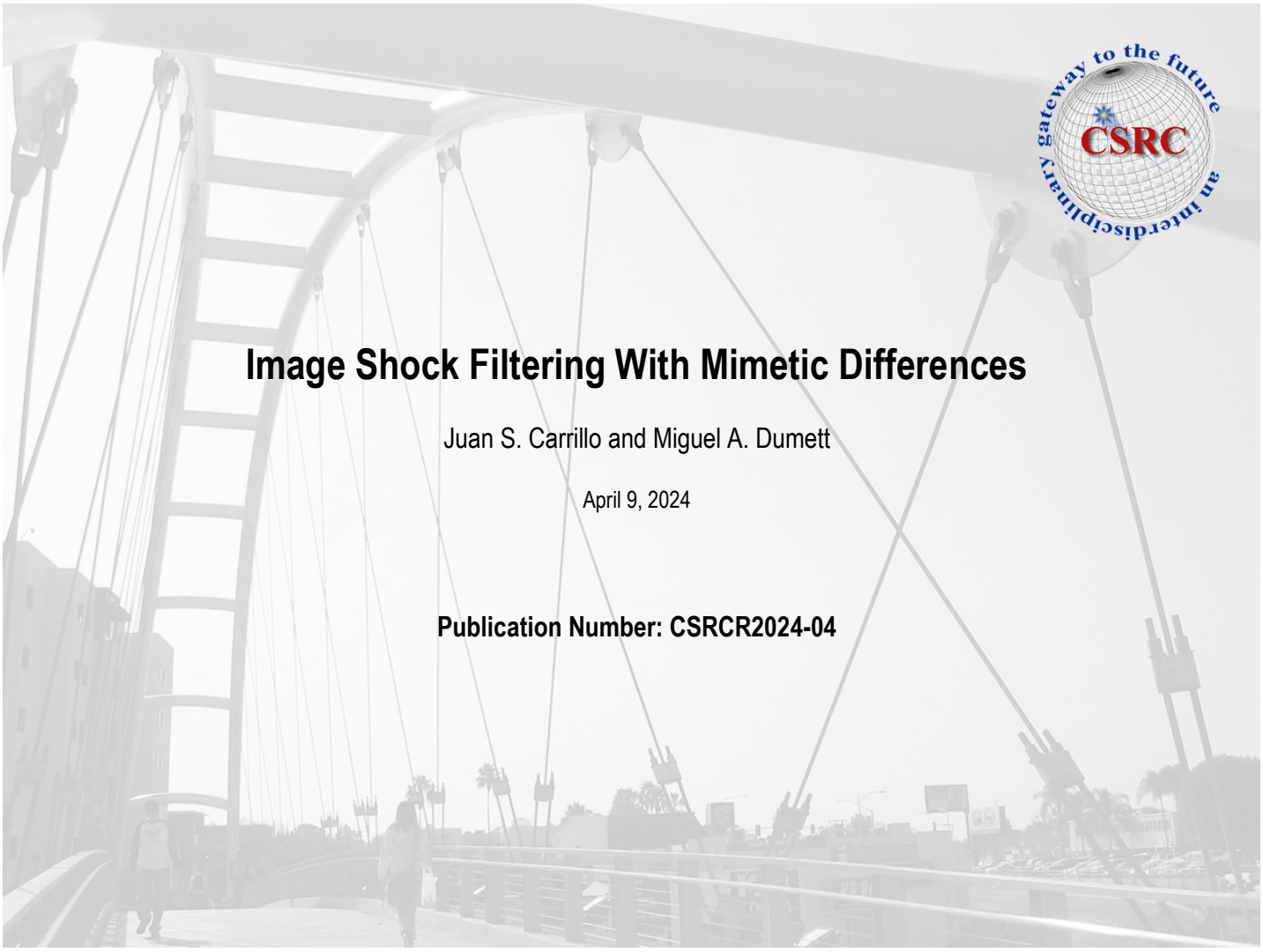

Image Shock Filtering With Mimetic Differences

Juan S. Carrillo and Miguel A. Dumett

April 9, 2024

Publication Number: CSRCR2024-04

Computational Science & Engineering Faculty and Students Research Articles

Database Powered by the Computational Science Research Center Computing Group & Visualization Lab

COMPUTATIONAL SCIENCE & ENGINEERING

SAN DIEGO STATE UNIVERSITY

Computational Science Research Center
College of Sciences
5500 Campanile Drive
San Diego, CA 92182-1245
(619) 594-3430

Image Shock Filtering With Mimetic Differences

Juan S. Carrillo* and Miguel A. Dumett †‡

April 2024

Abstract

In this report, we utilize the Mimetic Operators Library Enhanced to construct discrete analogs of the Corbino-Castillo mimetic differences for solving some partial differential techniques for deblurring images.

1 Introduction

Image processing is a pivotal field within computer science and engineering, encompassing a wide array of techniques aimed at manipulating and enhancing digital images to extract meaningful information or improve their visual quality. One fundamental challenge in image processing is image deblurring, which involves the restoration of blurred or degraded images to recover their original clarity and sharpness.

The importance of image deblurring, see Figure 1, lies in its wide range of applications across various domains. In fields such as medical imaging, astronomy, surveillance, and photography, obtaining clear and accurate images is crucial for analysis, diagnosis, interpretation, and decision-making processes. Blurred images can result from factors such as motion blur, defocus blur, or imperfections in imaging devices, and addressing these blurs is essential for ensuring the reliability and usefulness of the captured visual data.

Figure 1: Image Debluring

Among the numerous methods developed for image deblurring, the shock filter has emerged as a notable approach for enhancing image quality. The shock filter operates based on principles derived from mathematical equations and concepts related to image analysis and edge detection. By leveraging the properties of shock waves and nonlinear operators, the shock filter aims to identify and enhance image features, particularly edges and singularities, which play a crucial role in defining object boundaries and structures within images.

*Computational Science PhD Program at San Diego State University (jcarrillo5698@sdsu.edu).
†Editor: Jose E. Castillo
‡Computational Science Research Center at San Diego State University (mdumett@sdsu.edu).

The development of a multidimensional shock filter necessitates the resolution of differential equations [5], incorporating nonlinear amalgamations of propagation components and edge-detection operators. These equations are tailored to identify and amplify edges by modifying the spatial dynamics surrounding pivotal features in the image, see Figure 2. Through prioritizing the zero crossings of differential operators and elements sensitive to curvature, the shock filter adeptly refines edges, thereby augmenting the clarity of the entire image [4].

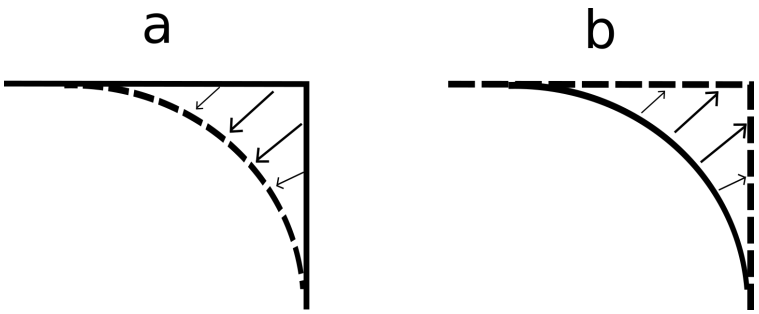

Figure 2: (a) Gaussian filter effect on edges, (b) Shock filter effect on edges

Moreover, the shock filter's implementation could involve intricate computational procedures and iterative algorithms aimed at processing image data efficiently while maintaining computational tractability. Techniques such as discretization, numerical analysis, and optimization could be employed to ensure the filter's effectiveness and stability during the deblurring process.

The application of the shock filter has yielded promising results in various experiments and real-world scenarios. By demonstrating its capabilities on standard images and datasets, researchers have showcased the filter's ability to significantly enhance image quality, restore details, and improve visual fidelity. Furthermore, the shock filter's adaptability to different types of images, including grayscale and color images, underscores its versatility and potential for widespread use in diverse applications requiring image deblurring and enhancement.

2 Methods

This section outlines the approach taken to apply the shock filter, a fundamental technique in image enhancement. This segment provides a concise overview of the computational and mathematical procedures involved in utilizing the shock filter to sharpen edges and improve image clarity.

In a single dimension, the shock filter is expressed as a partial differential equation (PDE):

$$u_t = -\left|u_x\right| F(u_{xx}) \tag{1}$$

with F defined as (2)

$$\begin{cases} F(x) > 0 & if\ x > 0 \\ F(x) = 0 & if\ x = 0 \\ F(x) < 0 & if\ x < 0 \end{cases} \tag{2}$$

For this case, the function sign will be used to fulfill the place of F, see (3)

$$sgn(x) = \begin{cases} 1 & if\ x > 0 \\ 0 & if\ x = 0 \\ -1 & if\ x < 0 \end{cases} \tag{3}$$

Equation (1) could be extended to work in 2D by using (4)

$$u_t = -\left|\nabla u\right| F(\mathcal{L}(u)) \tag{4}$$

In order to detect borders in horizontal and vertical directions it is possible to use (5)

$$\mathcal{L}(u) = \Delta u \tag{5}$$

To enhance this, it is possible to employ a different definition for L, thereby incorporating non-tangential directions into the search path, see (7)

$$\mathcal{L}(u) = \nabla u \cdot \begin{bmatrix} u_{xx} & u_{xy} \\ u_{yx} & u_{yy} \end{bmatrix} \nabla u \tag{6}$$

$$\mathcal{L}(u) = \begin{bmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial u}{\partial y} \end{bmatrix} \cdot \begin{bmatrix} u_{xx} & u_{xy} \\ u_{yx} & u_{yy} \end{bmatrix} \begin{bmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial u}{\partial y} \end{bmatrix}$$

$$\mathcal{L}(u) = \begin{bmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial u}{\partial y} \end{bmatrix} \cdot \begin{bmatrix} u_{xx}u_x + u_{xy}u_y \\ u_{yx}u_y + u_{yy}u_y \end{bmatrix}$$

$$\mathcal{L}(u) = u_x^2 u_{xx} + 2u_x u_y u_{xy} + u_y^2 u_{yy} \tag{7}$$

$$u_t = -\left|\nabla u\right| F(u_x^2 u_{xx} + 2u_x u_y u_{xy} + u_y^2 u_{yy}) \tag{8}$$

2.1 Preprocessing

Before applying the filtering, it is a common and essential practice to preprocess the image by normalizing its pixel values. This preprocessing step ensures that the pixel intensities are within a standardized range, typically between 0 and 1. Normalization is crucial for ensuring that the filtering process operates consistently across different images and prevents any biases that might arise from variations in pixel intensity scales.

Conversion to Grayscale (if necessary): If the image is in color, it's often converted to grayscale to simplify the processing. This step involves transforming the image from a multi-channel representation (e.g., RGB) to a single-channel representation where each pixel represents the intensity of light.

2.2 Initial condition and boundary condition

After the process mentioned in the preceding section has been completed, it is possible to proceed to discussing the practical application of the filtering technique. This includes not only solving the partial differential dquation (PDE) but also defining initial and boundary conditions. In this case, the initial condition refers to the state of the images after the preprocessing steps have been carried out.

The choice of boundary conditions plays a critical role in shaping the behavior of filtering algorithms, particularly when dealing with edge detection and enhancement. One commonly used boundary condition is the Neumann boundary condition. In this context, Neumann boundaries assume that the gradient of the image intensity at the edges is zero. This means that the rate of change of pixel values at the boundary is constrained, resulting in a smoothing effect along the edges of the image. While Neumann boundary conditions can help reduce artifacts and maintain continuity, they may also lead to the loss of sharpness in important features and edges. Therefore, careful consideration of the trade-offs between smoothness and edge preservation is essential when selecting Neumann boundaries for image filtering tasks.

3 Metrics

When assessing the efficacy of image deblurring techniques, it is essential to employ appropriate metrics to quantitatively measure the extent of improvement achieved. Two commonly used metrics for this purpose are the Mean Structural Similarity Index (MSSI)[1] and the Peak Signal-to-Noise Ratio (PSNR).

The MSSI metric evaluates the structural similarity between the original and deblurred images by considering luminance, contrast, and structure. Higher MSSI values indicate greater similarity between the images.

On the other hand, the PSNR metric measures the quality of the deblurred image by quantifying the ratio between the maximum possible power of the image signal and the power of the noise present in the deblurred image [6], implying superior deblurring performance. By utilizing this metric, researchers can comprehensively evaluate the effectiveness of different deblurring algorithms and make informed decisions regarding their suitability for practical applications.

$$\text{SSIM}(x, y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \tag{9}$$

The Structural Similarity Index (SSIM) formula, (9), is a crucial tool for evaluating the likeness between two images, commonly applied in various image processing and computer vision contexts. Within the SSIM equation, the mean pixel intensity of each image (μ_x and μ_y) and their corresponding standard deviations (σ_x and σ_y) are computed, representing the central tendency and variability of pixel values, respectively. Additionally, the covariance (σ_{xy}) between pixel values is determined, indicating their joint variation.

These components collectively contribute to the numerator of the SSIM formula, which assesses the similarity between local pixel intensity patterns. The denominator, incorporating the products of mean and variance terms alongside stabilization constants c_1 and c_2, serves to normalize the SSIM computation, mitigating potential instability.

Through this formulation, SSIM offers a comprehensive evaluation of luminance, contrast, and structural similarity between images, yielding a single scalar value that quantifies their overall resemblance. Higher SSIM values signify a greater likeness between the images, facilitating robust evaluation and comparison of image processing algorithms across academic research and practical applications [1].

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

Where MAX is the maximum value of a pixel (255).

$$\text{MSE} = \frac{1}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n}[I(i,j) - K(i,j)]^2$$

4 Numerical solution - Mimetic scheme

Implementing a shock filter using mimetic methods involves transforming continuous problems into discrete formulations. Equation (8) represents the core equation governing the behavior of the system. In this equation, u_t denotes the temporal derivative of the field variable u, which represents the evolving state of the system. The term ∇u refers to the gradient of u, capturing its spatial variations.

Equation 7 illustrates the multidimensional nature of the problem, where u depends on multiple spatial directions denoted by x and y. The terms u_{xx}, u_{xy}, u_{yx}, and u_{yy} represent the second partial derivatives of u with respect to these spatial directions.

The final expression in (8) represents the temporal evolution of u under the influence of the shock filter. Here, F denotes a function that modulates the effect of the spatial derivatives on the temporal evolution of the system. This equation serves as the cornerstone of the numerical scheme, guiding the evolution of the system over time.

In this case, the initial condition u signifies the image itself. This indicates that the discretization of u is determined by the values within the matrix corresponding to the image, while ensuring a uniform grid. Consequently, we denote this discretization as U.

To illustrate the derivation of the Corbino-Castillo mimetic scheme [2], an example is provided for the term $u_x^2 u_{xx}$.

$$u_x^2 u_{xx}$$

By replacing the respective operators G and D, it is discretized into:

$$G_x U^2 D_x G_x U$$

Where G_x and D_x are the 2D gradient and divergence operators in the horizontal direction.

$$I_x^{f\to c} G_x U^2 D_x G_x U$$

In this final step, the respective interpolator is included to map back to the domain of the centers of the staggered mesh, specifically in the horizontal direction. This is necessary as the output of the operator G_x results in the faces of the staggered mesh oriented in the horizontal direction.

$$T_1 = diag(I_x^{f\to c} G_x U^2) D_x G_x U \tag{11}$$

Matlab diagonal function is applied to maintain the correct sizes for performing the multiplications.

Following the same structure, the mimetic scheme is derived for each of the terms. For $2u_x u_y u_{xy}$:

$$2[I_x^{f\to c} G_x U I_y^{f\to c} G_y U D_x I_x^{c\to f} I_y^{f\to c} G_y U]$$

$$T_2 = 2[diag(I_x^{f\to c} G_x U) diag(I_y^{f\to c} G_y U) D_x I_x^{c\to f} I_y^{f\to c} G_y U] \tag{12}$$

For $u_y^2 u_{yy}$:

$$I_y^{f\to c} G_y U^2 D_y G_y U$$

$$T_3 = diag(I_y^{f\to c} G_y U^2) D_y G_y U \tag{13}$$

And finally, the norm of the gradient, $|\nabla u|$, is determined by:

$$Norm = \sqrt{I_y^{f\to c} G_y U^2 + I_x^{f\to c} G_x U^2}$$

We have the discrete version of the shock filter summarized as:

$$-Norm \cdot F(T_1 + T_2 + T_3) \tag{14}$$

and the method to calculate the next step and update the image by applying the filter is through the equation:

$$U_{n+1} = U_n + d_t * Norm \cdot F(T_1 + T_2 + T_3) \tag{15}$$

Where dt is related to the discretization of time and is determined based on the Neumann stability.

$$\frac{dt}{dx^2 + dy^2} \leq 0.5 \tag{16}$$

For the experiments conducted in this work, dt was set to 0.1

We utilize the Mimetic Operators Library Enhanced (MOLE) [3] library to construct the discrete analogs of the spatial differential operators according to the Corbino-Castillo mimetic differences approach [2].

Algorithm Mimetic Differences for the Shock Filter

Load Mole path;
RGB ← Load Image;
Image Resize;
Add padding and boundary condition;
RGB ← Add noise;
UX ← convert to double(RGB);
Create 2D Staggered grid;
D ← div2D(k, m, dx, n, dy);
G ← grad2D(k, m, dx, n, dy);
dt ← $dx^2/(10 \times \alpha)$;
Icf ← interpolCentersToFacesD2D(k, m, n);
Ifc ← interpolFacesToCentersG2D(k, m, n);
U2 ← UX;
for $t \leftarrow 1$ **to** $Tmax$ **do**
    create vector with image data;
    Calculate the norm of U2;
    term1 ← $(ux)^2 U_{xx}$;
    term2 ← $2uxuyU_{xy}$;
    term3 ← $(uy)^2 U_{yy}$;
    F ← sign$(term1 + term2 + term3)$;
    L ← U2 + dt $\times$ $(norm \times F)$;
    U2 ← reshape$(L, 514, 514)$;
end for
U2 ← uint8$((U2 - 1))$;
Display results;

5 Results

In this section, we present the results of applying a filtering technique to an image contaminated with Gaussian noise. The filtering process aims to enhance the image quality by reducing noise and improving visual clarity. The effectiveness of the filter is evaluated using two key metrics: the Structural Similarity Index (SSIM) and the Peak Signal-to-Noise Ratio (PSNR). By analyzing the changes in these metrics before and after filtering, we assess the performance of the filter and its impact on image quality improvement.

5.1 Filtering an Image

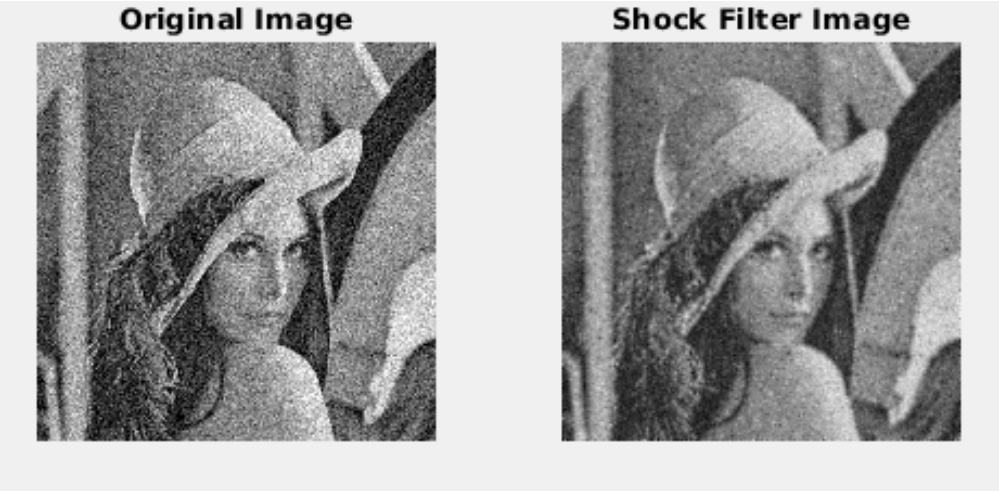

Figure 3: Result after 30 iterations of Shock Filter

The filter was applied to an image using 30 iterations. The original image was corrupted with Gaussian noise, and the objective of the filter was to enhance the image quality, see Figure 3. According to the results obtained from Structural Similarity Index (SSIM) and Peak Signal-to-Noise Ratio (PSNR), the filter demonstrated improvements in both metrics. Initially, the SSIM value of the noisy image was measured at 0.27, but after applying the filter, it increased to 0.49. Similarly, the PSNR of the noisy image was 20.04, and it rose to 23.94 after filtration. These results underscore the effectiveness of the filter in enhancing image quality, showcasing its ability to mitigate noise and enhance visual clarity.

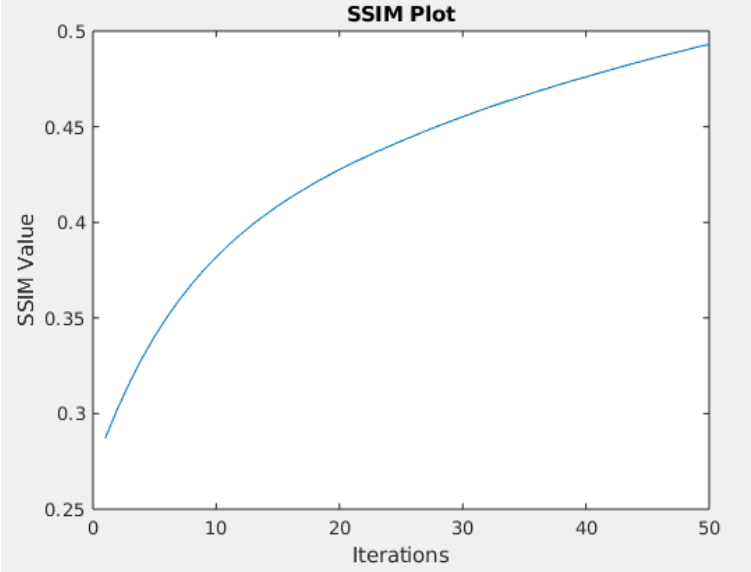

Figure 4: SSIM behavior over iterations

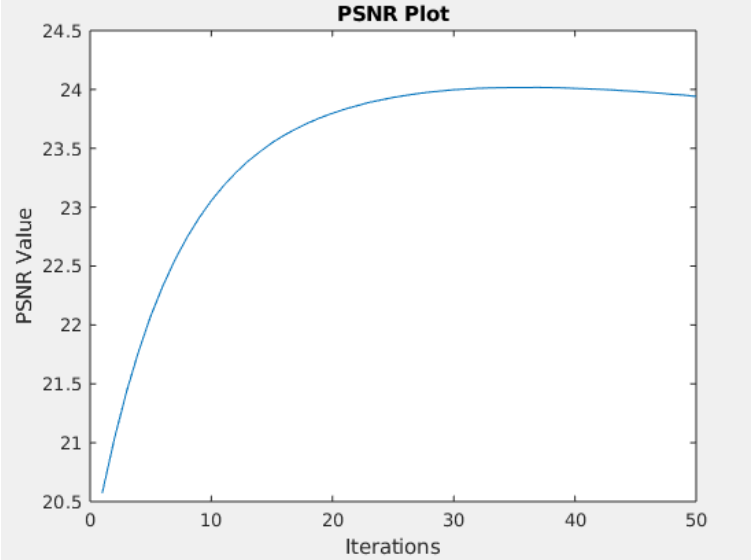

Figure 5: PSNR behavior over iterations

It also included a graph illustrating the variation of SSIM , see Figure 4 and PSNR , see Figure 5 values with each iteration. These graphs provide a visual representation of how the image quality metrics evolve throughout the filtering process, offering insights into the effectiveness and convergence behavior of the applied filter.

In the appendix, we include the MATLAB source code.

References

[1] Illya Bakurov et al. "Structural similarity index (SSIM) revisited: A data-driven approach". In: *Expert Systems with Applications* 189 (2022), p. 116087.

[2] Johnny Corbino and Jose E Castillo. "High-order mimetic finite-difference operators satisfying the extended Gauss divergence theorem". In: *Journal of Computational and Applied Mathematics* 364 (2020), p. 112326.

[3] *MOLE: Mimetic Operators Library Enhanced*. Version v2.0. June 2023. DOI: 10.5281/zenodo.7996615. URL: https://github.com/csrc-sdsu/mole.

[4] James H Money. "Variational methods for image deblurring and discretized Picard's method". PhD thesis. Citeseer, 2006.

[5] Stanley Osher and Leonid I Rudin. "Feature-oriented image enhancement using shock filters". In: *SIAM Journal on numerical analysis* 27.4 (1990), pp. 919–940.

[6] Alexander Tanchenko. "Visual-PSNR measure of image quality". In: *Journal of Visual Communication and Image Representation* 25.5 (2014), pp. 874–878.

Appendix

```
% Mimetic Differences for the Shock Filter
% Juan S. Carrillo, COMP670, Project
clc; clear; close all

% mole
addpath('mole-master/mole_MATLAB')

% Read lena image
RGB = imread('cameraman.tif');

RGB = imresize(RGB,[512 512]);
%RGB = rgb2gray(RGB);

% BC Neumann (add before noise)
RGB = padarray(RGB,[1 1],255,'both');
RGB(1,:) = RGB(2,:);
RGB(end,:) = RGB(end-1,:);
RGB(:,1) = RGB(:,2);
RGB(:,end) = RGB(:,end-1);

%Image to compare
orig = RGB;

%add noise to image
RGB = imnoise(RGB,'gaussian');


%U = rgb2gray(RGB);
U = RGB;

% convert integer values to double
UX = double(U);


% Define mimetic operators
k = 2;   % Order of accuracy
m = 512;  % Number of cells along the x-axis
n = m;   % Number of cells along the y-axis
a = 0;   % West
b = 512;   % East
c = 0;   % South
d = 512;   % North
dx = (b-a)/m;
dy = (d-c)/n;

% 2D Staggered grid
xgrid = [a a+dx/2 : dx : b-dx/2 b];
ygrid = [c c+dy/2 : dy : d-dy/2 d];

% 2D Mimetic divergence operator
D = div2D(k, m, dx, n, dy); % D from MOLE
D1=D(:,1:end/2); %Dx
D2=D(:,(end/2)+1:end); %Dy

% 2D Mimetic gradient operator
G = grad2D(k, m, dx, n, dy); %G from MOLE
G1=G(1:end/2,:); %Gx
G2=G((end/2)+1:end,:); %Gy

% alpha
alpha = 1;

% Check neumann stability
% Neumann stability criterion
dt = dx^2/(10*alpha); % alpha = 1
```

```
66
67
68 %get interpolators
69 Icf = interpolCentersToFacesD2D(k, m, n); %D
70 Ifc = interpolFacesToCentersG2D(k, m, n); %G
71
72 Icfx = Icf(1:(m+1)*(n) , 1:(m+2)*(n+2) );
73 Icfy = Icf((m+1)*(n)+1:(m+1)*(n)*2 , (m+2)*(n+2)+1:(m+2)*(n+2)*2 );
74 Ifcx = Ifc( 1:(m+2)*(n+2),1:(m+1)*(n));
75 Ifcy = Ifc( (m+2)*(n+2)+1:(m+2)*(n+2)*2 ,(m+1)*(n)+1:(m+1)*(n)*2 );
76
77 % Initial Condition
78 U2 = UX; % Initial condition image
79 %%%%%%%%%%%
80 % iteration-Loop
81 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
82 for t = 1 : 50
83     % create vector with image data
84     U2  = reshape(U2, [], 1);
85
86     ux = Ifcx*G1*U2;
87     uy = Ifcy*G2*U2;
88
89     %norm of U
90     norm =  sqrt( ux.^2 + uy.^2 );
91
92     %term1
93     % (ux)^2 Uxx
94     term1 = (spdiags(double(ux.^2), 0, length(ux), length(ux))) * D1*G1*U2;
95
96     %term2
97     %2ux uy uxy
98     term2 = 2* spdiags(ux, 0, length(ux), length(ux)) * spdiags(double(uy), 0, length(uy),
       length(uy)) * D1*Icfx*Ifcy*G2*U2;
99
100    %term3
101    %(uy)^2 Uyy
102    term3 =   (spdiags(uy.^2, 0, length(uy), length(uy))) * D2*G2*U2;
103
104    F = sign(term1 + term2 + term3);
105
106    % Operation (updating the image)
107    L = U2 + dt*(norm.*F);
108
109    % reshape vector to image
110    U2=reshape(L, 514, 514);
111
112 end
113 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
114
115 % convert image back to uint8
116 U2=uint8((U2-1));
117
118 % Output
119 figure
120 subplot(1,2,1)
121 imshow(U)
122 title('Original Image')
123 subplot(1,2,2)
124 imshow(U2)
125 title('Shock Filter Image')
126
127 ssim(U,orig)
128 ssim(U2,orig)
129
130 psnr(U,orig)
131 psnr(U2,orig)
```