Mimetic Differences for the Perona-Malik Equation

Thomas H. Keller and Miguel A. Dumett

July 21, 2023

Publication Number: CSRCR2023-06

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

© 2023

Mimetic Differences for the Perona-Malik Equation

Thomas H. Keller *and Miguel A. Dumett †‡

July 21, 2023

Abstract

The Perona-Malik diffusion is a non-linear anisotropic process. The equation is used to smooth or blur images. The advantage compared to a linear diffusion is that the equation does not smooth or blur the image around areas that have a high likelihood of being edges. This paper solves the Perona-Malik diffusion equation with mimetic differences. The mimetic method can be utilized for image processing driven by PDEs. Furthermore, the output of the algorithm illustrates that the edges are significantly sharper, and the other areas are more blurred or smoothed.

1 Introduction

This report is about solving the Perona-Malik diffusion equation with the Corbino-Castillo [1] mimetic differences in 2D. The Perona-Malik diffusion is a non-linear anisotropic process that was presented in 1987 by Perona and Malik [2]. In principle, the equation smooths or blurs the image, except around image areas that are likely to have edges. In contrast to a linear diffusion, which smooths the entire image in the same way (such as the Gaussian diffusion), the widespread areas should be smoothed while the edges remain sharp. For that reason, the equation is used to denoise or blur images in a smart way.

The mimetic discretization method uses a staggered grid, which is defined at the beginning of the code with the MOLE library [3]. The mimetic analogs for the differential operators for the gradient and the Laplacian are defined on a staggered grid. Instead of using the finite difference method by manually creating matrices or loops, the mimetic method creates them by just defining the grid and mimetic operators. As a result, the calculation of the PDE is clear and straightforward. The grid is exactly related to the number of pixels in the image. From there, it follows that the pixels and the grid are matching, and the values from the pixel centers are used for the calculations. Overall, the mimetic method simplifies the calculation and makes it clearer.

*Computational Science Master Program at San Diego State University (tkeller1432@sdsu.edu).
†Jose E. Castillo:Editor
‡Computational Science Research Center at San Diego State University (mdumett@sdsu.edu).

This work distinguishes from [4] because of the following:

1. The original image is not smoothed by a Gaussian kernel. The Gaussian kernel parameter in Bazan et al., changes at each time step.
2. Even though the anisotropy function g is equivalent, the parameter K is fixed and it does not change at each time step as in Bazan et al.
3. This work can be more easily generalized to higher-order of accuracy.
4. This work utilizes image processing MATLAB toolbox APIs for cropping, padding, etc.
5. This work does not attempt to solve the Perona-Malik equation with finite-differences or finite elements.

2 Equation with BC and IC

This section describes the equation, initial condition and the boundary conditions.

2.1 Perona Malik Equation

Perona Malik Equation [2]:

$$I_t = \boldsymbol{\nabla\cdot}\left(c(x,y,t)\nabla I\right) \tag{1}$$

Diffusion coefficient:

$$c\left(\|\nabla I\|\right) = \frac{1}{1+\left(\frac{\|\nabla I\|}{K}\right)^2} \tag{2}$$

where:

I = Image 512×512px [double]
I_t = Partial derivative with respect to time of the image content [double]
c = Diffusion coefficient [double]
K = Constant, controls sensitivity to edges [integer]
∇ = Gradient
$\boldsymbol{\nabla\cdot}$ = Divergence

2.2 Initial Condition

Any image with a minimum size of 512×512px can be used as the input image for the Perona-Malik algorithm. The algorithm converts the image to a black-and-white image and crops it to a size of 512×512px. As an example image, the colored Tom image (in Figure 1) is used. After converting it to black and white, the image is available in its initial condition as an 8-bit gray scale image with values from 0 to 255, with black as 0, and white as 255.

Figure 1: Tom, black and white image, initial condition, 512x512px

2.3 Boundary Condition

Neumann boundaries are being used for that project. Neumann boundaries request no change at the boundaries, as shown in equation (3). For that reason, the image domain is extended with pixels around the whole image. The added pixel values are set to the same values as the outermost edge of the image. That results in a smooth blurring around the image. The image has then a size of 514×514px.

$$\frac{\partial I}{\partial n} = 0 \qquad \text{in } \partial\Omega \times (0, +\infty), \tag{3}$$

where Ω denotes picture domain 512×512px [2].

3 Diffusion coefficient

The diffusion coefficient of the Perona-Malik equation is shown in equation (2). As a function of the coefficient, the likelihood of blurring the edges is reduced. Hence, the coefficient is a function of the image information, which is determined by the image gradient. This is implemented in MATLAB with the mimetic method by calculating the 2D gradient of the actual image for each iteration.

Constant $K = 7$, which controls the sensitivity to edges, is used. This value is found empirically. The next paragraph explains the determination of the parameter K more detailed.

In the first attempt of solving the equation with the mimetic method, the grid was set the values from paper [2]. The values were set to [a,b]×[c,d] = [0,1]×[0,1]. That results in a small values for dx and dy. For that reason, $K = 7000$ was found to obtain right image outputs.

```
m = 512;  % Number of cells along the x-axis
n = m;    % Number of cells along the y-axis
a = 0;    % West
b = 1;    % East
c = 0;    % South
d = 1;    % North
```

```
7       dx = (b-a)/m;
8       dy = (d-c)/n;
9
```

Listing 1: Matlab code part (grid based on 0,1)

The expected values range from 5 to 25 based on [5]. In this example, $K = 15$ is used. After completing the code and performing several test runs, the result emphasizes the difference is the definition of the grid, i.e. the another project has [a,b]×[c,d] = [0,512]×[0,512] as a grid. As a result, dx and dy equals to 1. Therefore, K can be much smaller for this grid and it is set to 7.

4 Image processing

First, the code initializes the image as an uint8 image (8-bit grayscale image). Then, it casts the image values to double and sets the initial and boundary conditions. Afterwards, the algorithm iterates through the main loop with the mimetic operators, and at the end, it converts the image back into an uint8 image. These steps are described in detail in the following subsections.

4.1 8-bit Image Value Handling

The initial image is an 8-bit black and white image, which has unsigned integer (uint8) values from 0 to 255. The mimetic method does not support values of type integer. For that reason, the image has to be converted from uint8 to double. This is accomplished by casting the uint8 to double and then adding 1.The conversion from double to uint8 is the opposite procedure. This follows from the offset in the colormap of MATLAB (to work with the image data, the 2D image was converted into a vector with the reshape command of MATLAB).

4.2 Iterations

The main loop of the algorithm calculates the new image (I_{0+dt}) (see equation (4)). For clarification the equation has the same symbols for the Operators as the Code. First, the loop creates a vector from the image data (I) and calculates the image gradient. Then, it adapts the diffusion coefficient and multiplies the diffusion coefficient (C) with the image gradient (GI). Afterwards, the algorithm takes the divergence (D) and multiplies it by dt.

$$I_{0+dt} = I_0 + dt \cdot (D(C \cdot (GI))) \tag{4}$$

For 2D linear problems, the Neumann stability factor is

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

Because of the non-linearity of equation 2, the Neumann stability factor is determined empirically, which leads to $dt = 0.1$. Last, the respective value is added to the actual image. The number of iterations has to be adjusted and depends on the case.

5 Code

The following Matlab code was utilized for solving the Perona-Malik equation.

```
% Perona Malik Equation with mimetic method (by Thomas Keller)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Read image
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
RGB = imread('tom2.jpg');

% crop image to 512x512px
RGB = imcrop(RGB,[0 0 512 512]);

% BC Neumann
RGB = padarray(RGB,[1 1],255,'both');
IRGB2(1,:) = RGB(2,:);
RGB(end,:) = RGB(end-1,:);
RGB(:,1) = RGB(:,2);
RGB(:,end) = RGB(:,end-1);

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

% Read and display an RGB image, and then convert it to grayscale
I = rgb2gray(RGB);

% convert integer values to double
IX = double(I)+1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Mimetic difference parameters
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
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);

% 2D Mimetic gradient operator
G = grad2D(k, m, dx, n, dy);

```

```
% alpha
alpha = 1;

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

% Initial condition (IC)
I2 = IX; % IC image

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Iteration-Loop
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for t = 1 : 30

    % create vector with image data
    I2  = reshape(I2, [], 1);

    % gradient of the image
    I3 = G*I2;

    % Diffusion coefficient
    % C caluclation
    k = 7;
    C = 1./(1+(I3./k).^2);

    % Operation
    L1 = C.*I3;
    L = I2 + dt*(D*L1);

    % reshape vector to image
    I2=reshape(L, 514, 514);

end

% convert image back to uint8
I2=uint8((I2-1));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Output image
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
figure
subplot(1,2,1)
imshow(I)
title('Original Image')

subplot(1,2,2)
imshow(I2)
title('Perona Malik Image')

```

Listing 2: Matlab code

6 Verification

6.1 Original Image

Alternatively, the Perona Malik diffusion can be applied to an original image (Figure 2). The output proves that the edges become significantly sharper and the other areas are more blurred.

Figure 2: Tom black and white image: unmodified initial image and output image of MATLAB code, 50 iterations, $K = 7$

6.2 Noisy Image

A measure of the denoise quality is the mean square error (MSE) and the peak signal-to-noise ratio (PSNR). The two graphs (Figures 3 and 4) show that about 90 iterations should result in the best image quality.

The result of the Perona Malik diffusion for a noisy input image is portrayed in the right image in Figures 5, 6, 7. The number of iterations varies from 50 to 90 to 140. The comparison to the original image highlights that the edges are sharper and the whole image is denoised. It is good to see that the 90 iterations that were evaluated in Figures 3 and 4 result in the subjectively best output. As expected, the right image in Figure 5 is still noisy, and the one in Figure 7 has lost many details. The number of iterations required to denoise images is significantly greater than for blurring unmodified images (Figure 1).

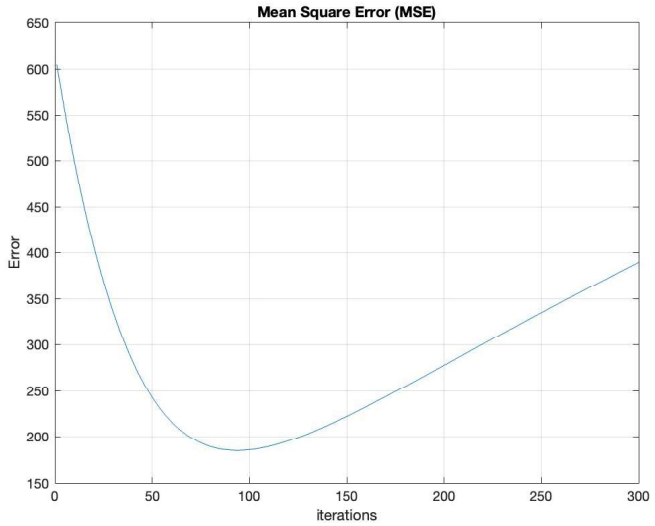

Figure 3: MSE for Tom black and white image, 300 iterations, K = 7

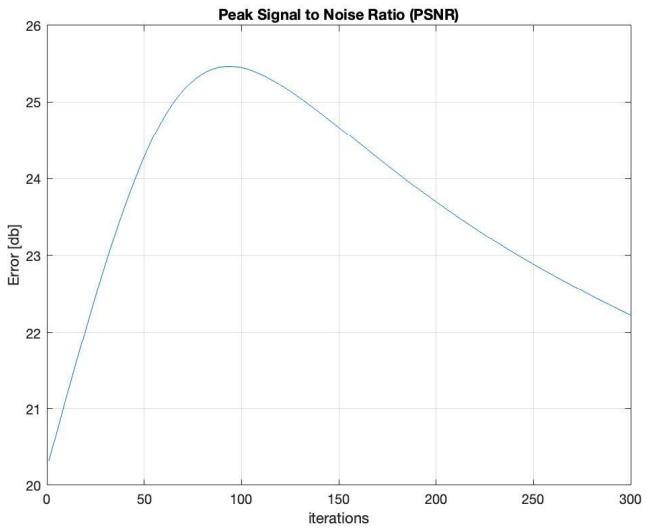

Figure 4: PSNR for tom black and white image, 300 iterations, K = 7

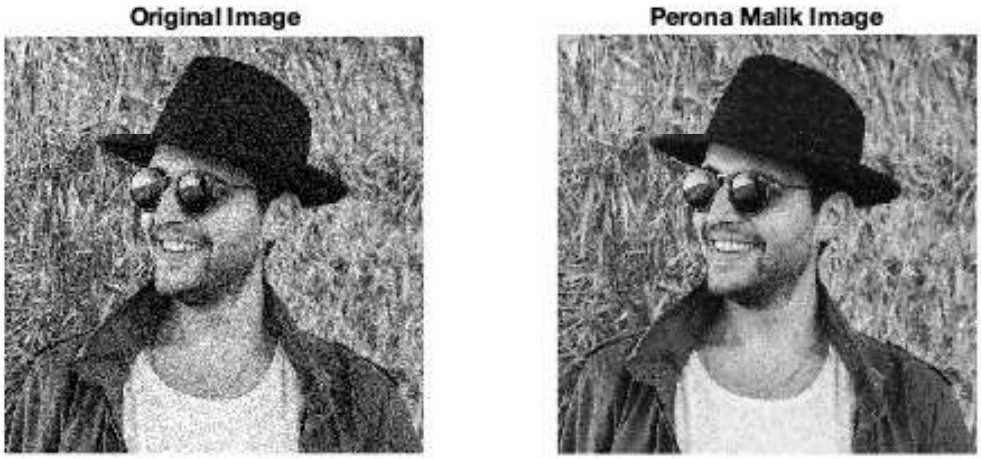

Figure 5: Tom black and white image: noisy initial image (noise with variance of 0.01) and output image of MATLAB code, 50 iterations, $K = 7$.

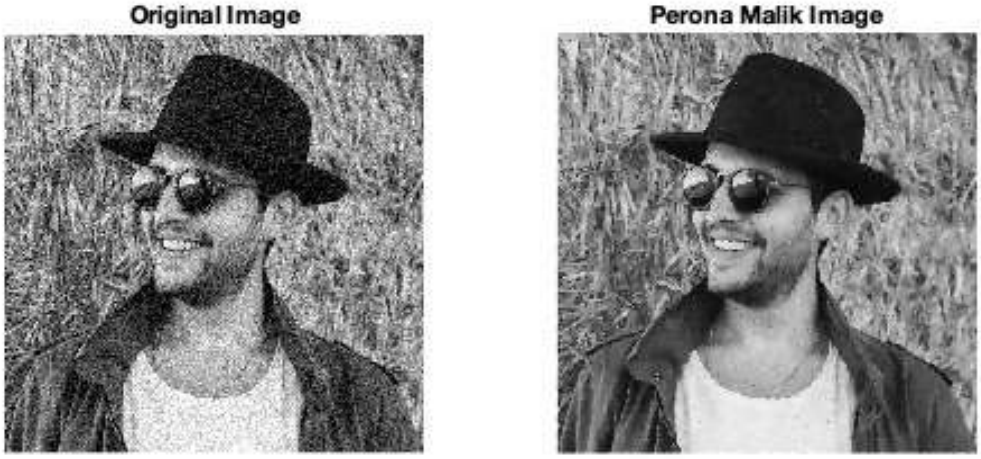

Figure 6: Tom black and white image: noisy initial image (noise with variance of 0.01) and output image of MATLAB code, 90 iterations, $K = 7$.

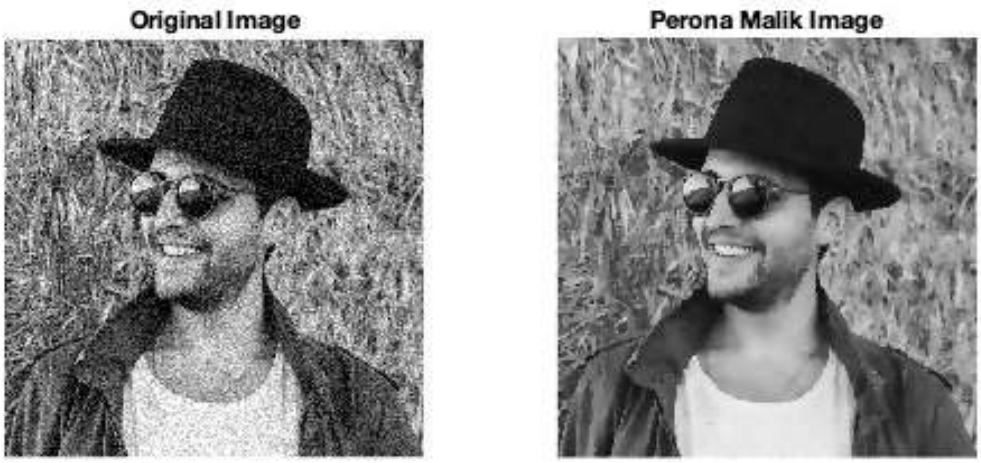

Figure 7: Tom black and white image: noisy initial image (noise with variance of 0.01) and output image of MATLAB code, 140 iterations, $K = 7$.

References

[1] Corbino, J., and Castillo, J.E., High-order mimetic difference operators satisfying the extended Gauss divergence theorem, Journal of Computational and Applied Mathematics, 364 (2020).

[2] Perona, P., and Malik, J., Scale-Space and Edge Detection Using Anisotropic Difussion, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 12, No. 7, July 1990.

[3] Corbino, J., and Castillo, J.E., MOLE: Mimetic Operators Library Enhanced: The Open-Source Library for Solving Partial Differential Equations using Mimetic Methods, 2017. GitHub, `https://github.com/csrc-sdsu/mole`.

[4] Bazan, C., Abouali, M., Castillo, J.E., and Blomgren, P., Mimetic Finite Difference Methods in Image Processing, Comput. Appl. Math. 30 (3), 2011.

[5] Fabian Herzog, PeronaMalikDiffusion, 2019. GitHub, `https://github.com/fubel/PeronaMalikDiffusion`.