Laboratory of Biomedical Signal and Image Processing

REPORT OF LAB 8

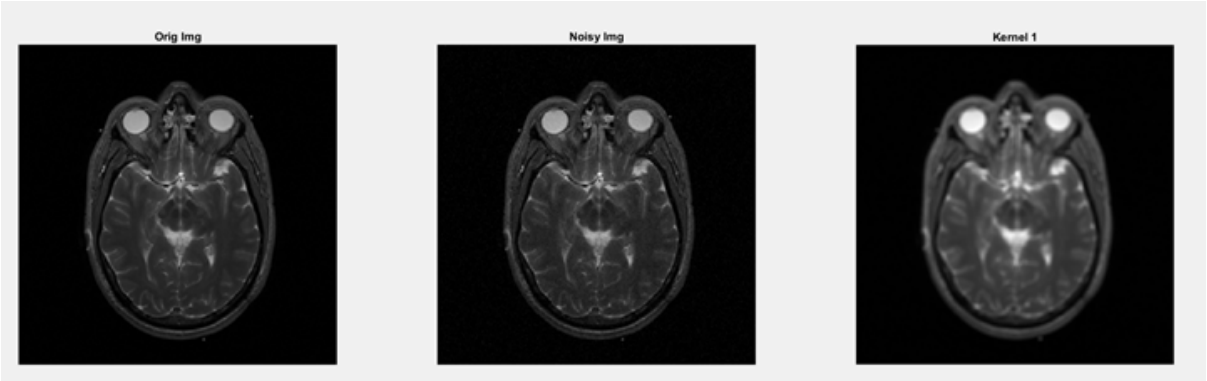
Sara Rezanejad | 99101643

Ali Khosravipour | 99101502

Mohamad Hosein Faramarzi | 99104095

Question 1)

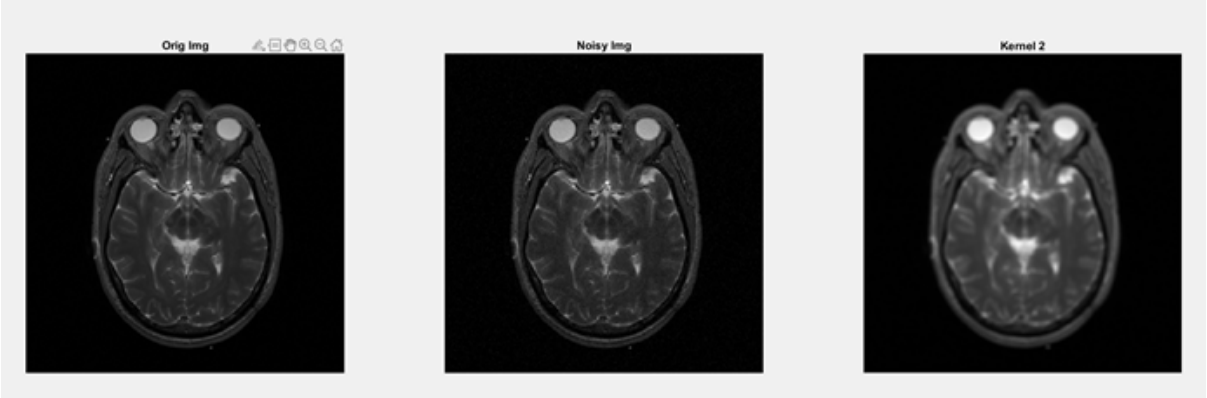
```
%% Ali Khosravipour 99101502 // MohamadHosein Faramarzi 99104095 // Sara Rezanejad 99101643
%% Q1
t2_img = imread('t2.jpg');
first_slice = t2_img(:,:,1);
var = 15;
noisy_first_slice = imnoise(first_slice, 'gaussian', 0, var / 255^2);
rows = 256;
cols = 256;
binary_img = zeros(rows, cols);
start_row = floor((rows - 4) / 2) + 1;
start_col = floor((cols - 4) / 2) + 1;
binary_img(start_row:start_row+3, start_col:start_col+3) = 1;
F_first_slice = fft2(double(first_slice));
F_binary_img = fft2(double(binary_img));
F_result = F_first_slice .* F_binary_img;
result_img = real(ifftshift(ifft2(F_result)));
figure;
subplot(1, 3, 1);
imshow(first_slice, []);
title('Orig Img');
subplot(1, 3, 2);
imshow(noisy_first_slice, []);
title('Noisy Img');
subplot(1, 3, 3);
imshow(result_img, []);
title('Kernel 1');
```



In this code, we processed an MRI image by introducing noise and applying a kernel in the frequency domain. First, we extracted the first slice of the MRI image and added Gaussian noise to simulate real-world imaging imperfections. Next, we created a small 4x4 binary kernel centered within a 256x256 zero matrix. We then applied the Fourier Transform (FFT) to both the image and the kernel, multiplied them in the frequency domain, and used the

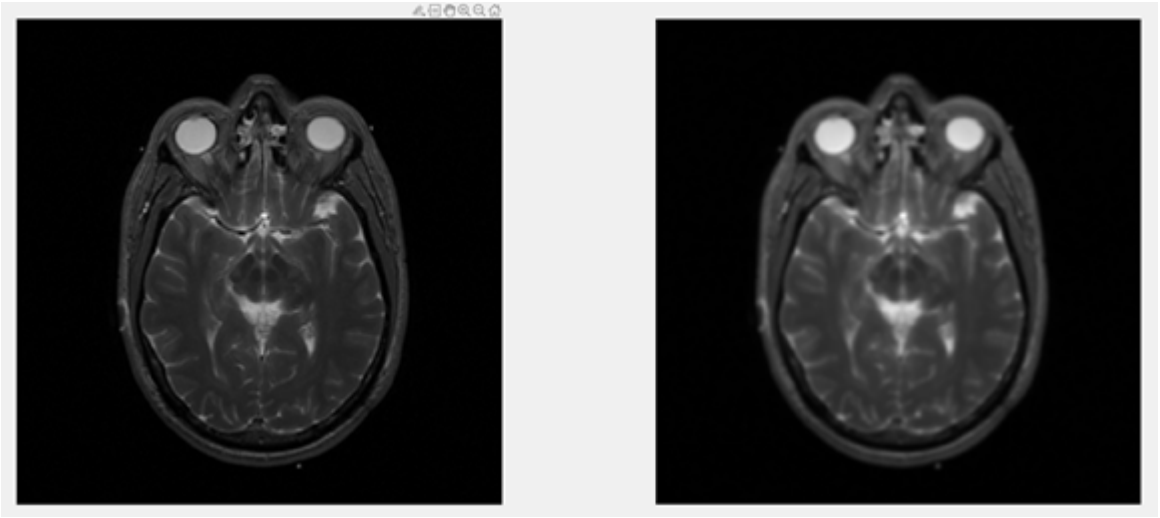
inverse Fourier Transform (IFFT) to bring the result back to the spatial domain. This operation effectively applied the kernel to the image, resembling a filtering process. Finally, we displayed the original image, the noisy version, and the filtered result, showing how the kernel influenced the image by enhancing certain features.

```
27      %%
28      start_row = floor((rows - 4) / 2) + 1;
29      start_col = floor((cols - 4) / 2) + 1;
30      binary_img2 = zeros(rows, cols);
31      binary_img2(start_row:start_row+3, start_col:start_col+3) = 1 / (4 * 4);
32      F_binary_img2 = fft2(double(binary_img2));
33      F_result2 = F_first_slice .* F_binary_img2;
34      result_img2 = real(ifftshift(ifft2(F_result2)));
35      figure;
36      subplot(1, 3, 1);
37      imshow(first_slice, []);
38      title('Orig Img');
39      subplot(1, 3, 2);
40      imshow(noisy_first_slice, []);
41      title('Noisy Img');
42      subplot(1, 3, 3);
43      imshow(result_img2, []);
44      title('Kernel 2');
```



In this code, we modified the binary kernel by normalizing its values and applied it to the MRI image in the frequency domain. We first centered a **4x4 kernel** with all elements set to 1/16 (1 divided by the total kernel size) within a 256×256 zero matrix. This normalization ensures that the kernel's effect is averaged over its region. We then transformed this new kernel into the frequency domain using the Fourier Transform (FFT), as we did earlier. The transformed kernel was multiplied element-wise with the frequency representation of the original MRI slice. Finally, we applied the inverse Fourier Transform (IFFT) to bring the result back to the spatial domain, where the kernel's effect could be visualized. The resulting image, labeled **Kernel 2**, shows a smoother and more averaged filtering effect compared to the first kernel. This happens because the normalized kernel acts like a **low-pass filter**, reducing high-frequency noise and smoothing the image while retaining its general structure. The three outputs—**original image**, **noisy image**, and **Kernel 2 result**—allow us to compare how the kernel processing mitigates noise and enhances the image.

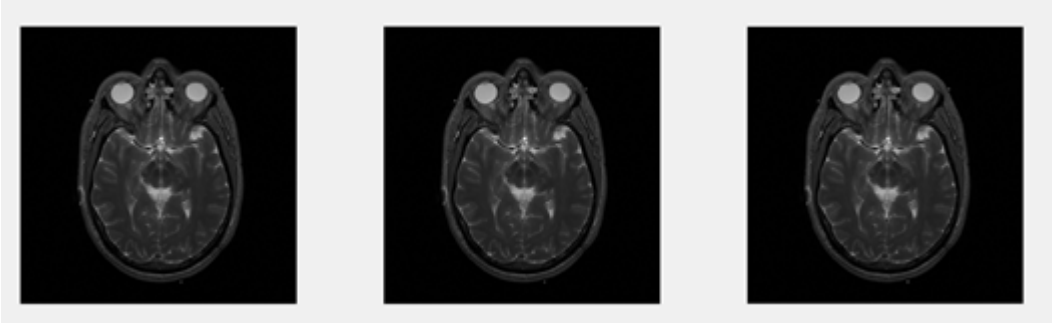
```
45  %%
46  gaussfilt_img = imgaussfilt(double(first_slice), sqrt(1));
47  figure;
48  subplot(1,2,1);
49  imshow(first_slice, []);
50  subplot(1,2,2);
51  imshow(gaussfilt_img, []);
```



In this part, we applied a Gaussian filter to the original MRI image to smooth it. The function imgaussfilt was used, where the standard deviation of the Gaussian kernel was set to one, ensuring a moderate smoothing effect. Gaussian filtering reduces high-frequency noise in the image while preserving its low-frequency components, resulting in a smoother and cleaner version. The output consists of two images displayed side by side: the original MRI slice on the left and the Gaussian-filtered image on the right. By comparing these two, we can observe that the Gaussian filter effectively smooths out fine noise and sharp variations while maintaining the overall structure of the image.

Question 2)

```
%% Q2
t2_img = imread('t2.jpg');
f = t2_img(:,:,1);
h = Gaussian(0.5,[256,256]);
g = conv2(double(f),h,'same');
G = fft2(g);
H = fft2(h);
F = G ./ H;
recon_f = abs(fftshift(ifft2(F)));
%
g_noised = imnoise(g,'gaussian',0,0.001);
G_noised = fft2(g_noised);
F2 = G_noised ./ H;
recon_f2 = abs(fftshift(ifft2(F)));
%
figure;
subplot(1,3,1);
imshow(f,[]);
subplot(1,3,2);
imshow(recon_f,[]);
subplot(1,3,3);
imshow(recon_f2,[]);
```



In this question, we implemented image reconstruction using Gaussian filtering and Fourier domain operations, starting with an MRI image slice. First, we created a Gaussian kernel h with a standard deviation of 0.5 and size 256×256. This kernel was convolved with the original image f, producing a blurred image g. The blurring simulates a loss of high-frequency details in the image.

To reconstruct the original image, we used the Fourier Transform. By transforming both g (blurred image) and h (Gaussian kernel) into the frequency domain, we computed the estimate of the original image F by dividing the transformed blurred image G by the transformed kernel HHH. Applying the inverse Fourier Transform, we recovered a version of the image recon_f, which appears sharper than the blurred one.

We then added Gaussian noise to the blurred image g to simulate realistic imperfections. The noisy image was similarly transformed into the frequency domain and used for reconstruction. The resulting image recon _f2 demonstrates the effect of noise on the reconstruction process. While the reconstructed image from the noise-free data looks clean and sharp, the one from the noisy data retains some noise, making it slightly less accurate.

The three images displayed are the original MRI slice, the reconstructed image from noise-free data, and the reconstructed image from the noisy version. This process illustrates how Gaussian filtering and noise affect image quality and how frequency-domain techniques can help recover images to a certain extent.

Proof of cost function:

$$J(f) = \|g - Df\|^2$$

$$\rightarrow J(f) = (g-Df)^T(g-Df)$$

$$\hookrightarrow J(f) = g^Tg - g^TDf - (Df)^Tg + (Df)^T(Df)$$

$$J(f) = g^Tg - 2f^TD^Tg + f^TD^TDf$$

$$\nabla_f J(f) = \frac{\partial}{\partial f}(g^Tg - 2f^TD^Tg + f^TD^TDf)$$

$$\hookrightarrow = -2D^Tg + 2D^TDf$$

$$= 2(D^TDf - D^Tg) = -2D^T(g-Df)$$

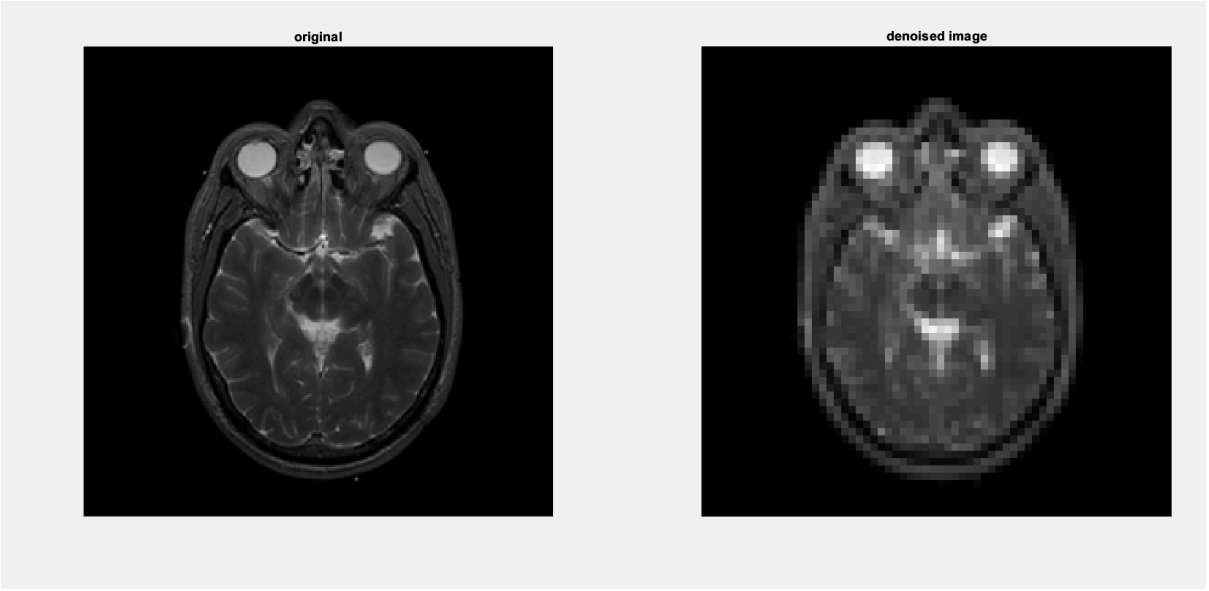
Gradient descent: $f_{k+1} = f_k - \alpha \nabla_f J(f_k)$

$$\Rightarrow f_{k+1} = f_k - \alpha(-2D^T(g - Df_k))$$

$\beta = 2\alpha \hookrightarrow$ $f_{k+1} = f_k + \beta D^T(g - Df_k)$

$f_c = 0$

Question 3)



Explanation of the Code:

Image Reading and Resizing:

- The code starts by reading an image from the specified file path and resizing it to 64x64 pixels.
- It then displays the resized image with a title.

2. Grayscale Conversion:
   - If the resized image is in RGB format (3 channels), it converts the image to grayscale.
3. Kernel Definition:
   - A 3x3 kernel (`h`) is defined, which is typically used for filtering (though it's referred to as Gaussian in a non-standard way here).
4. Matrix Construction:
   - A 4096x4096 matrix `D` is created, where each row is a shifted version of the kernel `K`, filling the matrix based on the kernel's position across the 64x64 image.
5. Image Reshaping:
   - The grayscale image is reshaped into a column vector (`image_1`) for matrix operations.
6. Applying Filter and Adding Noise:
   - The matrix `D` is multiplied by `image_1` to create a transformed image (`g`).
   - Gaussian noise is added to the transformed image to simulate noisy data.
7. Denoising:
   - The pseudo-inverse of matrix `D` is calculated and used to denoise the noisy image (`NoisyImage`).
   - The denoised image is reshaped back to a 64x64 format.

8. Display Results:
    - Finally, the original and the denoised images are displayed side by side.
9. Display Size Information:
    - The sizes of the original and resized images are printed for reference.

Differences in the result:

Original Image:

Details: The original image retains a significant amount of detail and texture. It is likely to provide clear contrasts between different structures, allowing for better interpretation.

Noise: Any noise present may appear as random grain or distortion, which affects image quality but does not necessarily obscure important features.

Removed Image:

Smoothing: The removed image appears smoother and less grainy compared to the original image. This indicates that the noise has been effectively removed, potentially improving visibility of underlying structures.

Loss of Detail: However, there is a significant loss of fine detail. Edges may appear softer and some fine features in the original image may be less distinct or disappear altogether.

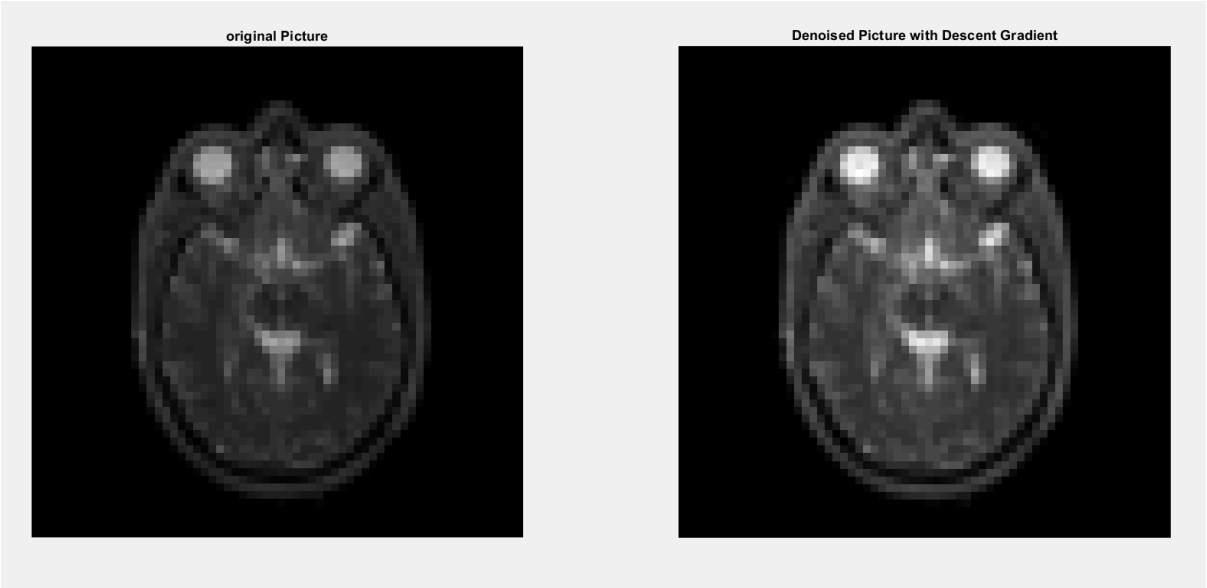
Contrast and Brightness: The removed image may also show altered contrast or brightness levels, which affects the overall perception of depth and detail.

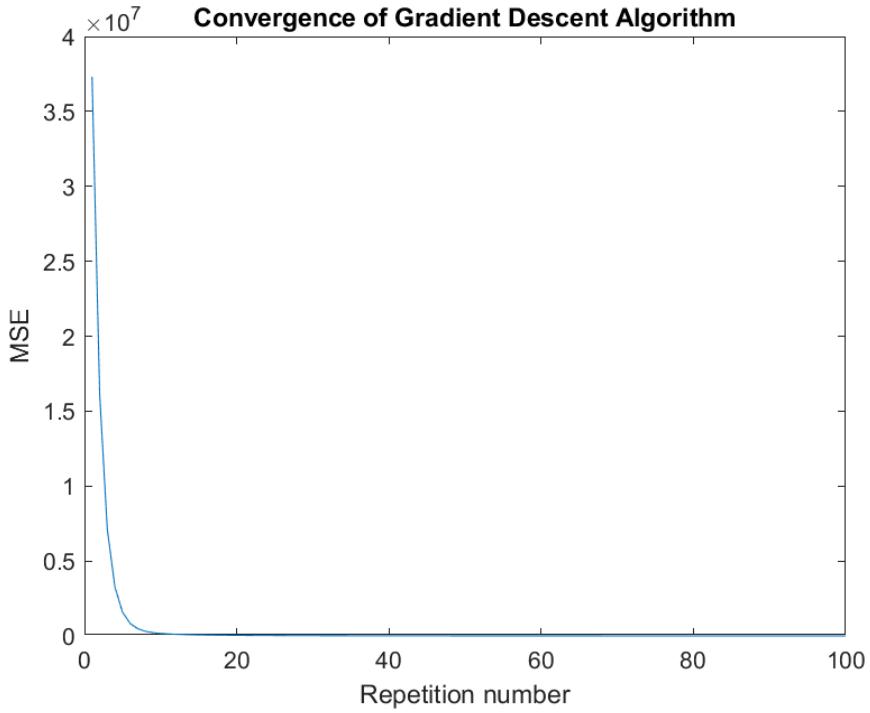
General differences:

Clarity vs. noise: The denoised image prioritizes clarity by reducing noise, but this comes at the cost of losing detail. The balance between noise reduction and detail preservation is a common challenge in image processing.

Usability: Depending on the context (e.g., medical imaging), the denoised image may be more usable for analysis if it highlights significant structures, but may lack important details necessary for full diagnosis.

Question 4)





Why gradient method in this code is superior in most cases to the method used in the previous question:

1. Adaptive Optimization:

- Learning Rate and Iterations: Gradient descent optimizes the denoising process iteratively by adjusting weights based on the error gradient, allowing it to adaptively improve the solution over multiple iterations.
- Error Minimization: The algorithm specifically aims to minimize the difference between the noisy image and the estimated clean image, leading to potentially better reconstruction of the original details.

2. Flexibility in Convergence:

- Customizable Iterations: The gradient descent approach allows for a specified number of iterations, enabling fine-tuning of image quality based on the complexity of the noise and desired quality.
- Control Over Updates: The learning rate (alpha) can be adjusted to control how aggressively the algorithm converges to the optimal solution, allowing for a balance between convergence speed and stability.

3. Efficiency in Handling Noise:

- Dealing with Non-Uniform Noise: Gradient descent can adapt better to various noise patterns compared to a fixed kernel approach, which may not effectively filter out all types of noise present in real-world images.
- Preservation of Features: Through its iterative nature, the gradient descent method can potentially preserve important image features while still reducing noise, addressing the common challenge of losing detail in denoising processes.

4. Error Feedback Loop:

- Continuous Error Monitoring: The method logs the error at each iteration, providing insights into convergence and allowing adjustments if necessary. This is an important aspect of ensuring the denoised image remains close to the actual clean version.

5. Generalization:

- Applicability to Various Problems: Gradient descent is a robust optimization technique used across many domains. Its principles can be readily applied to other image processing tasks beyond denoising, making it a more versatile approach.

Question 5)

Anisotropic Diffusion: Overview

Anisotropic Diffusion is an advanced, noise-reducing image filtering technique that maintains relevant image features, such as edges. Unlike conventional Gaussian filters, which have an equal response in all directions, anisotropic diffusion adapts the smoothing process according to local image features.

Key Concepts:

Edge Preservation: It reduces noise but does not let the edges blur off, unlike what usually happens in traditional methods.

Diffusion Equation: The methodology involves a PDE with variable diffusion coefficients, whose values are determined by the gradients of the image.

Isotropic vs. Anisotropic: Isotropic methods treat all directions equal, while anisotropic ones will vary the smoothing depending on the content of the image.

How It Is Done:

Initialization: An initial image is considered, usually noisy.

Computation of Gradients: The gradients of the image are computed to determine the presence of edges.

Diffusion Coefficient: Provide a diffusion coefficient which changes with the magnitude of the gradient. This is normally performed via a function like $g(|\nabla I|)$, where $|\nabla I|$ is the magnitude of the gradient. Iterative Updating: Perform iterative updating of the image by the diffusion equation until convergence or a specified number of iterations.

Anisotropic Diffusion Code Analysis

1. Setup and Parameter Initialization

```matlab
clear; close all; clc;

addpath('./S2_Q5_utils')

method = 2; % 0 for blur, 1 for simple AD, 2 for sophisticated AD

K = 3;  % Edge threshold parameter
```

- Clear Environment: Clears the workspace, closes figures, and clears the command window.
- Method Selection: The `method` variable determines the type of diffusion to be applied (blur, simple anisotropic diffusion, or sophisticated anisotropic diffusion).
- Edge Threshold (`K`): This parameter influences the edge-stopping function, which helps in preserving edges during diffusion.

2. Image Reading and Noise Addition

```matlab
f = imread('t1.jpg');

f = double(f(:,:,1)); % Convert to double and take the first channel (grayscale)

f = f + randn(size(f))*10; % Add Gaussian noise

f_orig = f; % Store the original noisy image
```

- The image is read and converted to grayscale (assuming a single channel). Gaussian noise is added to simulate a noisy input image.

3. PDE Time-Stepping Parameters

```matlab
delta_t = 0.1; % Time step

t_end = 30; % Total time for diffusion

lambda = 1; % Diffusion scaling factor
```

- Time Parameters: `delta_t` defines the time step for the diffusion process, while `t_end` sets the total duration of the simulation.

4. Main Loop for Anisotropic Diffusion

```
matlab

for t = 0:delta_t:t_end

    [dfdc dfdr] = gradient(f); % Compute gradients

    gradmag_f2 = dfdr.^2 + dfdc.^2; % Gradient magnitude squared


    div_f = (circshift(dfdr,[-1 0])-circshift(dfdr,[1 0]) + circshift(dfdc,[0
-1])-circshift(dfdc,[0 1])) / 2; % Divergence


    c = 1 ./ (1 + gradmag_f2/(K/2)^2); % Edge-stopping function

    if method == 0

        c = ones(size(c))/5; % Simple blurring

    end


    rhs = c .* div_f; % Right-hand side of PDE


    if method == 2

        [dcdc dcdr] = gradient(c);

        rhs = rhs + dcdc .* dfdc + dcdr .* dfdr; % Sophisticated version

    end


    f = f + lambda * delta_t * rhs; % Update image


    if mod(counter, 10) == 0 % Display every 10th iteration

        blah = round(size(f, 2) / 2);

        f_comp(:, blah:end) = f(:, blah:end);
```

```
        imshow(f_comp, [0 255]);
        title(['time = ' num2str(t) ' s']);
        drawnow;
    end
    counter = counter + 1;
end
```

- Gradient Calculation: The gradients in both the x and y directions are computed using the `gradient` function.
- Gradient Magnitude: The squared magnitude of the gradient is calculated to assess edge strength.
- Divergence Calculation: The divergence of the gradient is computed using `circshift`, which helps in approximating the PDE.
- Edge-Stopping Function: The coefficient `c` is calculated based on the gradient magnitude. It determines how much diffusion occurs based on edge strength.
- PDE Right-Hand Side: The right-hand side of the PDE is formed, and for the sophisticated method, additional terms related to the gradient of `c` are included.
- Image Update: The image is updated using the diffusion equation, iterating over time.
- Display: The image is displayed every 10 iterations, showing the original and the diffused image side by side.

Conclusion

The code successfully implements anisotropic diffusion, which helps reduce noise while maintaining edge clarity. The selection of parameters such as K, delta_t, and lambda can greatly influence the outcomes. Further Considerations Parameter Tuning: Trying out various values for K, delta_t, and t_end can lead to different results regarding noise reduction and edge clarity. Method Selection: Analyzing the differences among the three methods (blur, simple, sophisticated) can offer valuable insights into the balance between noise reduction and detail retention. Performance: Depending on the size and complexity of the image, it may be beneficial to optimize the code for better performance, particularly for larger images or when using more iterations.