Assignment 8

Mahdi Tabatabaei - Heliya Shakeri

Biomedical Signal and Image Processing Lab

Nastaran Nouri

December 14, 2024

Contents

Theory: Gradient Descent for Deconvolution

To address the problem of deconvolution, we consider the iterative update formula:

$$f_{k+1} = f_k + \beta D^T(g - Df_k), \quad f_0 = 0$$

where β is the learning rate, D represents the convolution operator (e.g., the PSF in the case of Gaussian blur), and g is the observed blurred image. This formula iteratively minimizes the cost function $\|g - Df\|^2$, which measures the difference between the observed blurred image and the reconstructed image.

Soloution

To prove the iterative update formula, let us expand the cost function $\|g - Df\|^2$ as follows:

$$\|g - Df\|^2 = (g - Df)^T(g - Df) = g^T g - 2g^T Df + f^T D^T Df$$

Now, we compute the gradient of $\|g - Df\|^2$ with respect to f:

$$\nabla\|g - Df\|^2 = -2D^T(g - Df)$$

Using gradient descent, the update formula for f is given by:

$$f_{k+1} = f_k + \beta D^T(g - Df_k)$$

where $f_0 = 0$. Hence, the formula is derived.

Simulation

Question 1: Gaussian Noise and Filtering

Read the first slice of the image `t2.jpg` from the `S2_Q1_utils` directory, and add Gaussian noise with a variance of 15. Then, create a square kernel of size 4×4, centered in the Fourier domain, and apply it to filter the noisy image. Normalize the kernel by ensuring the sum of its elements equals 1. Finally, use the `imgaussfilt` function to filter the noisy image with a Gaussian kernel and compare the results.

```
% load image
img = imread(data_path+"/S2_Q1_utils/t2.jpg");
figure
imshow(img);
title('S2-Q1-utils/t2.jpg')
saveas(gcf,results_path + "fig.q1.0.png");

% first slice of the image
img_d = img(:,:,1);

% add gaussian noise to the image
varr = 0.015; m = 0;
img_d_noisy = imnoise(img_d,'gaussian',m,varr);

% binary kernel
[X,Y] = ndgrid(-size(img,1)/2:size(img,1)/2-1,-size(img,1)/2:size(img,1)
-1);
k_size = 4;
kernel = zeros(size(img,1), size(img,2));
kernel(abs(X) < k_size & abs(Y) < k_size) = 1;
imshow(kernel)
title("Binary kernel, k size = " + k_size)
saveas(gcf,results_path + "fig.q1.00.png");

% normalize the kernel
kernel = kernel/sum(kernel(:));

% lets go to the fourier domain
kernel_fft = fftshift(fft2(kernel));
img_d_noisy_fft = fftshift(fft2(img_d_noisy));
G = kernel_fft.*img_d_noisy_fft;

% inverse fft
img_filtered = fftshift(abs(ifft2(G)));
img_filtered = img_filtered./max(img_filtered(:));

figure('units','normalized','outerposition',[0 0 1 1])
subplot(2,2,1)
imshow(img_d)
title('Original Image - first slice')

subplot(2,2,2)
imshow(img_d_noisy)
title('Image with added gaussian noise')

subplot(2,2,3)
imshow(img_filtered)
title('Filtered using rect kernel')

```

```
    % use imgaussfilt
    img_filtered_2 = imgaussfilt(img_d,1);
    subplot(2,2,4)
    imshow(img_filtered_2)
    title('Filtered using gaussian kernel on the original image')
    saveas(gcf,results_path + "fig.q1.1.png");
```

Source Code 1: Simulation of Gaussian Noise Addition and Filtering

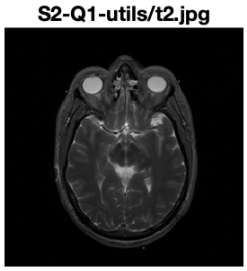

Figure 1: Original Image

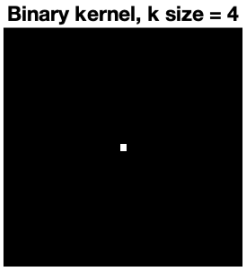

Figure 2: Binary Kernel (4×4)

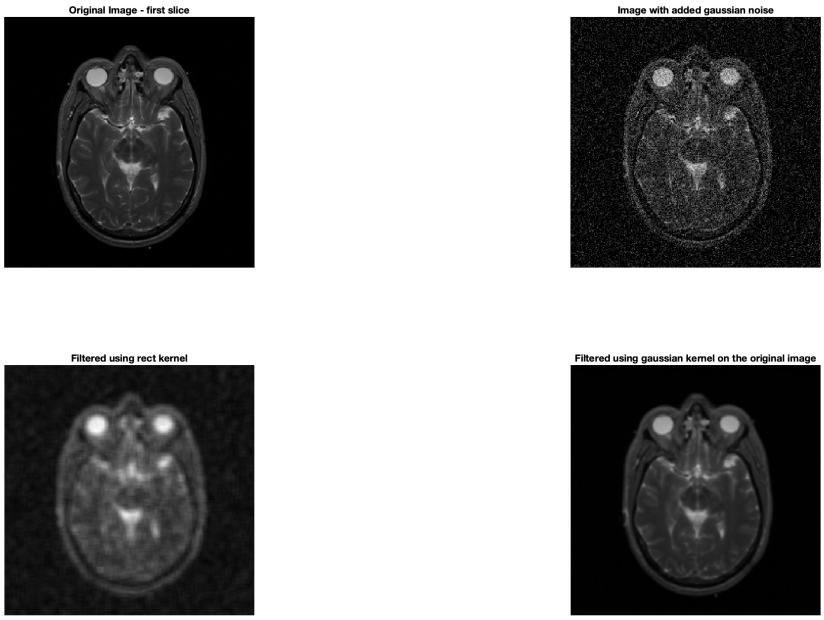

Figure 3: Filtering Results Comparison

Soloution

After comparing the results of the two filtering methods, it is observed that the square kernel smooths the image by averaging nearby pixels, effectively removing salt-and-pepper noise but also leading to a slight blurring of edges. The Gaussian filter, on the other hand, provides smoother edge preservation and better noise reduction in general while minimizing high-frequency components more effectively.

Question 2: Image Blurring and Reconstruction

Read the first slice of the image `t2.jpg` from the `S2_Q2_utils` directory. Blur the image significantly by applying a Gaussian filter (h) with a large variance, resulting in a blurred image. Then, reconstruct the original image using the formula $F = \frac{G}{H}$. Compare the reconstruction with the original and analyze the results.

Add Gaussian noise with a variance of 0.001 to the blurred image, apply the convolution filter, and attempt reconstruction again. Analyze and compare the outcomes.

```
% Load the image
img = imread(data_path+"/S2_Q2_utils/t2.jpg");
f = img(:,:,1);
figure;
imshow(f);
title('Original Image');
saveas(gcf, "results/fig.q2.0.png");

% Define Gaussian filter
f = double(image);
h = Gaussian(1, [256 256]);
g=conv2(f,h,'same');

% Fourier Transform
G_2=fftshift(fft2(ifftshift((g))));
H=fftshift(fft2(ifftshift((h))));
F=G_2./H;
f_recon=fftshift(ifft2(ifftshift(F)));

% Display results
figure('units', 'normalized', 'outerposition', [0 0 1 1]);
subplot(1, 3, 1);
imshow(f/max(f,[],'all'));
title('Original Image');
subplot(1, 3, 2);
imshow(g/max(g,[],'all'));
title('Blurred Image');
subplot(1, 3, 3);
imshow(f_recon/max(f_recon,[],'all'));
title('Reconstructed Image');
saveas(gcf, "results/fig.q2.1.png");

% Add Gaussian noise to blurred image
varr = 0.001;
g_noisy = g + randn(size(g)) * sqrt(varr);
g_noisy = g_noisy / max(g_noisy(:));
G_noisy = fftshift(fft2(ifftshift(g_noisy)));

% Inverse filtering for noisy image reconstruction
F_noisy = G_noisy ./ H;
```

```
    f_recon_noisy = abs(ifftshift(ifft2(ifftshift(F_noisy))));
    f_recon_noisy = f_recon_noisy / max(f_recon_noisy(:));

    % Display results
    figure('units', 'normalized', 'outerposition', [0 0 1 1]);
    subplot(1, 4, 1);
    imshow(f/max(f,[],'all'));
    title('Original Image');
    subplot(1, 4, 2);
    imshow(g/max(g,[],'all'));
    title('Blurred Image');
    subplot(1, 4, 3);
    imshow(g_noisy);
    title('Noisy Blurred Image');
    subplot(1, 4, 4);
    imshow(f_recon_noisy);
    title('Reconstructed Noisy Image');
    saveas(gcf, "results/fig.q2.2.png");
```

Source Code 2: Simulation of Blurring, Reconstruction, and Noise Addition

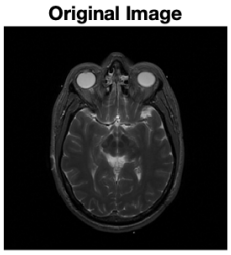

Figure 4: Original Image

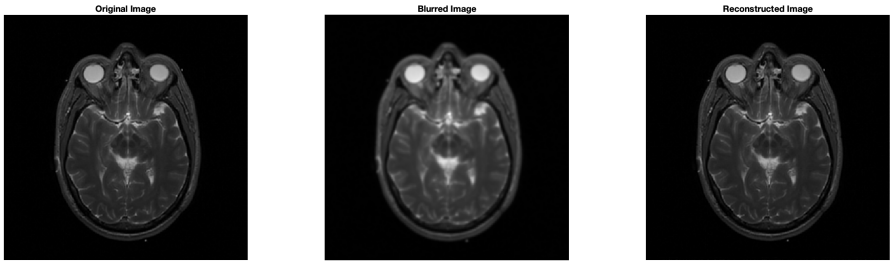

Figure 5: Blurred and Reconstructed Image Comparison

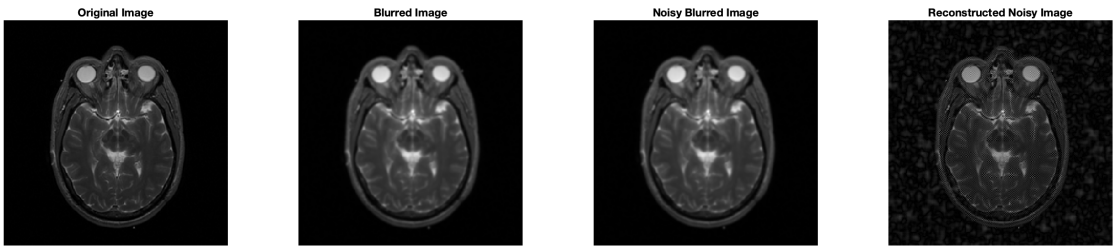

Figure 6: Filtering Results Comparison

Soloution

First, the blurred image is obtained by applying a Gaussian filter. The reconstruction using $F = \frac{G}{H}$ is successful, demonstrating that the Gaussian filter preserves sufficient information for recovery.
Next, Gaussian noise is added to the blurred image. Attempting reconstruction from the noisy blurred image shows significant degradation in quality, consistent with the ill-conditioned nature of the reconstruction problem in the presence of noise.

Question 3

In this question, we aim to compute the matrix $D \in \mathbb{R}^{N^2 \times N^2}$, knowing the blurring filter h, and use it to reconstruct the initial image from the previous question using the equation $f = Dtg$. To reduce computational overhead, first reduce the size of the initial image to 64×64. Note that each row of the matrix D corresponds to implementing the filtering effect for one pixel. Start by creating an all-zero image K with the same dimensions as the initial image, and place the filter h in the upper-left corner of this image. Then, for the pixel at location r, c, take the corresponding row in the matrix D by first shifting the image K by $c-1$ columns and $r-1$ rows. Explain why the entire rows of D correspond to a Toeplitz structure.

Construct the blurring matrix D and simulate the effect of the blurring filter h on the initial image (e.g., $h = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$ or a Gaussian filter). Then, add Gaussian noise with a standard deviation of 0.05 to Df to obtain the noisy blurred image g. Finally, reconstruct the initial image using the equation $f = D^{\dagger}g$ and analyze the results by displaying the initial image, the blurred image, and the reconstructed image, discussing the differences.

```
I = imread(data_path+"/S2_Q3_utils/t2.jpg");
I_d = I(:,:,1);
img = im2double(I_d);

% rescale image
scale = 1/4;
img = imresize(img, scale);

% K and h filter
K = zeros(size(img));
h = [0,1,0;...
    1,2,1;...
    0,1,0];
K(1:3,1:3) = h;

% create D by circulating K
[n, m] = size(img);
D = zeros(n*m, n*m);

count=1;
for c=1:64
    for r=1:64
        temp=circshift(K,[r-1 c-1]);
        D(count,:)=reshape(temp,1,64*64);
        count=count+1;
    end
end
```

```
    spy(D)

    img_vec = D*reshape(img,n*m,1);
    out_img = reshape(img_vec,n,m);
    img_vec_noisy = img_vec + 0.005*randn(length(img_vec),1);
    img_recon = pinv(D)*img_vec_noisy;

    figure('units','normalized','outerposition',[0 0 1 1])
    subplot(2,2,1);
    imshow(K)
    title('K image');

    subplot(2,2,2);
    imshow(out_img/max(out_img,[],'all'))
    title('Df');

    subplot(2,2,3);
    imshow(img);
    title('Orginial img')

    subplot(2,2,4);
    imshow(reshape(img_recon,m,n));
    title('Reconstructed img')
    saveas(gcf,results_path + "fig.q3.1.png");
```

Source Code 3: Simulation of Gaussian Noise Addition and Filtering

Soloution

The rows of the matrix D correspond to a Toeplitz structure because the operation of applying a filter (convolution) to an image involves shifting the filter over the image in a structured and repetitive manner. Specifically:

- In a Toeplitz matrix, each row is a shifted version of the previous row.
- In the case of D, each row represents the effect of the filter h on a specific pixel in the image. This means that the values in each row of D are derived by shifting the filter h across the image in a consistent pattern.
- Since the filter h is applied uniformly to all pixels in the image, the rows of D maintain a structured relationship, with each row being a shifted version of the same pattern (the filter kernel).

Thus, this repetitive shifting of the filter kernel h over the image grid inherently leads to a Toeplitz-like structure in the rows of D.

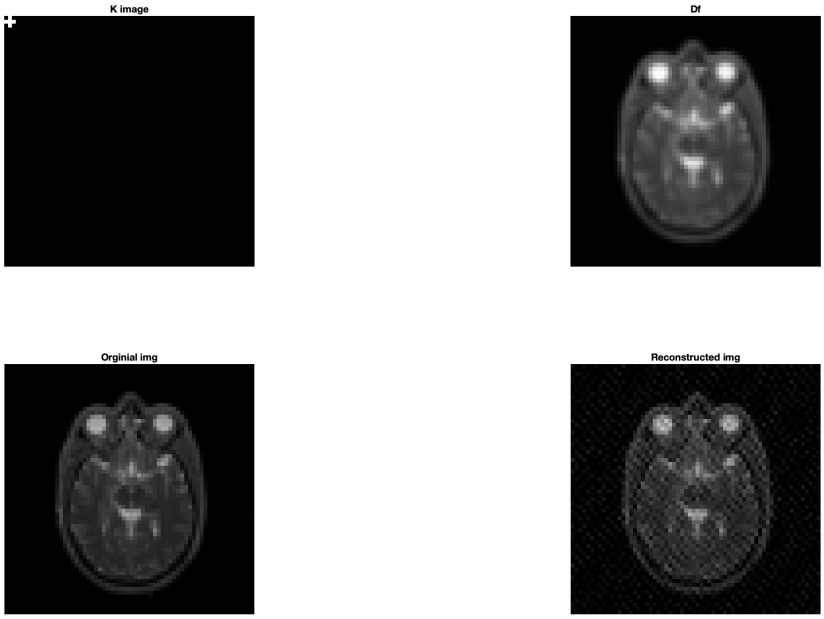

Figure 7: Filtering Results Comparison

Soloution

1. Top-left (K Image): The K image appears as an all-black image, indicating it was initialized as zeros and used in the process of constructing the matrix D. This step ensures that the filter h is correctly aligned with the image grid for constructing D.
2. Top-right (Df): This is the blurred image, created by applying the filter h through matrix D to the original image f.

- The image is blurred, indicating the smoothing or loss of fine details caused by the convolution operation of the filter h.
- The effect of the filter h is evident as the sharp edges and textures in the original image are smoothed out.
- Gaussian noise added during the process ($g = Df + \text{noise}$) introduces some level of distortion, though the overall structure of the image remains visible.

3. Bottom-left (Original Image): This is the unaltered original image before any blurring or filtering.

- The original image retains sharp details, such as the boundaries of the anatomical features in the MRI scan.
- These features are distinctly visible compared to the blurred image.

4. Bottom-right (Reconstructed Image, $f = D^{\dagger}g$): This is the reconstructed image, obtained by applying the pseudo-inverse of D ($D^{\dagger}$) to the noisy blurred image g.

- The reconstruction successfully restores much of the original image's structure.
- However, some artifacts (e.g., visible noise and slight checkerboard patterns) can be seen. These are likely due to:
  - Imperfections in the pseudo-inverse reconstruction.
  - Amplification of noise during the reconstruction process.
  - Limitations of reconstructing the fine details lost during the blurring and noise addition.

Key Differences:

- The **blurred image** (Df) shows significant loss of sharpness compared to the **original image**, which is expected due to the smoothing effect of the filter h.
- The **reconstructed image** recovers most of the image's original details but introduces noise and artifacts due to the noisy data and limitations of the pseudo-inverse operation.
- The overall shape and structure are preserved in the reconstruction, but finer textures and sharp edges may not be perfectly restored.

Question 4

Implement the Gradient Descent algorithm and execute the steps from the previous question, comparing the results. Additionally, analyze the convergence of different iterations of the algorithm through simulation. In your opinion, why does the Gradient Descent method often perform better than the method used in the previous question?

```
img_vec = D*reshape(img,n*m,1);
g = img_vec + 0.005*randn(length(img_vec),1);

% gradient descent
beta = 0.01;
f = reshape(img,n*m,1);
f_learn = zeros(size(f));
errors = [];
error_st = 100000000000;
counter = 1;
th = 0.00001;

while error_st >= th
    f_learn_upd = f_learn + beta*D'*(g - D*f_learn);
    errors(counter) = immse(f,f_learn_upd)/norm(f);
    error_st = errors(counter);
    f_learn = f_learn_upd;
    counter = counter + 1;
end

figure;
scatter(1:counter-1,errors,25,'black','filled','black');
```

```
    xlabel('Iteration'); ylabel('Recon MSE');
    title('Reconstruction err in each iteration');
    saveas(gcf,results_path + "fig.q4.0.png");

    figure('units','normalized','outerposition',[0 0 1 1])
    subplot(1,3,1);
    imshow(img)
    title('Orginal image');
    subplot(1,3,2);
    imshow(reshape(g,n,m)/max(reshape(g,n,m),[],'all'))
    title('Noisy image');
    subplot(1,3,3);
    f_learn = f_learn/max(f_learn,[],'all');
    imshow(reshape(f_learn,n,m))
    title('Reconstructed image');

    saveas(gcf,results_path + "fig.q4.1.png");
```

Source Code 4: Simulation of Gaussian Noise Addition and Filtering

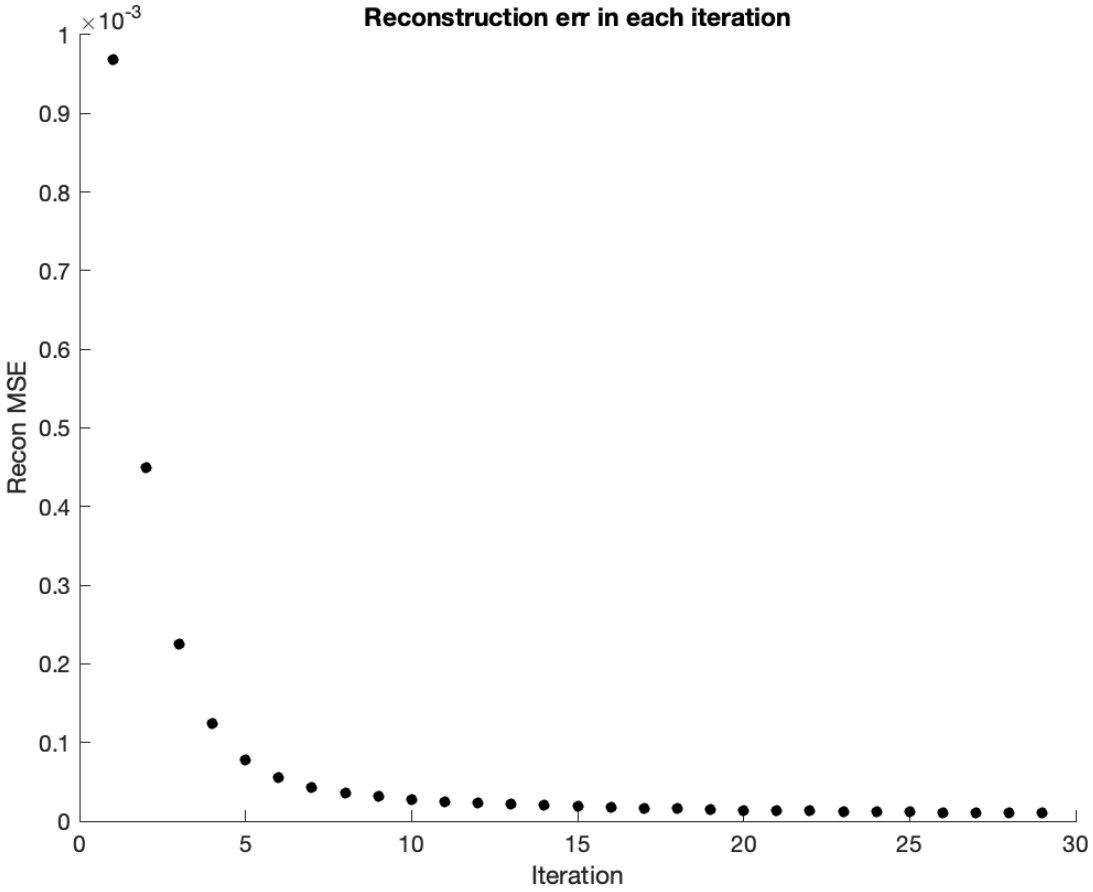

Figure 8: Reconstruction error in each iteration

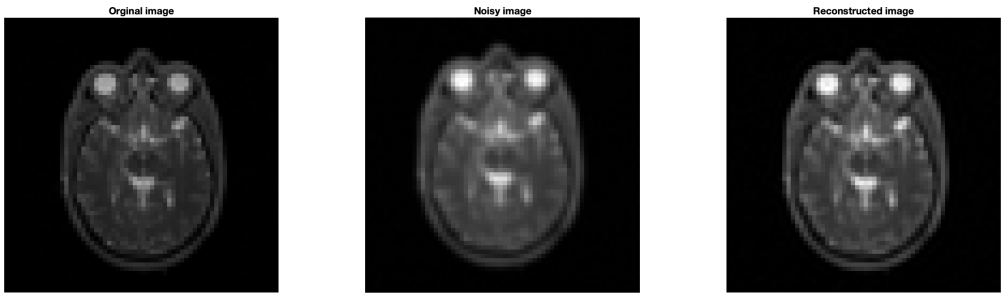

Figure 9: Filtering Results Comparison

Soloution

The convergence of the Gradient Descent algorithm was analyzed using the reconstruction error (Mean Squared Error, MSE) as a function of the iteration number, as shown in **Figure 8**. Key observations from the simulation are as follows:

- The reconstruction error decreases significantly in the initial iterations, demonstrating rapid convergence of the algorithm.
- After around 10-15 iterations, the error stabilizes and converges to a very low value, indicating that the algorithm has reached an optimal solution.
- The diminishing returns in later iterations illustrate the behavior of Gradient Descent, where the steps become smaller as the solution approaches the minimum of the error function.
- The final reconstruction achieves high accuracy, as evidenced by the low MSE and the visual quality of the reconstructed image (**Figure 9**).

This simulation confirms that the Gradient Descent algorithm is efficient and effective for solving the image reconstruction problem.

Soloution

The Gradient Descent method often performs better than the pseudo-inverse approach used in the previous question due to the following reasons:

- **Noise Sensitivity:** The pseudo-inverse method amplifies noise in the data because it involves matrix inversion, which can lead to instability and large numerical errors, especially when D is ill-conditioned. Gradient Descent avoids this problem by iteratively optimizing the solution.
- **Computational Efficiency:** Computing the pseudo-inverse of large matrices is computationally expensive and memory-intensive. In contrast, Gradient Descent requires only matrix-vector multiplications at each iteration, which is more efficient for high-dimensional problems.
- **Flexibility:** Gradient Descent can easily incorporate regularization techniques (e.g., L2 regularization) to handle noise and improve the stability of the solution. The pseudo-inverse method lacks this flexibility.
- **Scalability:** Gradient Descent scales better for large datasets and matrices, as it avoids the explicit computation of $D^{\dagger}$ and instead works with smaller iterative updates.

Question 5

Research Question: The methods mentioned for noise removal and interference reduction from medical images are among the most advanced techniques in this field. More advanced methods, such as the Anisotropic Diffusion Filtering method, have also been proposed. Research this method and its implementation. Additionally, execute the related code (S2_Q5_Anisotropic_Diffusion.m) and analyze the process and its execution.

Soloution

Anisotropic Diffusion Method: Anisotropic diffusion, also known as Perona-Malik diffusion, is a technique used in image processing for noise removal while preserving important structural details such as edges. It is based on solving a Partial Differential Equation (PDE) that smooths image intensity values over time while preventing smoothing across edges. The diffusion process is controlled by an edge-stopping function, which reduces diffusion near high gradients (edges) and enhances diffusion in flat, noisy regions. The general anisotropic diffusion equation is:

$$\frac{\partial f}{\partial t} = \nabla \cdot (c(\nabla f)\nabla f),$$

where:
- f is the image intensity,
- $c(\nabla f)$ is the edge-stopping function that depends on the gradient magnitude ∇f,
- $\nabla \cdot$ represents the divergence operator.

Two commonly used edge-stopping functions are:

$$c = \frac{1}{1 + \frac{|\nabla f|^2}{K^2}} \quad \text{(Perona-Malik exponential function)},$$

or

$$c = \exp\left(-\frac{|\nabla f|^2}{K^2}\right).$$

These functions ensure that diffusion slows down near edges (high gradient magnitude) and proceeds in flat regions (low gradient magnitude).

Implementation of the Method: The provided code implements anisotropic diffusion using the Perona-Malik model with several tunable parameters:

1. **Input Image:** The image is read and Gaussian noise is added for testing. The left half of the output displays the noisy input, while the right half shows the result of anisotropic diffusion at each timestep.

2. **Edge-Stopping Function:** The function c controls the diffusion process based on the gradient magnitude. The code uses the function:

$$c = \frac{1}{1 + \frac{\nabla f^2}{(K/2)^2}}.$$

Another edge-stopping function $c = \exp(-\nabla f^2/K^2)$ is provided as an alternative.

3. **PDE Solver:** The PDE is solved iteratively over time using a finite difference method. The code computes the gradient of the image f and updates the image at each timestep:

$$f = f + \lambda \cdot \delta_t \cdot \text{rhs},$$

where rhs is the right-hand side of the PDE, accounting for divergence and gradients.

4. **Sophisticated Diffusion Term:** For the more sophisticated method (`method=2`), the implementation includes an additional term for spatial variation of c, improving edge preservation.

Running Process: The diffusion process evolves over time, and the output is visualized at regular intervals. The following key steps occur during execution: - **Initialization:** The noisy input image is initialized, and parameters like K, δ_t, t_{end}, and λ are set. These parameters control the behavior of the diffusion process:
- K: Edge-stopping threshold.
- δ_t: Timestep size.
- t_{end}: Total simulation time.
- λ: Diffusion scaling factor.
- **Iterative Updates:** For each timestep, the gradients ∇f are computed, the divergence term is calculated, and the image is updated iteratively. The method prevents excessive smoothing near edges by dynamically adjusting the diffusion coefficient c.

Results and Observations: The output shows that anisotropic diffusion effectively reduces noise while preserving important edges:
- At early stages (t small), noise is reduced without blurring sharp edges.
- By $t = 30$, the right half of the image is denoised, with fine structural details intact.
- The diffusion process avoids oversmoothing, demonstrating the robustness of anisotropic diffusion.

So, Anisotropic diffusion is an advanced method for denoising images while preserving edges. The provided code effectively implements this method and demonstrates its ability to handle noisy images. Through the use of a tunable edge-stopping function, anisotropic diffusion ensures noise reduction without compromising critical image features, making it a powerful tool in medical image processing.