1].set_title('Filtered')

plt.show()
```



3.2.4 Enhancement filters : Laplace filter

*** Method**

Laplace filter is a type of edge detection filter that is used to highlight the edges of an image. The filter works by identifying regions of the image where the intensity changes rapidly, which typically correspond to the edges of objects in the image.

The Laplace filter is based on the second derivative of the image, and it is defined by a 3x3 kernel. The center pixel of the kernel has a negative value, while the surrounding pixels have positive values. The Laplace filter is typically

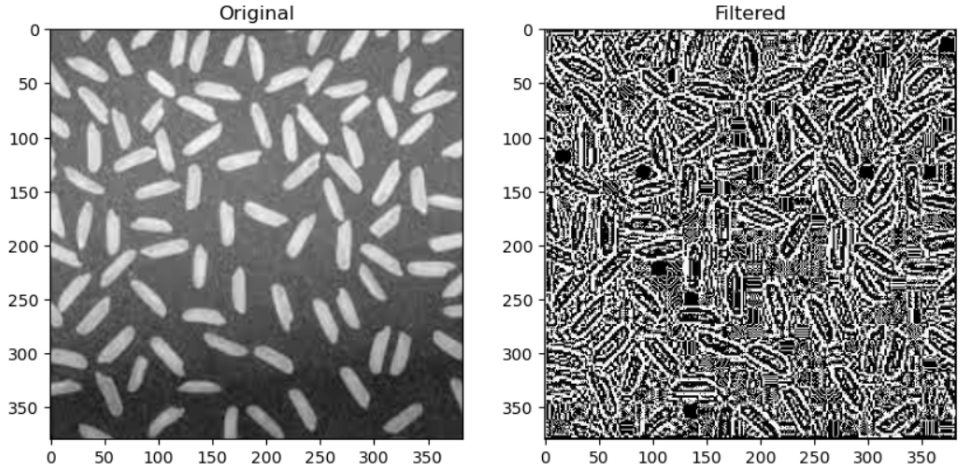
applied to a grayscale image, but it can also be applied to a color image by first converting it to grayscale.

The following array is an example of a 3x3 kernel for a Laplacian filter.

$$\begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$

**** Code**

```
# Load image and convert to grayscale
img = Image.open('rice.png').convert('L')

# Convert image to numpy array
img_array = np.asarray(img)

# Define kernel for Laplacian filter
kernel = np.array([[0, -1, 0],
                   [-1, 4, -1],
                   [0, -1, 0]])

# Apply filter to image
filtered_img = np.zeros_like(img_array)
for i in range(1, img_array.shape[0]-1):
    for j in range(1, img_array.shape[1]-1):
        filtered_img[i,j] = np.sum(img_array[i-1:i+2,j-1:j+2] *
    kernel)

# Display original and filtered images using subplots
fig, axs = plt.subplots(1, 2, figsize=(10,5))

axs[0].imshow(img, cmap='gray')
axs[0].set_title('Original')

axs[1].imshow(filtered_img, cmap='gray')
axs[1].set_title('Filtered')

plt.show()
```



3.2.5 Colour Filters

Colour filters can be used to increase or decrease the saturation of the image, change the colour balance or add colour filters to create artistic effects. Colour filters can be used to improve the overall quality of the image by adjusting the colour temperature, saturation and brightness.

Color filters can be used to modify the colors of an image or to isolate specific color channels. Here are some examples of color filters with their kernels :

Red filter :

$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

This filter keeps only the red channel of an image by setting the green and blue channels to 0.

Green filter :

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

This filter keeps only the green channel of an image by setting the red and blue channels to 0.

Blue filter :

$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$

This filter keeps only the blue channel of an image by setting the red and green channels to 0.

Inverse filter :

$$\begin{pmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{pmatrix}$$

This filter is used to invert the colors of an image by subtracting the average of the neighboring pixels from the center pixel.

Sepia filter :

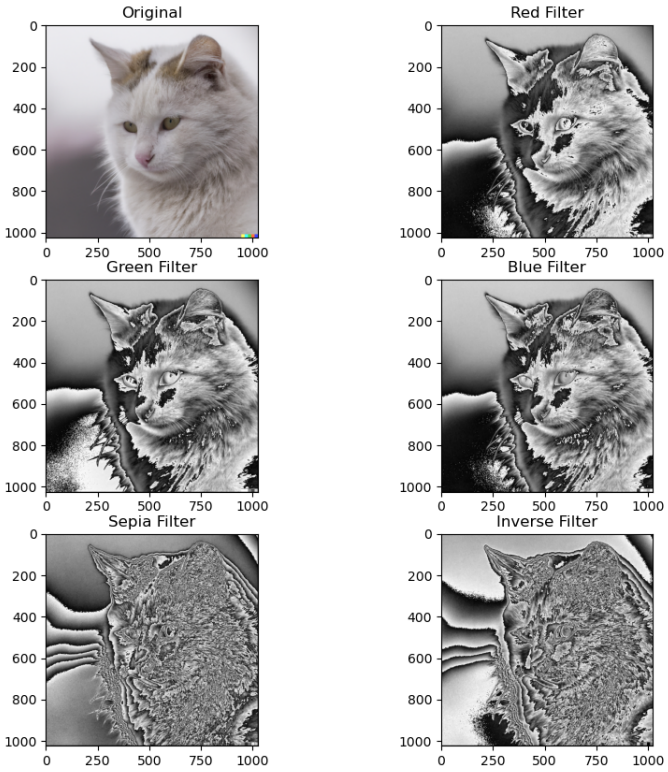
$$\begin{pmatrix} 0.393 & 0.769 & 0.189 \\ 0.349 & 0.686 & 0.168 \\ 0.272 & 0.534 & 0.131 \end{pmatrix}$$

This filter gives an image a vintage look by converting the colors to sepia tones. It multiplies the red channel by 0.393, green channel by 0.769, and blue channel by 0.189 for the top row ; red by 0.349, green by 0.686, and blue by 0.168 for the middle row ; and red by 0.272, green by 0.534, and blue by 0.131 for the bottom row.

**** Code**

```
# Load image
img = Image.open('01.png')

# Define color filter kernels
red_kernel = np.array([[1, 0, 0],
                       [0, 0, 0],
                       [0, 0, 0]])

green_kernel = np.array([[0, 0, 0],
                         [0, 1, 0],
                         [0, 0, 0]])

blue_kernel = np.array([[0, 0, 0],
                        [0, 0, 0],
                        [0, 0, 1]])

# Define sepia kernel
sepia_kernel = np.array([[0.393, 0.769, 0.189],
                         [0.349, 0.686, 0.168],
                         [0.272, 0.534, 0.131]])

# Define inverse kernel
inverse_kernel = np.array([[-1, 0, 0],
                           [0, -1, 0],
                           [0, 0, -1]])

# Convert image to numpy array
img_array = np.asarray(img)

# Apply each color filter to image
filtered_red = np.zeros_like(img_array)
filtered_green = np.zeros_like(img_array)
filtered_blue = np.zeros_like(img_array)
filtered_sepia = np.zeros_like(img_array)
```

```
filtered_inverse = np.zeros_like(img_array)

for i in range(1, img_array.shape[0]-1):
    for j in range(1, img_array.shape[1]-1):
        filtered_red[i,j] = np.sum(img_array[i-1:i+2,j-1:j+2] *
    red_kernel)
        filtered_green[i,j] = np.sum(img_array[i-1:i+2,j-1:j+2] *
    green_kernel)
        filtered_blue[i,j] = np.sum(img_array[i-1:i+2,j-1:j+2] *
    blue_kernel)
        filtered_sepia[i,j] = np.sum(img_array[i-1:i+2,j-1:j+2] *
    sepia_kernel)
        filtered_inverse[i,j] = np.sum(img_array[i-1:i+2,j-1:j+2] *
     inverse_kernel)

# Display original and filtered images using subplots
fig, axs = plt.subplots(3, 2, figsize=(10,10))

axs[0,0].imshow(img)
axs[0,0].set_title('Original')

axs[0,1].imshow(filtered_red, cmap='hot')
axs[0,1].set_title('Red Filter')

axs[1,0].imshow(filtered_green, cmap='hot')
axs[1,0].set_title('Green Filter')

axs[1,1].imshow(filtered_blue, cmap='hot')
axs[1,1].set_title('Blue Filter')

axs[2,0].imshow(filtered_sepia, cmap='hot')
axs[2,0].set_title('Sepia Filter')

axs[2,1].imshow(filtered_inverse, cmap='hot')
axs[2,1].set_title('Inverse Filter')

plt.show()
```



It is important to note that each filter can have different effects on the image and it is important to choose the appropriate filter for the desired effect. It is also important to use filters sparingly to avoid loss of image quality.

3.3 Comparison of Image Enhancement Techniques

The choice of an image enhancement technique depends on the specific requirements of the application. For example, if the goal is to enhance the contrast of an image, histogram equalization or low-pass filters can be used. On the other hand, if the goal is to enhance the edges and details of an image, high-pass filters or Laplace filters can be used.

Color filters can be used to enhance the color of an image, but they can also be used to selectively filter out certain colors. This can be useful in applications such as medical imaging, where certain organs or tissues may be highlighted by the use of specific color filters.

In conclusion, image enhancement techniques play a vital role in improving the quality of digital images. Each technique has its strengths and weaknesses, and the choice of a technique depends on the specific requirements of the application. A careful analysis of the image and the intended application is necessary to choose the most appropriate technique.

3.4 Conclusion of Image enhancement

In conclusion, image enhancement is a crucial process that improves the quality and visual appearance of digital images. Various techniques can be employed to enhance images, including linear algebra-based techniques such as histogram equalization, low-pass and high-pass filters, and enhancement filters like the Laplace filter. Color filters can also be used to enhance the colors and contrast of images. Each technique has its strengths and weaknesses, and the choice of the appropriate technique depends on the specific image characteristics and the desired outcome.

Moreover, image enhancement plays a vital role in various fields, including medical imaging, satellite imaging, and digital photography. It helps in improving the visibility of important features, reducing noise and artifacts, and enhancing the overall image quality, leading to more accurate analysis and interpretation.

However, it is essential to use image enhancement techniques with caution to avoid over-processing and distortion of images. Additionally, the quality of the original image also plays a critical role in the effectiveness of enhancement techniques. In summary, image enhancement is a powerful tool that can significantly improve the visual appearance and quality of digital images, but its use should be judicious and guided by the specific characteristics and desired outcomes of the images.

4 Conclusion

In conclusion, image processing is a fundamental aspect of computer vision and has a wide range of applications in various fields, such as medical imaging, robotics, and security systems. This project focused on two major aspects of image processing : image denoising and image enhancement.

For image denoising, we discussed several techniques based on linear algebra, including singular value decomposition, regularization, and filtering. We also compared the effectiveness of various filtering methods, such as the Gaussian, median, and Wiener filters, and found that each technique has its advantages and disadvantages depending on the type and severity of noise present in the image.

In image enhancement, we explored several techniques based on linear algebra, including histogram equalization, low-pass and high-pass filters, Laplace filters, and color filters. We also compared the effectiveness of each technique and found that histogram equalization and color filters are particularly useful for enhancing the contrast and color balance of images.

Overall, this project provided a comprehensive overview of image denoising and image enhancement techniques based on linear algebra. By understanding these techniques and their applications, researchers and engineers can continue to advance the field of image processing and develop innovative solutions to a variety of real-world problems.

Références

[1] Github https ://vincmazet.github.io/bip/restoration/denoising.html *Basics of image processing*

[2] Phillip K Poon, Wei-Ren Ng, Varun Sridharan *Image Denoising with Singular Value Decompositon and Principal Component Analysis*

[3] https ://www.frankcleary.com/svdimage/ *Singular Value Decomposition of an Image*

[4] L3HARRIS https ://www.l3harrisgeospatial.com *Low Pass Filtering*

[5] L3HARRIS https ://www.l3harrisgeospatial.com *High Pass Filtering*

[6] Wikipedia https ://fr.wikipedia.org/wiki/%C3%89galisation$_d$%27*histogrammeHistogram equalization*