CS663, Assignment 2 **Yash Sanghvi, Pradyot Prakash**
Instructor: Prof. Suyash Awate 13D070042, 130050008
Question 2: Edge-preserving Smoothing using Bilateral Filtering

Problem Statement

(30 points) Edge-preserving Smoothing using Bilateral Filtering.
Input image: 2/data/barbara.mat. Assume the pixel dimensions to be equal along both axes, i.e., assume an aspect ratio of 1:1 for the axes.
Corrupt the image with independent and identically-distributed additive zero-mean Gaussian noise with standard deviation set to 5% of the intensity range. Note: in Matlab, randn() gives random numbers drawn independently from a Gaussian with mean 0 and standard deviation 1.
Write code for bilateral filtering (standard slow algorithm is also fine) and apply it (one pass over all pixels) to the input image. For efficiency in Matlab, the code should, ideally, have maximum 1 2 for loops to go over the rows and columns of the image. At a specific pixel p, the data collection with a window, weight computations, and weighted averaging can be performed without using loops.
Define the root-mean-squared difference (RMSD) as the square root of the average, over all pixels, of the squared difference between a pixel intensity in the original image and the intensity of the corresponding pixel in the filtered image, i.e., given 2 images A and B with N pixels each, $RMSD(A,B) := \sqrt{(1/N)\Sigma_p(A(p)-B(p))^2}$, where A(p) is the intensity of pixel p in image A.
Tune the parameters (standard-deviations for Gaussians over space and intensity) to minimize the RMSD between the filtered and the original image.

1. Write a function myBilateralFiltering.m to implement this.
2. Show the original, corrupted, and filtered versions side by side, using the same (gray) colormap.
3. Show the mask for the spatial Gaussian, as an image.
4. Report the optimal parameter values found, say σ_{space} and $\sigma_{intensity}$, along with the optimal RMSD.
5. Report RMSD values for filtered images obtained with **(i)** $0.9\sigma_{space}$ **and** $\sigma_{intensity}$, **(ii)** $1.1\sigma_{space}$ **and** $\sigma_{intensity}$, **(iii)** σ_{space} **and** $0.9\sigma_{intensity}$, and (iv) σ_{space} **and** $1.1\sigma_{intensity}$, with all other parameter values unchanged.

Code

(i) Code for Bilateral Filter

```
function [outputImage, noisyImage, err] = myBilateralFiltering(
    inputImage, W, sigmaS, sigmaR)
        inputImage = double(inputImage);
        minIntensity = min(min(inputImage));
        maxIntensity = max(max(inputImage));

        stdDev = 0.05 .* (maxIntensity - minIntensity);
        noise = stdDev .* randn(size(inputImage));
        noisyImage = inputImage + noise;
```

```matlab

    [sizeX, sizeY] = size(noisyImage);
    outputImage = zeros(size(inputImage));

    maskS = fspecial('gaussian', [W, W], sigmaS);
    half = floor(W/2);

    norm_consR = sqrt(2*pi) * sigmaR;
    for i = 1:sizeX
        for j = 1:sizeY

            % li = max(1, i-half); ri = min(sizeX, i+half);
            % ti = max(1, j-half); bi = min(sizeY, j+half);
            % bloc(li:ri, ti:bi) = noisyImage(li:ri, ti:bi) - noisyImage(i, j);
            % maskR = exp(-0.5 .* (bloc .^ 2)) ./ norm_consR;

            bloc = zeros(W, W);
            for x = -half:half
                for y = -half:half
                    if i + x > 0 && i + x <= sizeX && j + y > 0 && j + y <= sizeY
                        bloc(1+half+x, 1+half+y) = noisyImage(i + x, j + y);
                        maskR(1+half+x, 1+half+y) = exp(((noisyImage(i + x, j + y) - noisyImage(i, j)) ^ 2) * (-0.5) / sigmaR^2) / norm_consR;
                    else
                        bloc(1+half+x, 1+half+y) = 0;
                        maskR(1+half+x, 1+half+y) = 0;
                    end
                end
            end

            mask = maskS .* maskR;
            num = sum(sum(bloc .* mask));
```

```
                        den = sum(sum(mask));
                        outputImage(i, j) = num/den;
                end
        end

        err = RMSD(inputImage, outputImage);
end
```

(ii) Main Script

```
%% MyMainScript
tic;
%% Your code here
W = 5; % should always be odd, can't be even

barbara_struct = load('../data/barbara');
barbara = barbara_struct.imageOrig;

sigmaS = 1.5;
sigmaR = 9.7;
[enhanced, noisy, err] = myBilateralFiltering(barbara, W, sigmaS,
    sigmaR);
plotAndSave(barbara, noisy, enhanced, 'optimal', 1);
fprintf('optimal: dist: %d\n', err);

[enhanced, noisy, err] = myBilateralFiltering(barbara, W, 0.9*sigmaS
    , sigmaR);
plotAndSave(barbara, noisy, enhanced, '9space', 2);
fprintf('0.9 space: dist: %d\n', err);

[enhanced, noisy, err] = myBilateralFiltering(barbara, W, 1.1*sigmaS
    , sigmaR);
plotAndSave(barbara, noisy, enhanced, '11space', 3);
fprintf('1.1 space: dist: %d\n', err);

[enhanced, noisy, err] = myBilateralFiltering(barbara, W, sigmaS,
    0.9*sigmaR);
plotAndSave(barbara, noisy, enhanced, '9range', 4);
fprintf('0.9 range: dist: %d\n', err);

[enhanced, noisy, err] = myBilateralFiltering(barbara, W, sigmaS,
    1.1*sigmaR);
plotAndSave(barbara, noisy, enhanced, '11range', 5);
fprintf('1.1 range: dist: %d\n', err);

maskS = fspecial('gaussian', [W, W], sigmaS);
figure(10000);
v = myLinearContrastStretching(maskS);
imshow(v, []);
axis on;
```

```
title('gaussian along space');
saveas(10000, 'space_gaussian.png');
toc;
```

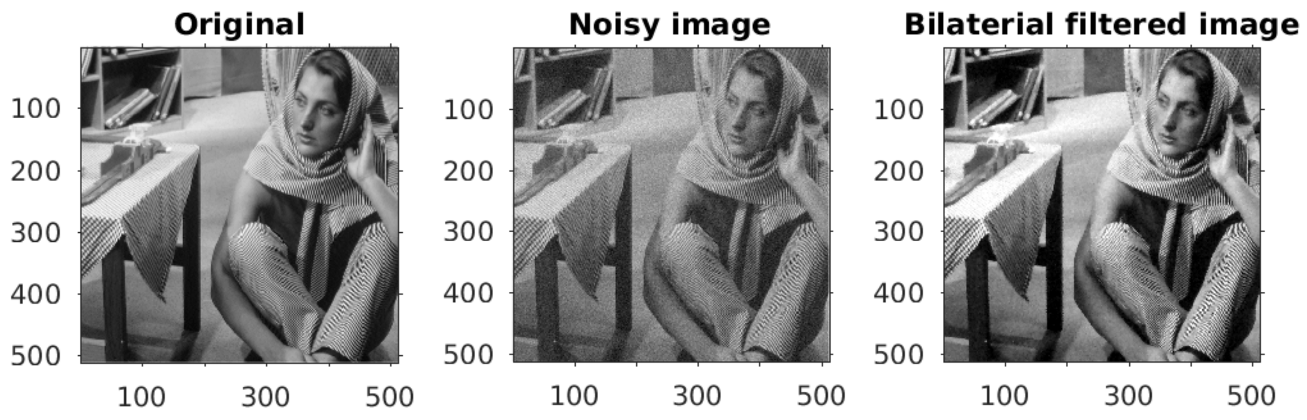
Explanation

In bilateral filtering, a weighted average is done of the neighboring pixels. The way the weights of the terms are calculated, makes the approach interesting. The contribution to the weights comes from both the spatial and the intensity domains.

The code operates with a window size of 5x5 and the parameter optimizations have been done with respect to this window size. A space gaussian is computed with standard deviation 1.5 which is around 1/3 of the window size. The intensity mask also is Gaussian in nature with a standard deviation of 9.7, around 2 times the window size. The minimum RMSD error comes out to be 3.300 which is a local minima with respect to the neighboring parameter values as asked by the problem statement. A square 5x5 window runs over the entire image with the exception of boundary points, where a similar window operation has been used but the "imaginary" points outside the image region are all set to zero, effectively rendering their contribution as null.

The optimal parameters were figured out using running an iterative loop over the two parameters. Both varied from 1 to 10 with a step size of 1. This initial search found the parameters to be 2(space) and 9(range). For a more refined search, we searched for space in the interval 1:0.1:3 and range in 8:0.1:10 (in matlab syntax). This gave the minimum value of the RMSD. The gaussian noise added to the image indeed makes it look distorted and the fitering process clearly shows the robustness of the algorithm.

Result Images

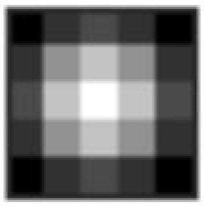


(Left) Original Image **(Centre)** Corrupted Image **(Right)** Image after applying Bilateral Filter of optimum parameter to minimize RMSD between the source image and the image obtained after bilateral filtering.

Optimal Parameters

The optimal parameters were figured out using running an iterative loop over the two parameters. Both varied from 1 to 10 with a step size of 1. This initial search found the parameters to be 2(space) and 9(range). For a more refined search, we searched for space in the interval 1:0.1:3 and range in 8:0.1:10 (in matlab syntax). This gave the minimum value of the RMSD at parameters
σ_{space} **= 1.5 and** $\sigma_{intensity} = 9.7$ **with optimal RMSD = 3.300023**

1. RMSD at $0.9\sigma_{space}$ and $\sigma_{intensity} = 3.313036$
2. RMSD at $1.1\sigma_{space}$ and $\sigma_{intensity} = 3.305919$
3. RMSD at σ_{space} and $0.9\sigma_{intensity} = 3.338741$
4. RMSD at σ_{space} and $1.1\sigma_{intensity} = 3.306026$



Gaussian Kernel used for making the patches isotropic