Engineering Mathematics Project

Instructor: Prof. Hamid Aghajan

Sharif University of Technology

From Noise to Clarity: A Study of Filtering Techniques in Biomedical Imaging

Authors:
Mohammadparsa Ghaderahmadi mhparsaghdi@gmail.com
Matin M.Babaei matinmb82@gmail.com

By: Moein Yousefinia
moein.yoo84@sharif.edu
moein_yoo@outlook.com

Winter 1403

Preface

Notes on the project:

- **Due date:** 1403/11/12
- The project must be done individually. Each individual will present their results in an online session. Exact date of the online sessions will be announced.
- Please submit your project report as a `.pdf` file. Include all outputs and final results in the report. Make sure to list the practice test questions and provide a concise explanation of your problem-solving approach in each section.
- Ensure that all codes are provided in a separate `.m/.py/.ipynb` file. If a code cannot be tested accurately upon submission, the reported results will be considered invalid, and no points will be awarded in such cases.
- You have the flexibility to utilize either **MATLAB** or **Python** for your project.
- Ensure that you save all files, including your report, codes, helper functions, and any additional outputs, if required, in a compressed file format such as `.zip` or `.rar`. This compressed file should then be uploaded to the CourseWare (CW) submission platform.
- Your file names must be in the following format:

  `Project.#StudentID.zip/.rar/.pdf/.m/.py/.ipynb`

- The details of the grading system of this project will be provided in the coming days. Generally, the project is worth a total of 1 point, which could account for the grades lost in the final exam.
- In this project, it is essential to uphold the principles of academic integrity and refrain from any form of cheating or copying. Cheating undermines the learning process, diminishes personal growth, and compromises the trust placed in us as students/researchers/professionals. It is crucial to recognize that engaging in dishonest practices not only tarnishes our own reputation but also has serious consequences, both ethically and academically. **We emphasize that if anyone is found to have cheated, their results will not be accepted in this project, and they will receive a zero mark.**

Contents

1 Introduction

Biomedical image processing is a specialized field focused on the analysis and manipulation of medical and biological images to enhance their quality and extract valuable information. It plays a crucial role in diagnostics, treatment planning, and medical research by utilizing images from modalities such as MRI, CT scans, ultrasound, and microscopy. The primary steps involved include image acquisition, preprocessing (such as noise reduction and artifact correction), enhancement, segmentation, feature extraction, and classification.

Filters are integral to biomedical image processing, aiding in tasks like noise reduction, edge detection, and feature enhancement to improve image clarity and accuracy. Common types of filters used include smoothing filters (e.g., Gaussian and median filters), edge detection filters (e.g., Sobel and Canny), sharpening filters, and specialized filters like Gabor and wavelet transforms. These tools help in isolating relevant structures, highlighting important details, and ensuring that the images are suitable for accurate analysis and interpretation.

This project investigates various filtering techniques and their impact on biomedical images, focusing on noise reduction and feature preservation. The study also evaluates filter performance using quantitative measures, providing insights into optimal methods for specific image processing challenges.

1.1 Goal of the Project

The primary objectives of this project are:

- To explore the fundamental aspects of biomedical image processing, including noise modeling and filtering techniques.
- To implement and evaluate various filters such as mean, Gaussian, bilateral, and adaptive filters on noisy biomedical images.
- To analyze and compare the effectiveness of these filters using quantitative metrics like Signal-to-Noise Ratio (SNR) and Peak Signal-to-Noise Ratio (PSNR).

2 Aspects of an Image

In this part we want to investigate through different properties of an image. In image processing, analyzing different aspects helps us understand and manipulate visual information. Among these aspects we can mention Magnitude, Phase and Frequency domain. To gain a better understanding of this issue, experiments will be conducted.

2.1 Questions

Using pic1.jpg answer questions below:

1. Intuitively, What do low and high frequencies of an arbitrary image contain? Which one is more important? Do you think your answer depends on the image? Explain it.

Answer

In high frequencies, sudden intensity variations, edges, and fine details of the image are captured, while in low frequencies, the overall structure, continuous color spectra, and gradual intensity variations are more prominent.
The importance of each frequency component depends on the specific application. For example:

- In **image compression**, high-frequency components can be partially removed to reduce data size without significantly altering the overall perception of the image.
- In **computer vision and object recognition**, high-frequency information (such as edges) is often more critical.
- In **image reconstruction and noise reduction**, preserving low-frequency components is essential for maintaining the general structure of the image.

2. Explain about low pass filter(LPF) and high pass filter(HPF) . In what applications LPF and HPF are used? Write two functions that apply LPF and HPF to an arbitrary image.

Answer

A **Low-Pass Filter (LPF)** allows low-frequency components to pass while attenuating high-frequency components. In image processing, LPFs smooth images by reducing noise and blurring fine details.
Applications of LPF:

- **Image smoothing and blurring** (e.g., Gaussian blur)
- **Noise reduction** (e.g., removing high-frequency noise)
- **Resampling and anti-aliasing** (avoiding jagged edges in image scaling)
- **Medical image processing** (e.g., denoising MRI or X-ray images)

A **High-Pass Filter (HPF)** allows high-frequency components to pass while attenuating low-frequency components. In image processing, HPFs enhance edges and fine details by removing smooth background variations.
Applications of HPF:

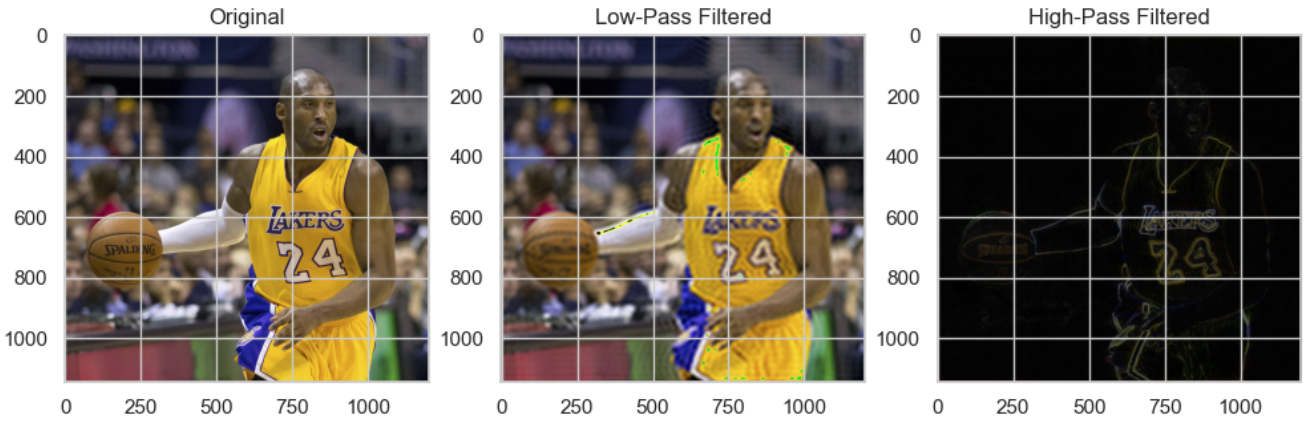
- **Edge detection** (e.g., Sobel, Prewitt, and Laplacian filters)
- **Feature extraction** (e.g., extracting textures in object detection)
- **Image sharpening** (e.g., enhancing details in blurred images)
- **Medical imaging** (e.g., detecting boundaries in ultrasound or MRI scans)

```python
def filterOfChannel(channel, mask_type, cutoff):
    dft = np.fft.fft2(channel)
    dft_shift = np.fft.fftshift(dft)
    rows, cols = channel.shape
    crow, ccol = rows // 2, cols // 2 # center
    mask = np.ones((rows, cols), np.uint8) if mask_type == "HPF"
       else np.zeros((rows, cols), np.uint8)
    if mask_type == "LPF":
        mask[crow - cutoff:crow + cutoff, ccol - cutoff:ccol +
           cutoff] = 1
    else:
        mask[crow - cutoff:crow + cutoff, ccol - cutoff:ccol +
           cutoff] = 0
    dft_shift *= mask
    dft_ishift = np.fft.ifftshift(dft_shift)
    filtered_channel = np.fft.ifft2(dft_ishift)
    filtered_channel = np.abs(filtered_channel)
    return filtered_channel

def lowPassFilter(image, cutoff):
    b, g, r = cv2.split(image)
    b_filtered = filterOfChannel(b, "LPF", cutoff)
    g_filtered = filterOfChannel(g, "LPF", cutoff)
    r_filtered = filterOfChannel(r, "LPF", cutoff)
    return cv2.merge([b_filtered, g_filtered, r_filtered])
```

```
23
24  def highPassFilter(image, cutoff):
25      b, g, r = cv2.split(image)
26      b_filtered = filterOfChannel(b, "HPF", cutoff)
27      g_filtered = filterOfChannel(g, "HPF", cutoff)
28      r_filtered = filterOfChannel(r, "HPF", cutoff)
29      return cv2.merge([b_filtered, g_filtered, r_filtered])
```

3. Apply LPF and HPF to pic1.jpg using functions written in the previous part. Explain the results.

Figure 1: Kobe Bryant (August 23, 1978 – January 26, 2020)

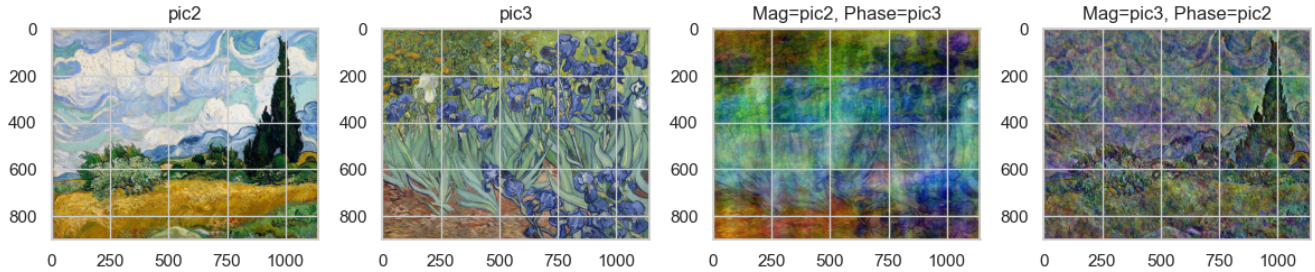


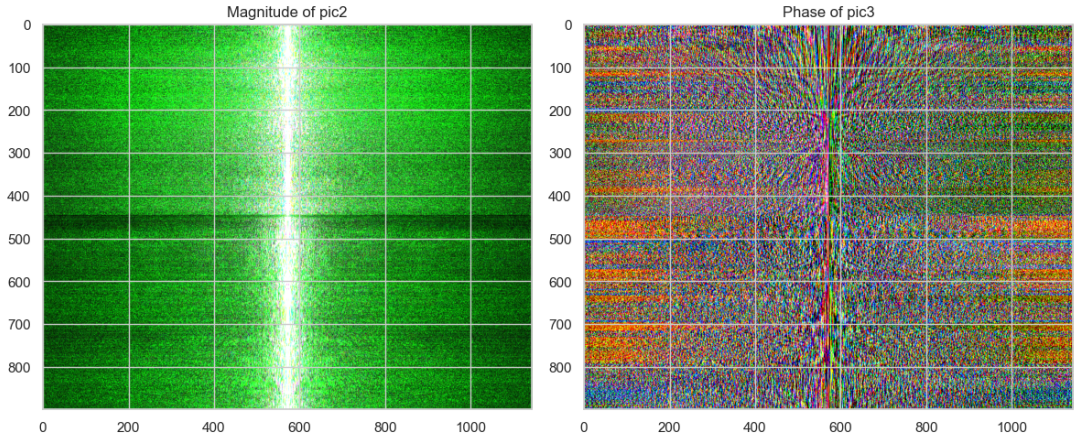
Using pic2.jpg and pic3.jpg answer questions below:

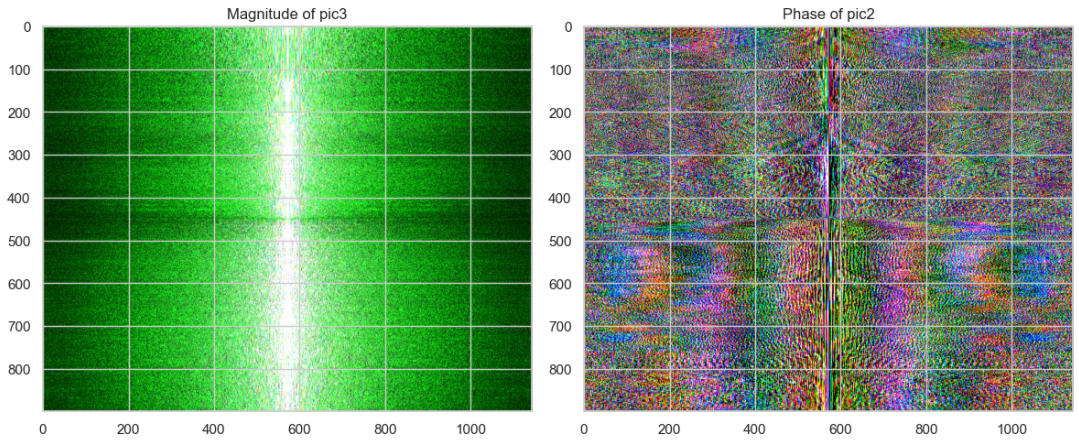
1) Plot the magnitude and phase of both images. Then, plot an image with magnitude of pic2.jpg and phase of pic3.jpg. Plot the other alternate.

```
def swapMP(channel1, channel2, method):
    dft1 = np.fft.fft2(channel1)
    dft2 = np.fft.fft2(channel2)
    if method == 1:
        dft_combined = np.abs(dft1) * np.exp(1j * np.angle(dft2))
    else:
        dft_combined = np.abs(dft2) * np.exp(1j * np.angle(dft1))
    channel_reconstructed = np.fft.ifft2(dft_combined)
    channel_reconstructed = np.abs(channel_reconstructed)
    channel_reconstructed = cv2.normalize(channel_reconstructed,
       None, 0, 255, cv2.NORM_MINMAX)
    return channel_reconstructed.astype(np.uint8)

def createImage(image1, image2, method):
    b1, g1, r1 = cv2.split(image1)
    b2, g2, r2 = cv2.split(image2)
    b_reconstructed = swapMP(b1, b2, method)
    g_reconstructed = swapMP(g1, g2, method)
    r_reconstructed = swapMP(r1, r2, method)
    return cv2.merge([b_reconstructed, g_reconstructed,
       r_reconstructed])

image1 = cv2.imread("pics/pic2.jpg")
image1 = cv2.cvtColor(image1, cv2.COLOR_BGR2RGB)
image2 = cv2.imread("pics/pic3.jpg")
image2 = cv2.cvtColor(image2, cv2.COLOR_BGR2RGB)
image2 = cv2.resize(image2, (image1.shape[1], image1.shape[0]))

image1_reconstructed = createImage(image1, image2, 1)
image2_reconstructed = createImage(image1, image2, 2)

plt.figure(figsize=(12, 12))
plt.subplot(1,4,1), plt.imshow(image1), plt.title('pic2')
plt.subplot(1, 4, 2), plt.imshow(image2), plt.title('pic3')
plt.subplot(1, 4, 3), plt.imshow(image1_reconstructed.astype(np.
   uint8)), plt.title('Mag=pic2, Phase=pic3')
plt.subplot(1, 4, 4), plt.imshow(image2_reconstructed.astype(np.
   uint8)), plt.title('Mag=pic3, Phase=pic2')

plt.tight_layout()
plt.show()
```

2) Using the results above, Explain Why is the phase information more critical for reconstructing recognizable features of an image compared to the magnitude information?







Answer

1. Role of Phase Information

- **Phase** determines the *positioning* of image features like edges and textures.
- Swapping the **phase** between two images results in an output that resembles the image whose phase was used.

2. Role of Magnitude Information

- **Magnitude** represents the strength of frequencies but not their location.
- Affects overall **brightness** and **contrast**.

Therefore, in image reconstruction, **phase information is more critical** for maintaining recognizable features.

(a) Wheat Field with Cypresses

(b) Irises

3 Noise

In this part we explore more about Noise. Noise in digital images and signals is an unwanted disturbance that affects quality and can originate from various sources. It can appear in different forms, such as salt-and-pepper noise, Gaussian noise and 50 HZ noise. Each type of noise requires specific filtering techniques for reduction, and understanding the nature of noise is crucial to effectively restoring signal or image quality. The following questions will help you better track the rest of the project.

3.1 Questions

1. What are the primary sources of salt-and-pepper noise in digital images, and what filtering techniques are most effective in mitigating this type of noise?

Answer

- **Intro** Salt-and-pepper noise, also known as impulse noise, is a form of noise sometimes seen on digital images. For black-and-white or grayscale images, is presents as sparsely occurring white and black pixels, giving the appearance of an image sprinkled with salt and pepper.
- **Cause** Salt-and-pepper noise can be caused by sharp and sudden disturbances in the image signal. These may be from transmission errors, corrupted pixel elements in the camera sensors, or faulty memory locations in the storage media.
- **Removal** An effective noise reduction method for this type of noise is a median filter[2] or a morphological filter.[3] For reducing either salt noise or pepper noise, but not both, a contraharmonic mean filter can be effective. Linear filters are generally ineffective for removing impulse noise.

2. How does electrical interference introduce 50Hz noise into signal recordings, and what methods can be employed to eliminate or reduce this interference?

Answer

Electrical interference at 50Hz is commonly caused by power line interference, which affects sensitive signal recordings. The primary sources of 50Hz noise include:

- **Electromagnetic Coupling with Power Lines**

  Power lines generate alternating electromagnetic fields at 50Hz. Nearby electronic circuits, especially low-power and high-impedance ones, can pick up these fields through inductive (magnetic) or capacitive (electric) coupling.

- **Ground Loops**

  When multiple devices share an improper ground connection, current can circulate between them, introducing 50Hz noise.

- **Nearby Electrical Devices**

  Fluorescent lights, transformers, motors, and switching power supplies can emit electromagnetic noise at 50Hz and its harmonics.

- **Poor Shielding in Cables**

  If cables carrying the signal are unshielded or poorly shielded, they act as antennas, picking up 50Hz interference. Mains Leakage in Power Supplies

- Low-quality power adapters can leak 50Hz noise into the signal ground, affecting the recordings.

3. Identify the common sources of Gaussian noise in imaging systems and discuss the most suitable algorithms for its removal without significantly degrading image quality.

Answer

- **Intro** Principal sources of Gaussian noise in digital images arise during acquisition e.g. sensor noise caused by poor illumination and/or high temperature, and/or transmission e.g. electronic circuit noise.

- **Removal** In digital image processing Gaussian noise can be reduced using a spatial filter, though when smoothing an image, an undesirable outcome may result in the blurring of fine-scaled image edges and details because they also correspond to blocked high frequencies. Conventional spatial filtering techniques for noise removal include: mean (convolution) filtering, median filtering and Gaussian smoothing.

4. Explore SNR and PSNR. What roles do Signal-to-Noise Ratio (SNR) and Peak Signal-to-Noise Ratio (PSNR) play in evaluating the effectiveness of noise removal techniques in digital images? Compare and contrast the effectiveness of SNR and PSNR in different applications such as medical imaging, video compression, and wireless communications.

Answer

- **Definition of SNR**

  Signal-to-Noise Ratio (SNR) is a key metric in image processing, particularly in medical imaging, where noise can impact diagnosis. It is defined as:

$$SNR = \frac{E_x}{E_n} \tag{1}$$

  where:

  - E_x represents the expected energy of the image signal.
  - E_n represents the expected energy of noise in the image.

  A higher SNR indicates better image quality with reduced noise.

- **Definition of PSNR**

  Peak Signal-to-Noise Ratio (PSNR) is commonly used in video compression and image denoising:

$$PSNR = 10 \log_{10} \left(\frac{MAX^2}{MSE} \right) \tag{2}$$

  where:

  - MAX is the maximum pixel intensity value.
  - MSE (Mean Squared Error) represents the average squared difference between the original and noisy image.

- **Comparison of SNR and PSNR**

Application	SNR	PSNR
Medical Imaging (MRI)	Measures overall signal clarity	Less relevant
Video Compression	Less common	Key metric for measuring quality
Wireless Communications	Critical for signal detection	Rarely used

- **Conclusion**

  In MRI and other medical imaging applications, **SNR** is the preferred metric, as it directly reflects the clarity of anatomical structures. In contrast, **PSNR** is more relevant for video compression and image restoration techniques.

3.2 Noise Modeling

In practice, when working with real-world signals, it is often observed that the received signal differs from the theoretically expected one. To model this phenomenon, we can consider the received signal as the superposition of two signals:

- The original signal, which is theoretically expected.

- An unwanted signal, which is beyond our control and is referred to as noise.

If we denote the original signal as $x(t)$, the noise as $n(t)$, and the received signal as $x_n(t)$, we can model their relationship in two common ways, corresponding to additive and multiplicative noise:

$$x_n(t) = x(t) + n(t), \tag{3}$$
$$x_n(t) = x(t) \cdot n(t). \tag{2}$$

If the values of the noise $n(t)$ were known over time, we could easily recover the original signal using the relations $x(t) = x_n(t) - n(t)$ or $x(t) = \frac{x_n(t)}{n(t)}$. However, the main issue is that noise is inherently random, and its exact values at each moment cannot be precisely determined. Therefore, we model noise at each moment as a random variable with a specified distribution. For example, two proposed distributions for noise could be:

$$n(t) \sim U(-a, a), \tag{3}$$
$$n(t) \sim \mathcal{N}(0, \sigma^2). \tag{4}$$

In this section, we will first generate several types of noise and examine their properties as well as their superposition with noiseless signals.

3.2.1 Function Definition

Write a function named `noise` that takes three parameters a, L, and `type`, and works as follows:

- If the `type` input is the string `"uniform"`, the function returns a vector of length L where each component is a random variable from the distribution $U(-a, a)$.
- If the `type` input is the string `"normal"`, the function returns a vector of length L where each component is a random variable from the distribution $N(0, a^2)$.

```
def noise(a, L, type):
    if type == "Normal":
        return np.random.normal(0, a, L)
    elif type == "Uniform":
        return np.random.uniform(-a, a, L)
```

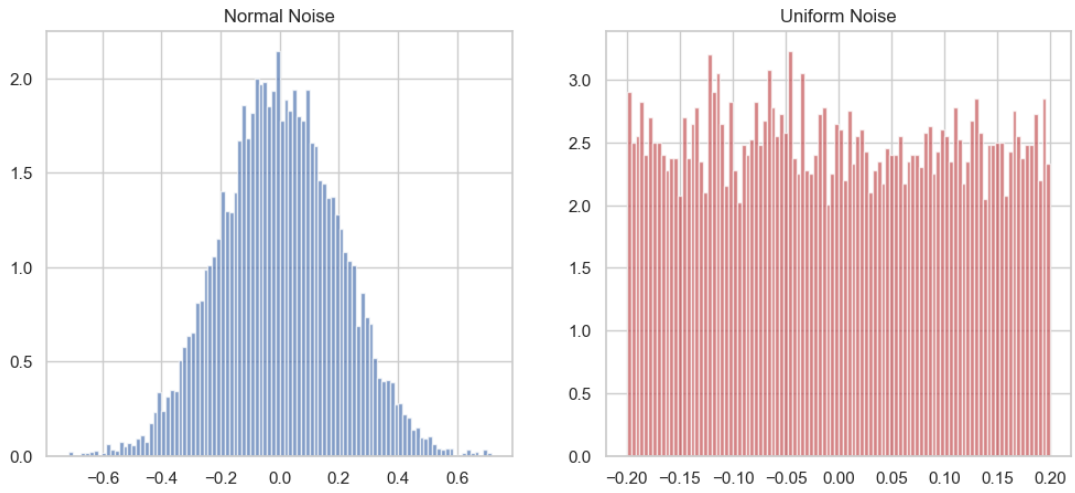
3.2.2 Statistical Analysis of Noise Vectors

Before adding noise to the original signal, we observe some statistical properties of the noise and compare them with theoretical expectations.

(a) Generate and Plot Noise Vectors

Generate two noise vectors N_1 and N_2 with $a = 0.2$, using uniform and normal distributions respectively, with $L = 10000$. Plot the histogram of each vector (using MATLAB's `histogram` function). Do the resulting shapes confirm the expected probability distributions?

```
a = 0.2
L = 10000
N1 = noise(a, L, "Normal")
N2 = noise(a, L, "Uniform")
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1), plt.hist(N1, bins=100, density=True, color=
    'b', alpha=0.7), plt.title('Normal Noise')
plt.subplot(1, 2, 2), plt.hist(N2, bins=100, density=True, color=
    'r', alpha=0.7), plt.title('Uniform Noise')
plt.grid(True)
plt.show()
```



Yes,It resembles what we really want.

(b) Mean of Noise Vectors

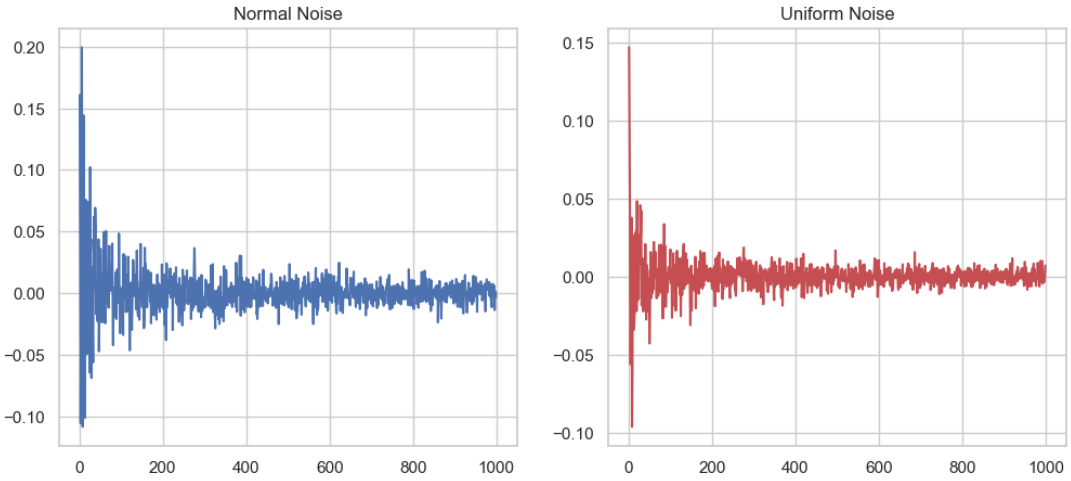
For the noise vector N_1, define:

$$N_1 = \frac{1}{L} \sum_{i=1}^{L} N_1[i]. \tag{5}$$

Compute N_1 for $L \in \{1, 2, \ldots, 1000\}$ and plot it as a function of L. Repeat the same for N_2. What is the limit of N_1 and N_2 as $L \to \infty$? Which theorem or statement in probability theory precisely explains this result?

```
a = 0.2
L = np.arange(0, 1000, 1)
```

```
means_normal = []
means_uniform = []
for l in L:
    N1 = noise(a, l, "Normal")
    N2 = noise(a, l, "Uniform")
    means_normal.append(np.mean(N1))
    means_uniform.append(np.mean(N2))
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1), plt.plot(L, means_normal, color='b'), plt.title('Normal Noise')
plt.subplot(1, 2, 2), plt.plot(L, means_uniform, color='r'), plt.title('Uniform Noise')
plt.grid(True)
plt.show()
```



It uses CLT with Sample Mean in Statistics which prove as $L \to \infty$ the $\mu_X = \mu$.

(c) Energy of Noise Vectors

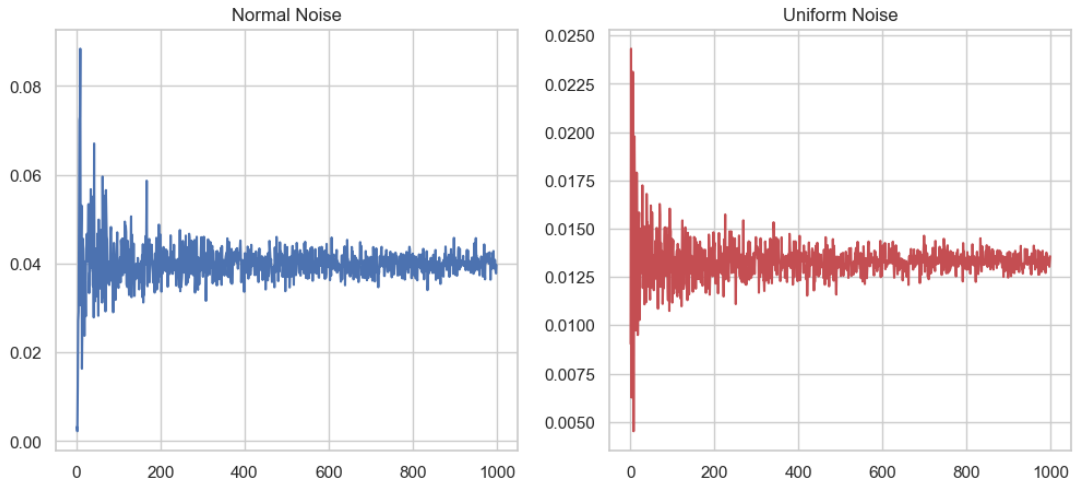
Consider the following definition of the energy of a noise vector N:

$$E_N = \frac{1}{L} \sum_{i=1}^{L} (N[i])^2 . \tag{6}$$

For each of the two noise vectors N_1 and N_2, compute the energy as a function of L for $L \in \{1, 2, \ldots, 1000\}$, and plot it. What is the limit of E_{N_1} and E_{N_2} as $L \to \infty$? Verify your result through simulation.

```
a = 0.2
L = np.arange(0, 1000, 1)
energy_normal = []
energy_uniform = []
for l in L:
    N1 = noise(a, l, "Normal")
```

```
    N2 = noise(a, l, "Uniform")
    energy_normal.append(np.sum(N1 ** 2) / l)
    energy_uniform.append(np.sum(N2 ** 2) / l)
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1), plt.plot(L, energy_normal, color='b'), plt.
   title('Normal Noise')
plt.subplot(1, 2, 2), plt.plot(L, energy_uniform, color='r'), plt
   .title('Uniform Noise')
plt.grid(True)
plt.show()
```



It seems to $E_N = \frac{1}{L}\sum_{i=1}^{L}(N[i])^2 \to \sigma^2$ as $L \to \infty$

(d) Cross-Correlation of Noise Vectors

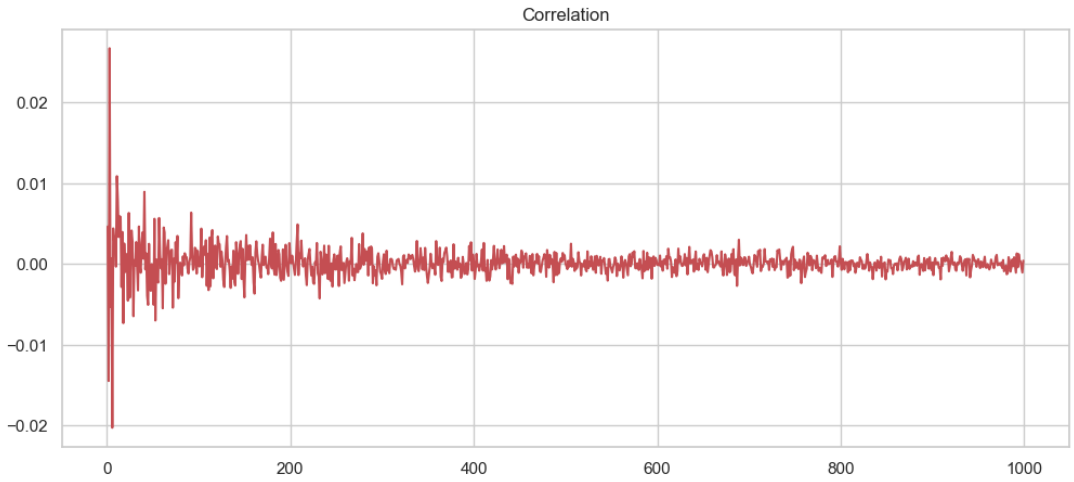
For the two noise vectors N_1 and N_2, define:

$$c_{12} = \frac{1}{L}\sum_{i=1}^{L} N_1[i]N_2[i]. \tag{7}$$

Plot c_{12} as a function of L for $L \in \{1, 2, \ldots, 1000\}$. Also, compute the limit of c_{12} as $L \to \infty$ and verify it through simulation. Explain what information c_{12} provides about the two vectors. Justify how c_{12} serves as a measure of similarity.

```
a = 0.2
L = np.arange(0, 1000, 1)
correlation = []
for l in L:
    N1 = noise(a, l, "Normal")
    N2 = noise(a, l, "Uniform")
    correlation.append(np.dot(N1, N2.T) / l)
plt.figure(figsize=(12, 5))
plt.plot(L, correlation, color='r'), plt.title('Correlation')
plt.grid(True)
```

```
plt.show()
```



Answer

$$c_{12} = \frac{1}{L}\sum_{i=1}^{L} N_1[i]N_2[i]. \tag{7}$$

This represents the empirical cross-correlation (or sample covariance, if the means of N_1 and N_2 are zero). Let us compute the limit of c_{12} as $L \to \infty$. If $N_1[i]$ and $N_2[i]$ are zero-mean independent noise processes, then:

$$\mathbb{E}[N_1[i]N_2[i]] = \mathbb{E}[N_1[i]]\mathbb{E}[N_2[i]] = 0.$$

By the Law of Large Numbers (LLN):

$$c_{12} = \frac{1}{L}\sum_{i=1}^{L} N_1[i]N_2[i] \xrightarrow{P} \mathbb{E}[N_1[i]N_2[i]] = 0 \quad \text{as} \quad L \to \infty.$$

Thus, if $N_1[i]$ and $N_2[i]$ are independent, the empirical cross-correlation converges to zero as $L \to \infty$.
If N_1 and N_2 are **correlated** with covariance σ_{12}, then:

$$\mathbb{E}[N_1[i]N_2[i]] = \sigma_{12}.$$

In this case, by the LLN:

$$c_{12} \xrightarrow{P} \sigma_{12}, \quad \text{as} \quad L \to \infty.$$

1. If $c_{12} = 0$, $\to$ uncorrelated. 2. If $c_{12} > 0$, N_1 and N_2 tend to increase and decrease together (positive correlation). 3. If $c_{12} < 0$, the two vectors tend to vary in opposite directions (negative correlation).
Thus, c_{12} serves as a **measure of similarity**, capturing how much the two noise vectors share common variations.

3.2.3 Sinusoidal Signal and Noise Addition

In this question, the original signal used is a simple sinusoidal signal defined as:

$$x(t) = \sin(2\pi f_0 t), \quad f_0 = 1 \text{ Hz}.$$

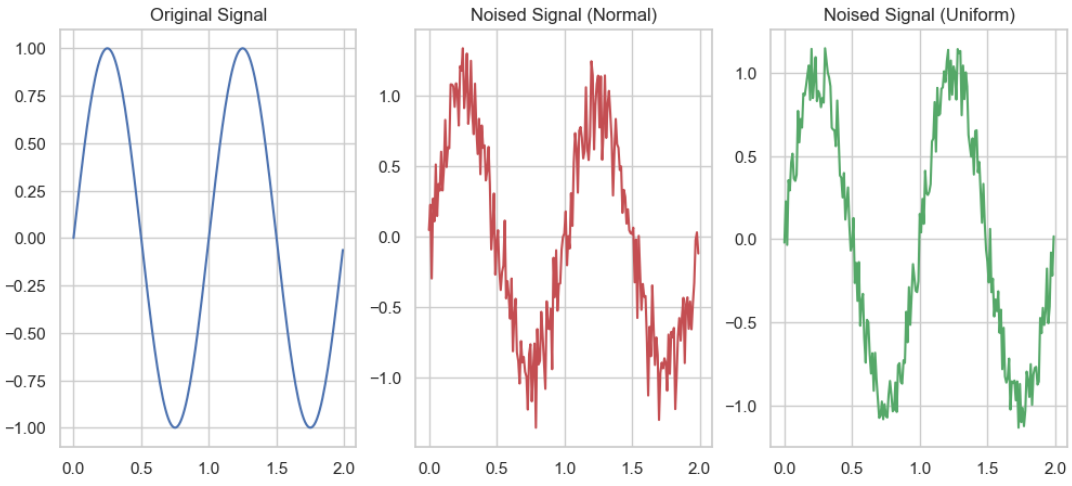
The time range of the signal is considered as $[0, 2]$. Generate the signal $x(t)$ with an appropriate sampling frequency such that the plotted graph in MATLAB closely resembles a continuous sinusoidal curve. Plot $x(t)$ and specify the sampling frequency used in the report.

Next, add the noise vectors N_1 and N_2 to the generated signal to produce two noisy signals x_{n1} and x_{n2}:

$$x_{n1}(t) = x(t) + N_1(t), \quad x_{n2}(t) = x(t) + N_2(t). \tag{8}$$

Plot both noisy signals in the report.

```
sampling_frequency = 100
frequency = 1
pi = np.pi
t = np.arange(0, 2, 1 / sampling_frequency)
signal = np.sin(2 * pi * frequency * t)
N1 = noise(0.2, len(t), "Normal")
N2 = noise(0.2, len(t), "Uniform")
signal_noised1 = signal + N1
signal_noised2 = signal + N2
plt.figure(figsize=(12, 5))
plt.subplot(1, 3, 1), plt.plot(t, signal, color='b'), plt.title('
    Original␣Signal')
plt.subplot(1, 3, 2), plt.plot(t, signal_noised1, color='r'), plt
    .title('Noised␣Signal␣(Normal)')
plt.subplot(1, 3, 3), plt.plot(t, signal_noised2, color='g'), plt
    .title('Noised␣Signal␣(Uniform)')
plt.grid(True)
plt.show()
```



3.2.4 Energy Function Definition

Write a function to compute the energy of a vector x, as defined by equation (6):

$$E = \texttt{energy}(x),$$

where the function takes a vector as input and outputs its energy.

```
def energyCalculator(signal):
    return np.sum(signal ** 2) / len(signal)
```

3.2.5 Energy Computation of Signals

Using the `energy` function written in the previous section, compute the energy of the following signals:

- The original signal x.
- The noise vectors N_1 and N_2.
- The noisy signals x_{n1} and x_{n2}.

In total, compute the energies of five signals. Report the computed energy values.

Verify whether the relation $E_{x_n} = E_x + E_N$ approximately holds. For any two signals x and y, determine if the statement $E_{x+y} = E_x + E_y$ is true or false. Prove or disprove the statement and validate it for this specific example.

Answer

- **1. Relation** $E_{x_n} = E_x + E_N$

  For the noisy signal $x_n = x + N$, the energy of x_n is:

  $$E_{x_n} = \sum_n |x_n[n]|^2 = \sum_n |x[n] + N[n]|^2$$

  Expanding the squared term:

  $$E_{x_n} = \sum_n \left(|x[n]|^2 + 2 \cdot x[n] \cdot N[n] + |N[n]|^2\right)$$

  Thus, we have:

  $$E_{x_n} = E_x + 2\sum_n x[n] \cdot N[n] + E_N$$

  Therefore, the relation $E_{x_n} = E_x + E_N$ does not hold exactly unless x and N are uncorrelated, i.e.,

  $$\sum_n x[n] \cdot N[n] = 0$$

- **2. Relation** $E_{x+y} = E_x + E_y$

  For two signals x and y, the energy of their sum $x + y$ is:

  $$E_{x+y} = \sum_n |x[n] + y[n]|^2$$

  Expanding the squared term:

  $$E_{x+y} = \sum_n \left(|x[n]|^2 + 2 \cdot x[n] \cdot y[n] + |y[n]|^2\right)$$

  Thus:

  $$E_{x+y} = E_x + E_y + 2\sum_n x[n] \cdot y[n]$$

  So, the relation $E_{x+y} = E_x + E_y$ does not hold exactly unless x and y are uncorrelated, i.e.,

  $$\sum_n x[n] \cdot y[n] = 0$$

```
y = np.cos(2 * pi * frequency * t)
signal_energy = energyCalculator(signal)
signal_noised_energy1 = energyCalculator(signal_noised1)
signal_noised_energy2 = energyCalculator(signal_noised2)
```

```
noise_energy1 = signal_noised_energy1 - signal_energy
noise_energy2 = signal_noised_energy2 - signal_energy
N1_energy = energyCalculator(N1)
N2_energy = energyCalculator(N2)
print("Energy of Original Signal: ", signal_energy)
print("Energy of Noised Signal (Normal): ", signal_noised_energy1
    )
print("Energy of Noised Signal (Uniform): ",
    signal_noised_energy2)
print("Energy of Normal Noise: ", N1_energy)
print("Energy of Noised Signal - Energy of Original Signal (
    Normal): ", noise_energy1)
print("Energy of Uniform Noise: ", N2_energy)
print("Energy of Noised Signal - Energy of Original Signal (
    Uniform): ", noise_energy2)
print("Energy of cos(2*pi*f*t): ", energyCalculator(y))
print("Energy of sin(2*pi*f*t): ", energyCalculator(signal))
print("Ex + Ey = ", energyCalculator(signal) + energyCalculator(y
    ))
```

```
Energy of Original Signal:  0.5000000000000001
Energy of Noised Signal (Normal):  0.5173038023362065
Energy of Noised Signal (Uniform):  0.5133046556821247
Energy of Normal Noise:  0.03933377492865725
Energy of Noised Signal - Energy of Original Signal (Normal):
    0.01730380233620643
Energy of Uniform Noise:  0.012032309028086838
Energy of Noised Signal - Energy of Original Signal (Uniform):
    0.013304655682124578
Energy of cos(2*pi*f*t):  0.5
Energy of sin(2*pi*f*t):  0.5000000000000001
Ex + Ey =  1.0
```

Listing 1: Result

3.2.6 Signal-to-Noise Ratio (SNR)

The ratio of the energy (power) of the original signal to the energy (or power) of the noise is called the Signal-to-Noise Ratio (SNR):

$$\text{SNR} = \frac{E_x}{E_N}.$$

Compute and report the SNR for the signals x_{n1} and x_{n2} generated in Section 3.2.3.

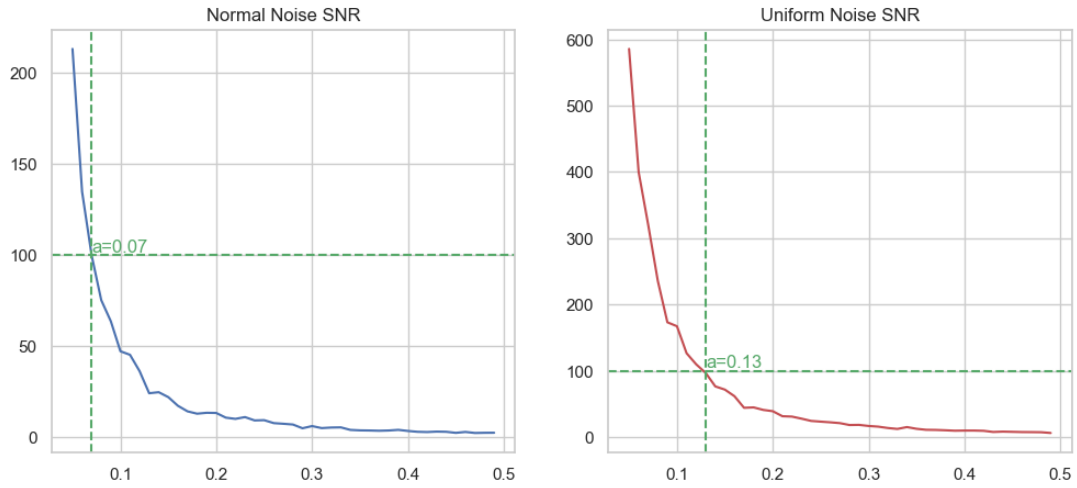
Subsequently, increase the value of a in the noise definitions until the smallest value of a is found such that there is no noticeable visual similarity between x and x_n. Report the corresponding value of a and the SNR for each type of noise separately.

```
print("The SNR of the signal with normal noise is: ",
    signal_energy / N1_energy)
```

```
print("The␣SNR␣of␣the␣signal␣with␣uniform␣noise␣is:␣",
   signal_energy / N2_energy)
a = np.arange(0.05, 0.5, 0.01)
SNR_normal = []
SNR_uniform = []
for i in a:
    N1 = noise(i, len(t), "Normal")
    N2 = noise(i, len(t), "Uniform")
    signal_noised = signal + N1
    signal_energy = energyCalculator(signal)
    noise_energy1 = energyCalculator(N1)
    noise_energy2 = energyCalculator(N2)
    SNR_normal.append(signal_energy / noise_energy1)
    SNR_uniform.append(signal_energy / noise_energy2)

plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1), plt.plot(a, SNR_normal, color='b'), plt.
   title('Normal␣Noise␣SNR'), plt.axhline(y=100, color='g',
   linestyle='--')
SNR_normal = np.array(SNR_normal)
index = np.argmin(np.abs(SNR_normal-100))
x_intercept2 = a[index]
plt.axvline(x=x_intercept2, color='g', linestyle='--')
plt.text(x_intercept2, 100, f'a={x_intercept2:.2f}', color='g',
   verticalalignment='bottom')
plt.subplot(1, 2, 2), plt.plot(a, SNR_uniform, color='r'), plt.
   title('Uniform␣Noise␣SNR'), plt.axhline(y=100, color='g',
   linestyle='--')
SNR_uniform = np.array(SNR_uniform)
index = np.argmin(np.abs(SNR_uniform-100))
x_intercept1 = a[index]
plt.axvline(x=x_intercept1, color='g', linestyle='--')
plt.text(x_intercept1, 100, f'a={x_intercept1:.2f}', color='g',
   verticalalignment='bottom')
plt.grid(True)
plt.show()
```

```
    The SNR of the signal with normal noise is:
       12.711721692283268
    The SNR of the signal with uniform noise is:
       41.55478377698392
```

Listing 2: Result


Normal Noise SNR
200
150
100
50
0
a=0.07
0.1
0.2
0.3
0.4
0.5
Uniform Noise SNR
600
500
400
300
200
100
0
a=0.13
0.1
0.2
0.3
0.4
0.5

4 Magnetic Resonance Imaging (MRI): A Comprehensive Overview

4.1 Introduction

Magnetic Resonance Imaging (MRI) is a non-invasive diagnostic technique that produces detailed images of the internal structures of the body. Utilizing strong magnetic fields and radio waves, MRI scans help in diagnosing a variety of conditions by revealing differences between healthy and diseased tissue.

4.2 How MRI works

MRI operates on the principles of nuclear magnetic resonance. Here's a step-by-step explanation:

- **Magnetic Alignment**: The human body is composed largely of water molecules, which contain hydrogen nuclei (protons). When a person enters the MRI scanner, a powerful magnetic field aligns these protons.
- **Radiofrequency Pulses**: Short bursts of radio waves are directed at the aligned protons, knocking them out of their equilibrium state.
- **Signal Emission**: When the radiofrequency pulse is turned off, the protons realign with the magnetic field, releasing energy in the process.
- **Image Formation**: This released energy is detected by coils in the scanner. Computers process these signals to construct detailed images of the body's interior.

Here is a brief explanation of the advantages of MRI for brain and spine, this information gives you a good insight into the rest of the project.

4.3 Brain MRI

MRI is particularly valuable for neurological applications due to its superior contrast resolution, which allows for:

- **Detection of Tumors and Lesions** : Identifying brain tumors, cysts, and other anomalies.
- **Stroke Evaluation**: Differentiating between ischemic and hemorrhagic strokes.
- **Multiple Sclerosis (MS)**: Visualizing demyelinating plaques characteristic of MS.
- **Aneurysm and Vascular Malformations**: Assessing blood vessel integrity and abnormalities.
- **Infection and Inflammation**: Detecting conditions like encephalitis or abscesses.

4.4 Spine MRI

Spine MRI provides critical information about spinal anatomy and pathology:

- **Disc Disorders**: Diagnosing herniated or degenerated discs.
- **Spinal Cord Compression**: Identifying causes of spinal cord pressure, such as tumors or bone fragments.
- **Inflammation and Infections**: Detecting osteomyelitis or epidural abscesses.
- **Congenital Anomalies**: Assessing birth defects affecting the spine.
- **Trauma**: Evaluating injuries to vertebrae, discs, ligaments, and the spinal cord itself.

4.5 Questions

Explore the main sources of noise in this method and suggest some ways to deal with this problem.

Answer of Reasons

- **Thermal Noise (Johnson-Nyquist Noise)**
  - **Source**: Caused by the random thermal motion of charged particles (electrons) in the MRI system's hardware (such as the coils and electronics).
  - **Effect**: Appears as random, grainy noise across the image, often referred to as white noise or Gaussian noise.
  - **Factors**: Influenced by temperature and resistance, this noise is more prominent in high-frequency components of the signal.
- **RF (Radiofrequency) Noise**
  - **Source**: RF noise can come from the interaction between the MR signal and surrounding electronics, including the RF coils, transmitter, and receiver.
  - **Effect**: This noise introduces low-frequency fluctuations in the signal, causing distortions in the image quality, especially during high field strength imaging.
- **Motion Artifacts**
  - **Source**: Patient motion, breathing, or involuntary movements during the MRI scan.
  - **Effect**: Causes blurring and ghosting of the image. This is particularly problematic in dynamic imaging (e.g., during functional MRI or cardiac MRI).
  - **Factors**: Motion of tissues or organs during the scan can result in significant image distortions.
- **Noise from External Sources**
  - **Source**: Electromagnetic interference (EMI) from external sources such as power lines, nearby electronic equipment, or other medical devices.
  - **Effect**: This induces external electrical noise, resulting in fluctuating background signals.
  - **Factors**: Equipment malfunction or unshielded wiring can introduce low-frequency interference into the system.

Answer of Reduction Methods

- **Noise Reduction Filters (Spatial and Temporal Filtering)**
  - **Method**: Filtering algorithms like Gaussian filters or wavelet denoising can reduce random noise without significantly affecting image details.
  - **How it helps**: Useful for eliminating Gaussian noise and RF noise.
  - **Drawback**: If applied incorrectly, filters may blur edges or remove important high-frequency information.
- **Using Motion Correction Techniques**
  - **Method**: Advanced motion correction algorithms detect and compensate for patient motion, reducing the effect of motion artifacts.
  - **How it helps**: Corrects blurring and ghosting, improving the quality of dynamic imaging.

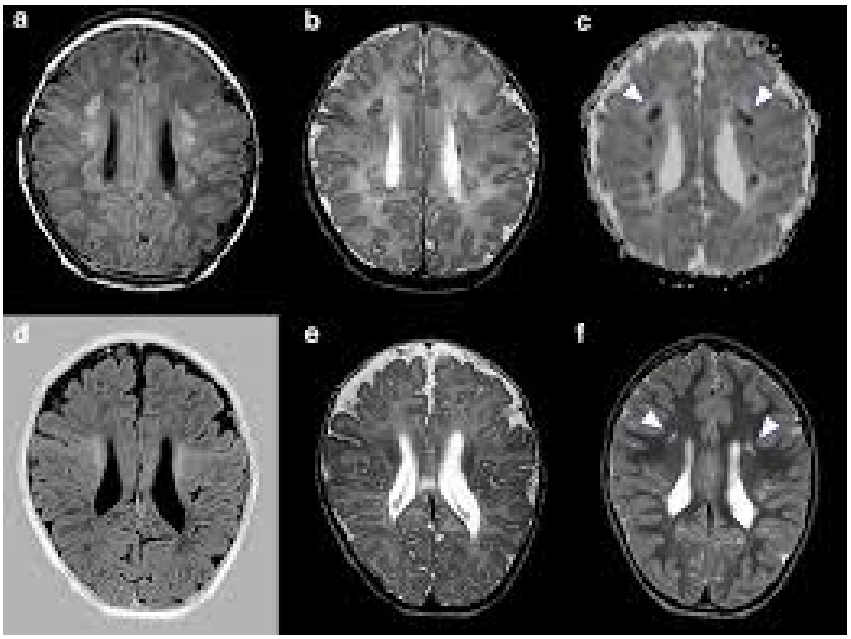


Figure 3: Brain MRI

5 Filters

In this part we learn about different filters, how they work and what's their impact on image. Then using these filters, you denoise two given noisy images and measure how effective each filter is.

Notice: You are not allowed to use built-in functions for this part.

5.1 Mean Filter

Investigate the Mean filter and implement a function to apply it to an input image. Explain about the effect of kernel size.

```
def meanFilter(image, kernel_size):
    h,w = image.shape
    pad_w = kernel_size // 2
    padded_image = np.pad(image, ((pad_w, pad_w), (pad_w, pad_w))
       , mode='constant', constant_values=0)
    filteredImage = np.zeros((h, w))
    for i in range(h):
        for j in range(w):
            kernel = padded_image[i:i+kernel_size, j:j+
               kernel_size]
            filteredImage[i, j] = np.mean(kernel)
    return filteredImage
```

5.2 Gaussian Filter

Explore about Gaussian filter and write a function that applies a Gaussian filter to an input image. What is the effect of variance and kernel size on the output image?

```
def gaussianFilter(image, variance, kernel_size):
    h,w = image.shape
    pad_w = kernel_size // 2
    padded_image = np.pad(image, ((pad_w, pad_w), (pad_w, pad_w))
       , mode='constant', constant_values=0)
    filteredImage = np.zeros((h, w))
    #Gaussian Kernel
    gaussian_kernel = np.zeros((kernel_size, kernel_size))
    for i in range(kernel_size):
        for j in range(kernel_size):
            x = i - pad_w
            y = j - pad_w
            gaussian_kernel[i, j] = np.exp(-(x**2 + y**2) / (2 *
               variance ** 2))
    gaussian_kernel /= np.sum(gaussian_kernel)
    for i in range(h):
        for j in range(w):
            kernel = padded_image[i:i + kernel_size, j:j +
               kernel_size]
```

```
            filteredImage[i, j] = np.sum(kernel * gaussian_kernel)
    return filteredImage
```

5.3 Bilateral Filter

Explore about Bilateral filter and write a function that applies a Bilateral filter to an input image.

```
def bilateralFilter(image, sigma_d, sigma_r, kernel_size):
    h, w = image.shape
    pad_w = kernel_size // 2
    padded_image = np.pad(image, ((pad_w, pad_w), (pad_w, pad_w)), mode='constant', constant_values=0)
    filteredImage = np.zeros((h, w))
    # Gaussian Kernel
    gaussian_kernel = np.zeros((kernel_size, kernel_size))
    for i in range(kernel_size):
        for j in range(kernel_size):
            x = i - pad_w
            y = j - pad_w
            gaussian_kernel[i, j] = np.exp(-(x**2 + y**2) / (2 * sigma_d ** 2))
    gaussian_kernel /= np.sum(gaussian_kernel)
    for i in range(h):
        for j in range(w):
            wsb = 0
            filtered_value = 0
            for ii in range(kernel_size):
                for jj in range(kernel_size):
                    spatial_weight = gaussian_kernel[ii, jj]
                    intensity_diff = padded_image[i + ii, j + jj] - padded_image[i + pad_w, j + pad_w]
                    intensity_weight = np.exp(-0.5 * (intensity_diff / sigma_r)**2)
                    weight = spatial_weight * intensity_weight
                    filtered_value += padded_image[i + ii, j + jj] * weight
                    wsb += weight
            if wsb != 0:
                filteredImage[i, j] = filtered_value / wsb
            else:
                filteredImage[i, j] = padded_image[i + pad_w, j + pad_w]

    return filteredImage
```

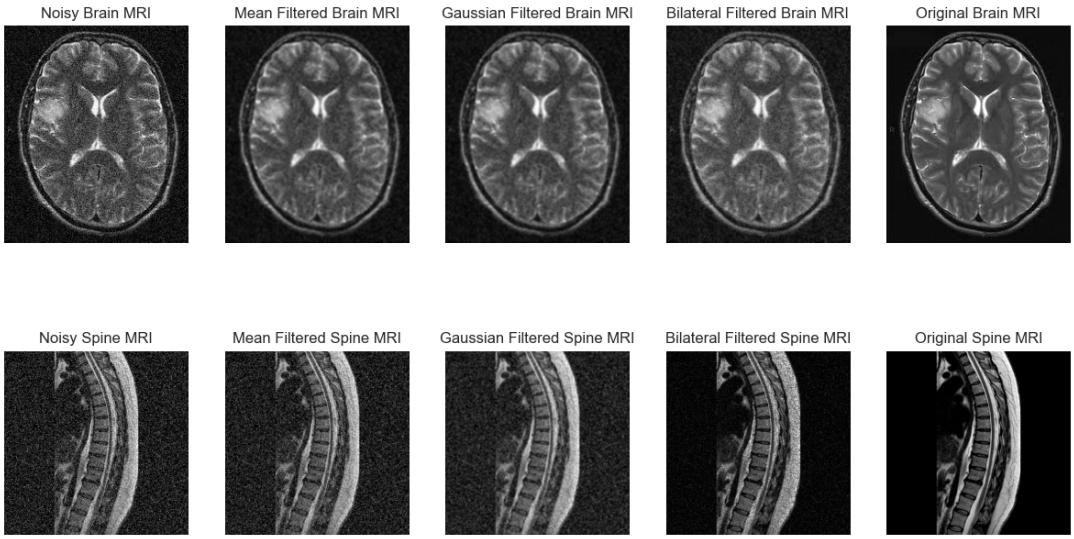
5.4 Evaluation

Evaluate the output of the three methods discussed in this section using SNR and PSNR criteria and present the results, compared with original_brain_MRI.jpg and original_spine_MRI.jpg in a table. Also, calculate these two criteria for the initial noisy image and declare which method performed better in noise reduction. Can you guess what kind of noise each picture has?

```
def calculate_psnr(noisy_img, filtered_img):
    mse = np.mean((noisy_img - filtered_img) ** 2)
    if mse == 0:
        return 100
    max_pixel = 255.0
    psnr = 10 * np.log10((max_pixel ** 2) / mse)
    return psnr

def calculate_snr(noisy_img, filtered_img):
    noise = noisy_img - filtered_img
    signal_power = energyCalculator(filtered_img)
    noise_power = energyCalculator(noise)
    snr = signal_power / noise_power
    return snr

brain_img = cv2.imread('pics/noisy_tumor.jpg', cv2.
   IMREAD_GRAYSCALE)
spine_img = cv2.imread('pics/noisy_spine.jpg', cv2.
   IMREAD_GRAYSCALE)
brain_img_original = cv2.imread('pics/tumor_original.jpg', cv2.
   IMREAD_GRAYSCALE)
spine_img_original = cv2.imread('pics/spine_MRI_image.jpg', cv2.
   IMREAD_GRAYSCALE)
# Apply Mean Filter
mean_filtered_brain = meanFilter(brain_img, 3)
mean_filtered_spine = meanFilter(spine_img, 3)
# Apply Gaussian Filter
gaussian_filtered_brain = gaussianFilter(brain_img, 1, 4)
gaussian_filtered_spine = gaussianFilter(spine_img, 2, 10)

# Apply Bilateral Filter
bilateral_filtered_brain = bilateralFilter(brain_img, 1, 100, 5)
bilateral_filtered_spine = bilateralFilter(spine_img, 1, 45, 5)

results = []
"""
# Noisy image comparison
results.append({
    'Image': 'Noisy Image',
    'Brain MRI (SNR)': calculate_snr(brain_img_original,
       brain_img),
    'Brain MRI (PSNR)': calculate_psnr(brain_img_original,
       brain_img),
```

```
    'Spine MRI (SNR)': calculate_snr(spine_img_original, spine_img),
    'Spine MRI (PSNR)': calculate_psnr(spine_img_original, spine_img),
})
"""
# Mean Filter
results.append({
    'Image': 'Mean Filter',
    'Brain MRI (SNR)': calculate_snr(brain_img, mean_filtered_brain),
    'Brain MRI (PSNR)': calculate_psnr(brain_img, mean_filtered_brain),
    'Spine MRI (SNR)': calculate_snr(spine_img, mean_filtered_spine),
    'Spine MRI (PSNR)': calculate_psnr(spine_img, mean_filtered_spine),
})

# Gaussian Filter
results.append({
    'Image': 'Gaussian Filter',
    'Brain MRI (SNR)': calculate_snr(brain_img, gaussian_filtered_brain),
    'Brain MRI (PSNR)': calculate_psnr(brain_img, gaussian_filtered_brain),
    'Spine MRI (SNR)': calculate_snr(spine_img, gaussian_filtered_spine),
    'Spine MRI (PSNR)': calculate_psnr(spine_img, gaussian_filtered_spine),
})

# Bilateral Filter
results.append({
    'Image': 'Bilateral Filter',
    'Brain MRI (SNR)': calculate_snr(brain_img, bilateral_filtered_brain),
    'Brain MRI (PSNR)': calculate_psnr(brain_img, bilateral_filtered_brain),
    'Spine MRI (SNR)': calculate_snr(spine_img, bilateral_filtered_spine),
    'Spine MRI (PSNR)': calculate_psnr(spine_img, bilateral_filtered_spine),
})

df = pd.DataFrame(results)
print(df)

fig, axes = plt.subplots(2, 5, figsize=(15, 8))
plt.grid(False)

```

```
75
76  axes[0, 0].imshow(brain_img, cmap='gray')
77  axes[0, 0].set_title('Noisy Brain MRI')
78  axes[0, 1].imshow(mean_filtered_brain, cmap='gray')
79  axes[0, 1].set_title('Mean Filtered Brain MRI')
80  axes[0, 2].imshow(gaussian_filtered_brain, cmap='gray')
81  axes[0, 2].set_title('Gaussian Filtered Brain MRI')
82  axes[0, 3].imshow(bilateral_filtered_brain, cmap='gray')
83  axes[0, 3].set_title('Bilateral Filtered Brain MRI')
84  axes[0, 4].imshow(brain_img_original, cmap='gray')
85  axes[0, 4].set_title('Original Brain MRI')
86
87
88  axes[1, 0].imshow(spine_img, cmap='gray')
89  axes[1, 0].set_title('Noisy Spine MRI')
90  axes[1, 1].imshow(mean_filtered_spine, cmap='gray')
91  axes[1, 1].set_title('Mean Filtered Spine MRI')
92  axes[1, 2].imshow(gaussian_filtered_spine, cmap='gray')
93  axes[1, 2].set_title('Gaussian Filtered Spine MRI')
94  axes[1, 3].imshow(bilateral_filtered_spine, cmap='gray')
95  axes[1, 3].set_title('Bilateral Filtered Spine MRI')
96  axes[1, 4].imshow(spine_img_original, cmap='gray')
97  axes[1, 4].set_title('Original Spine MRI')
98  plt.axis('off')
99  for ax in axes.flatten():
100     ax.axis('off')
101
102 plt.axis('off')
103
104 plt.show()
```



These are 2 MRI images that show salt and pepper noise. It seems that Gaussaian and Bilateral Filters (Gaussian for Tumor) and (Bilateral for Spine) are much better filters

for the detailed parts of images.

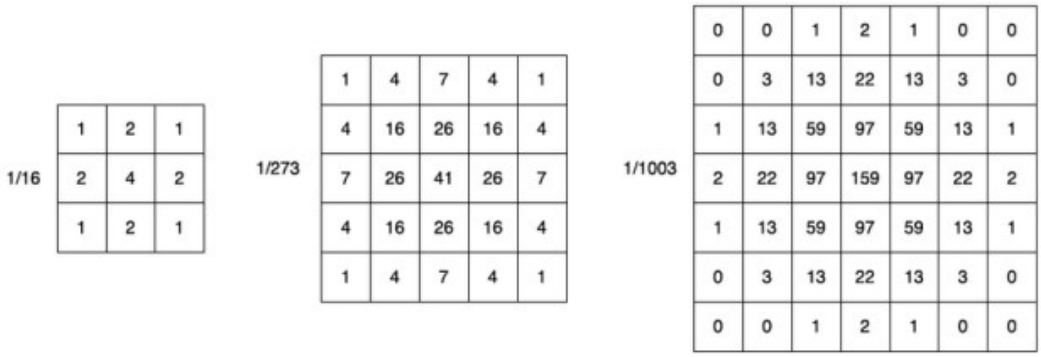


Figure 4: Discrete approximation of the Gaussian kernels 3x3, 5x5, 7x7

6 Adaptive Filter

6.1 Introduction

Adaptive filters are dynamic signal processing tools that adjust their parameters in real-time to accommodate varying input signal characteristics. Unlike fixed filters, which have static coefficients, adaptive filters modify their behavior based on the changing conditions of the data they process. This flexibility makes them particularly effective in environments where signal properties fluctuate, such as in biomedical image processing. In this field, adaptive filters are used for tasks like noise reduction, artifact removal, and image enhancement, ensuring that important features are preserved while unwanted disturbances are minimized. They employ algorithms, such as the Least Mean Squares (LMS) or Recursive Least Squares (RLS), to continuously optimize filter parameters based on feedback from the output. This ability to learn and adapt makes adaptive filters invaluable for improving image quality and accuracy in medical diagnostics and research applications.

6.2 Questions

1. What are finite impulse response (FIR) Adaptive Filters, and What are their specific applications in Biomedical Image Processing? Explore about Wiener filters.

Answer

Finite Impulse Response (FIR) Adaptive Filters:
FIR filters are digital filters that have a finite number of taps (or coefficients). They are widely used in signal processing for tasks like noise reduction, image enhancement, and feature extraction in biomedical image processing. FIR filters have the following characteristics:

- **Finite Duration Impulse Response:** The filter has a finite number of coefficients, meaning it responds to an impulse for only a finite amount of time.
- **Linear Phase Response:** FIR filters provide a linear phase response, meaning they do not distort the phase of the signal. This is important in biomedical imaging, where the integrity of the image is crucial.
- **Stability:** FIR filters are inherently stable because their coefficients are non-recursive.

Adaptive FIR Filters:
Adaptive FIR filters can dynamically change their coefficients in response to changes in the input signal, making them ideal for environments with non-stationary noise. These filters adjust the coefficients continuously based on the signal's characteristics, minimizing a predefined cost function.

Applications in Biomedical Image Processing:
Adaptive FIR filters are commonly used in:

- **Noise Reduction:** FIR filters help reduce noise (e.g., Gaussian noise, salt-and-pepper noise) in medical images such as MRI, CT, and X-ray scans.
- **Edge Detection:** FIR filters enhance or suppress certain features like edges in medical images, aiding in image segmentation and feature extraction (e.g., tumor detection).
- **Image Enhancement:** These filters improve contrast or sharpness in regions of interest in medical images, enhancing diagnostic capability.
- **Signal Processing in Time-Series:** In signals like ECG and EEG, adaptive FIR filters are used to filter out noise while preserving critical signal features.

Answer

Wiener Filter:
The Wiener filter is an adaptive filter used for noise reduction and signal restoration by minimizing the mean square error (MSE) between the estimated signal and the original. It is optimal when the signal and noise are statistically independent.
Wiener Filter Characteristics:

- **Optimal Filtering:** The Wiener filter is optimal in terms of minimizing error when the signal and noise characteristics are known.
- **Adaptive Nature:** It can be adapted based on local statistics of the image or signal in real-world applications.
- **Trade-off Between Noise Reduction and Feature Preservation:** The Wiener filter aims to balance noise removal and the preservation of important features like edges.

Wiener Filter in Biomedical Image Processing:
The Wiener filter is effective in the following applications:

- **Noise Reduction in Medical Imaging:** It is effective for reducing Gaussian noise in medical images (e.g., MRI, CT).
- **Image Restoration:** The Wiener filter restores images that have been degraded by motion blur or other artifacts.
- **Edge Preservation:** The filter helps preserve edges while removing noise, which is important in medical image analysis.

Mathematical Formulation of the Wiener Filter:
The Wiener filter is defined in the frequency domain by:

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

where:

- $H(u,v)$ is the Wiener filter in the frequency domain.
- $S(u,v)$ is the power spectrum of the signal.
- $N(u,v)$ is the power spectrum of the noise.

Steps to Apply Wiener Filter:

- **Fourier Transform:** Compute the Fourier Transform of the noisy image.
- **Compute Power Spectra:** Estimate the power spectra of the signal and noise.
- **Apply the Wiener Filter:** Calculate the Wiener filter using the power spectra.
- **Inverse Fourier Transform:** Apply the inverse Fourier transform to obtain the filtered image.

2. Explore the role of Stochastic Gradient Descent(SGD) in optimizing filter coefficients in real-time, its convergence properties, and its application in adaptive filter design.

How SGD works

Stochastic Gradient Descent (SGD) is a popular optimization algorithm used for training machine learning models, especially for large datasets and real-time applications. In adaptive filter design, SGD can be effectively used to optimize filter coefficients, especially when real-time performance is critical. This optimization occurs by iteratively updating the filter coefficients to minimize the error between the desired and actual filter output.

General Workflow of SGD in Adaptive Filters:

1. **Initialization:** Start with an initial guess for the filter coefficients (often zero or small random values).
2. **Error Calculation:** For each input sample, compute the error $e(t) = d(t) - y(t)$, where $e(t)$ is the error at time t, $d(t)$ is the desired signal, and $y(t)$ is the actual output.
3. **Gradient Calculation:** Compute the gradient of the error with respect to the filter coefficients. This is typically done by taking the derivative of a loss function (e.g., Mean Squared Error, MSE).
4. **Coefficient Update:** Update the filter coefficients using the gradient of the error:

$$\mathbf{w}(t+1) = \mathbf{w}(t) - \eta \nabla J(\mathbf{w}(t))$$

where $\mathbf{w}(t)$ are the filter coefficients at time t, η is the learning rate, and $\nabla J(\mathbf{w}(t))$ is the gradient of the cost function at time t. 5. **Iteration:** Repeat this process for each new sample until the error converges to a minimum.

Key Points about SGD in Adaptive Filter Design:

- **Real-time Performance:** SGD processes one sample at a time, enabling real-time adaptation of filter coefficients. This is ideal for applications such as speech enhancement, denoising, and image processing.
- **Memory Efficiency:** Since SGD uses one sample at a time for updates, it is memory-efficient, crucial for systems with limited computational resources.
- **Flexibility:** SGD can be applied to different types of filters (e.g., FIR, IIR) and used in a wide range of applications where real-time filter coefficient optimization is needed.

Convergence Properties

Convergence Properties of Stochastic Gradient Descent:
The convergence of SGD is an essential feature, but it comes with some challenges. Understanding these properties is key to ensuring the algorithm works effectively for optimizing filter coefficients.

- **Stochastic Nature:** Due to the stochastic updates, SGD does not guarantee convergence to the global minimum of the cost function but instead oscillates around it.
- **Learning Rate:** The choice of learning rate η is critical. A high learning rate might cause the algorithm to overshoot the minimum, while a low learning rate might lead to slow convergence.
- **Batch Processing:** In large datasets, SGD can be extended to mini-batch or batch gradient descent for smoother updates.
- **Convergence with Large Datasets:** SGD can be slow for non-convex problems, but it works efficiently for convex problems.
- **Convergence Rate:** SGD's convergence rate depends on factors such as the geometry of the loss function, the learning rate, and the dataset size. A decaying learning rate is often used for faster initial convergence and finer adjustments later.

Application of SGD

Application of SGD in Adaptive Filter Design:
1. **Noise Cancellation:** SGD is widely used for noise cancellation in systems such as hearing aids or speech enhancement, where filter coefficients are adjusted in real-time to reduce noise.
2. **Echo Cancellation:** In communication systems, adaptive filters are used for echo cancellation, with SGD optimizing the filter coefficients to remove echoes and improve signal quality.
3. **Signal Enhancement:** SGD helps in tasks such as image enhancement, denoising, and feature extraction by adjusting filter coefficients in real-time.
4. **System Identification:** In biomedical signal processing (e.g., ECG, EEG), SGD is used for system identification, where the adaptive filter models the system's behavior in real-time.
5. **Adaptive Equalization:** In communication systems, adaptive equalizers optimize filter coefficients in real-time using SGD to improve signal reception quality.
Advantages of Using SGD in Adaptive Filter Design:

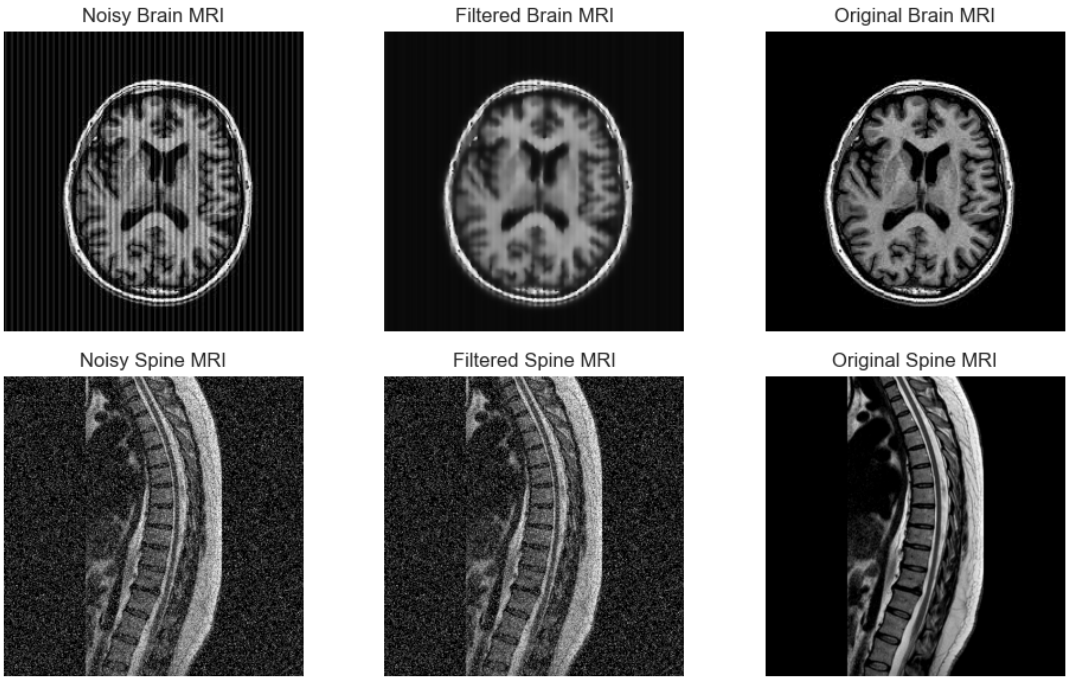
- **Real-time Adaptation:** SGD allows for real-time optimization of filter coefficients, making it ideal for dynamic environments.
- **Low Computational Complexity:** Since SGD processes one sample at a time, it is computationally efficient and suitable for systems with limited resources.
- **Scalability:** SGD can be easily scaled to handle large datasets or real-time data streams.
- **Flexibility:** It can be applied to various types of filters (FIR, IIR) and noise conditions.

3. Write a function that applies an Adaptive filter to an input image. Suggest a way to tune learning parameter. Apply this filter to original_brain_MRI.jpg and original_spine_MRI.jpg.

```
def adaptiveFilter(image, kernel_size, learning_rate, epochs,
    filter_method):
    h, w = image.shape
    pad_w = kernel_size // 2
    padded_image = np.pad(image, ((pad_w, pad_w), (pad_w, pad_w))
        , mode='reflect')
    padded_image = padded_image.astype(np.float32)
    epsilon = 1e-6
    for i in range(epochs):
        local_mean = cv2.GaussianBlur(padded_image, (kernel_size,
            kernel_size), 0.9)
        local_var = cv2.GaussianBlur(padded_image**2, (
            kernel_size, kernel_size), 0.9) - local_mean**2
        if (filter_method == 'Gaussian'):
            local_mean = cv2.GaussianBlur(padded_image, (
                kernel_size, kernel_size), 0.8)
```

```
            local_var = cv2.GaussianBlur(padded_image**2, (
                kernel_size, kernel_size), 0.8) - local_mean**2
        else:
#             local_mean = bilateralFilter(padded_image, 1, 45,
    kernel_size)
#             local_var = bilateralFilter(padded_image**2, 1, 45,
    kernel_size) - local_mean**2

            # Normalize the image to [0, 1] if necessary (for
                better processing with float images)
            local_mean = cv2.bilateralFilter(padded_image,
                kernel_size, 10, 10)
            local_var = cv2.bilateralFilter(padded_image**2,
                kernel_size, 10, 10) - local_mean**2
        noise_var = np.mean(local_var)
        wien_coef = local_var / (local_var + noise_var + epsilon)
        estimate = local_mean + wien_coef * (padded_image -
            local_mean)
        padded_image = padded_image + learning_rate * (estimate -
            padded_image)

    filteredImage = padded_image[pad_w:h+pad_w, pad_w:w+pad_w]
    filteredImage = np.clip(filteredImage, 0, 255)
    return np.uint8(np.clip(filteredImage, 0, 255))

brain_img = cv2.imread('pics/noisy_image_50.jpg', cv2.
    IMREAD_GRAYSCALE)
brain_img_original = cv2.imread('pics/original_2.jpg', cv2.
    IMREAD_GRAYSCALE)
filtered_brain_image = adaptiveFilter(brain_img, kernel_size=5,
    learning_rate=0.09, epochs=80, filter_method='Gaussian')
spine_img = cv2.imread('pics/noisy_spine.jpg', cv2.
    IMREAD_GRAYSCALE)
spine_img_original = cv2.imread('pics/spine_MRI_image.jpg', cv2.
    IMREAD_GRAYSCALE)
filtered_spine_image = adaptiveFilter(spine_img, kernel_size=3,
    learning_rate=0.01, epochs=80, filter_method='Bilateral')


plt.figure(figsize=(10, 6))

plt.subplot(2, 3, 1)
plt.imshow(brain_img, cmap='gray')
plt.title('Noisy␣Brain␣MRI')
plt.axis('off')

plt.subplot(2, 3, 2)
plt.imshow(filtered_brain_image, cmap='gray')
plt.title('Filtered␣Brain␣MRI')
plt.axis('off')

```

```
plt.subplot(2, 3, 3)
plt.imshow(brain_img_original, cmap='gray')
plt.title('Original Brain MRI')
plt.axis('off')

plt.subplot(2, 3, 4)
plt.imshow(spine_img, cmap='gray')
plt.title('Noisy Spine MRI')
plt.axis('off')

plt.subplot(2, 3, 5)
plt.imshow(filtered_spine_image, cmap='gray')
plt.title('Filtered Spine MRI')
plt.axis('off')

plt.subplot(2, 3, 6)
plt.imshow(spine_img_original, cmap='gray')
plt.title('Original Spine MRI')
plt.axis('off')

plt.tight_layout()
plt.show()
```



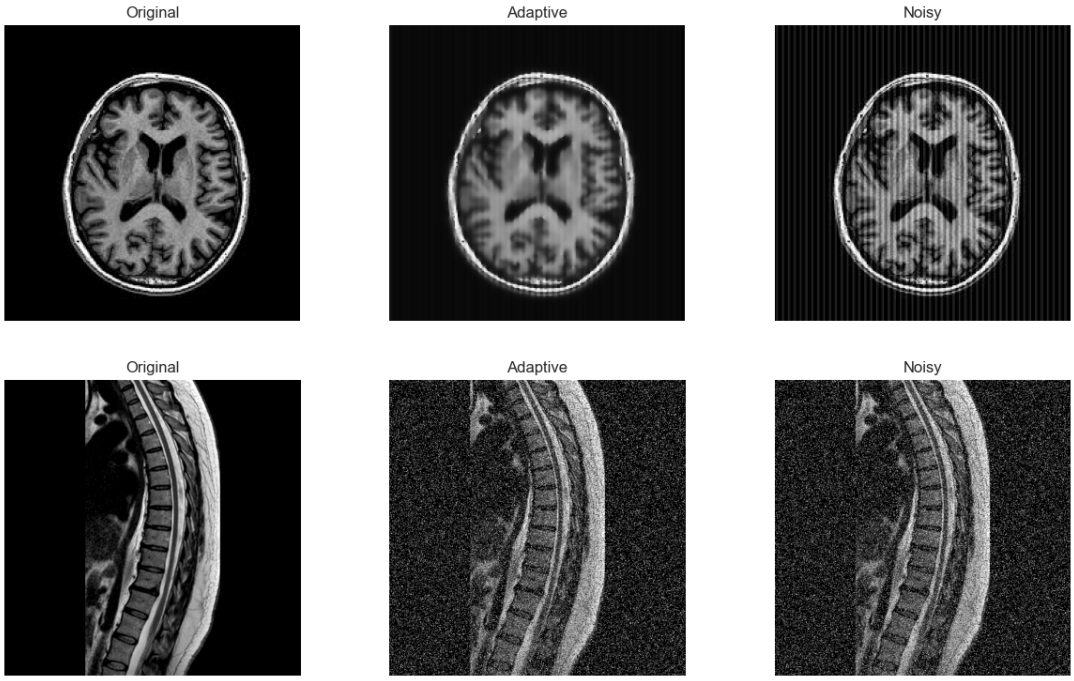
4. Evaluate the output of this method using SNR and PSNR criteria and present the results. Which of these four filters did better for each image? Can you explain why?

```
results = []
results.append({
    'Image': 'Adaptive Filter',
```

```
    'Brain MRI (SNR)': calculate_snr(brain_img, filtered_brain_image),
    'Brain MRI (PSNR)': calculate_psnr(brain_img, filtered_brain_image),
})
results.append({
    'Image': 'Adaptive Filter',
    'Spine MRI (SNR)': calculate_snr(spine_img, filtered_spine_image),
    'Spine MRI (PSNR)': calculate_psnr(spine_img, filtered_spine_image),
})
df = pd.DataFrame(results)
print(df)

fig, axes = plt.subplots(2, 4, figsize=(15, 8))
plt.grid(False)


axes[0,0].imshow(brain_img_original, cmap='gray')
axes[0,0].set_title('Original')
axes[0,1].imshow(filtered_brain_image, cmap='gray')
axes[0,1].set_title('Adaptive')
axes[0,2].imshow(brain_img, cmap='gray')
axes[0,2].set_title('Noisy')
axes[1,0].imshow(spine_img_original, cmap='gray')
axes[1,0].set_title('Original')
axes[1,1].imshow(filtered_spine_image, cmap='gray')
axes[1,1].set_title('Adaptive')
axes[1,2].imshow(spine_img, cmap='gray')
axes[1,2].set_title('Noisy')

for ax in axes.flatten():
    ax.axis('off')

plt.axis('off')

plt.show()
```

```
Image Brain MRI (SNR)  Brain MRI (PSNR)  Spine MRI (SNR)
              1.29607          27.92259        36.266215
   Spine MRI (PSNR)
         48.852381
```

Listing 3: Result



Answer

1. Higher SNR and PSNR
values indicate better filter performance, as they suggest more effective noise reduction and better preservation of the image details.

2. The Adaptive filter
is generally expected to outperform the other filters because it is designed to adapt to the image content, making it more effective in preserving the original structure while reducing noise.

3. Bilateral filters
are effective at noise reduction while maintaining edges, but they may still blur fine details, especially in regions with complex texture.

4. Gaussian filters
are efficient at smoothing the image but tend to blur edges, which might result in a slight loss of fine details in noisy regions.

5. Mean filters
are simple to implement but generally perform the worst. They tend to blur the image more than the other filters, especially when the image contains fine structures or edges, leading to significant loss of detail.

Based on these observations, we expected that the **Adaptive filter** provides the best performance in terms of both SNR and PSNR, followed by the **Bilateral filter**, which strikes a balance between noise reduction and edge preservation. The **Gaussian filter** will typically provide a smoother output but with potential blurring of finer details, while the **Mean filter** will generally produce the least effective results due to its simplicity and tendency to blur the image excessively.

Conclusion: The Adaptive filter is likely the most effective filter for both reducing noise and preserving important image details. The Bilateral filter provides a good compromise, while Gaussian and Mean filters offer simpler alternatives at the cost of performance.

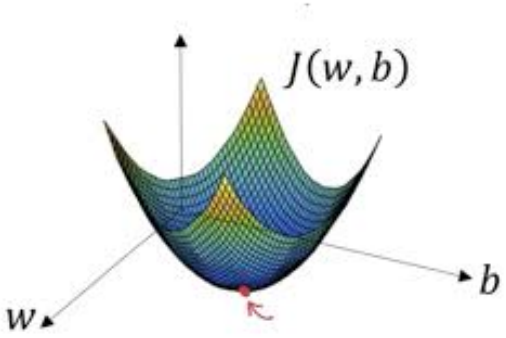


Figure 5: Gradient Descent Algorithm

7 References

- Jyothi, Sudagani & Perumal, Muthu. (2024). Performance Comparison of Different Digital Image Filters Used for Biomedical Signals. 10.1007/978-981-97-5786-2_36.
- Tomasi, C., & Manduchi, R. (1998). Bilateral filtering for gray and color images. Proceedings of the 1998 IEEE International Conference on Computer Vision, 839-846
- Gonzalez, R. C., & Woods, R. E. (2017). Digital Image Processing. Pearson
- Westin, C.-F., Knutsson, H., & Kikinis, R. (2000). Adaptive Image Filtering. In I. Bankman (Ed.), Handbook of Medical Imaging. Academic Press
- Gaussian filter (Wikipedia)
- Bilateral filter (Wikipedia)
- Salt-and-pepper noise (Wikipedia)
- Adaptive filter (Wikipedia)
- Adaptive filter (YouTube)