Thomas Keller
SDSU ID 828545348

Project Report

COMP 670
December 9, 2022

Perona Malik with Mimetic Method

Abstract

The Perona-Malik diffusion is a non-linear anisotropic process. The equation is used to blur or denoise images. The advantage compared to a linear diffusion is that the equation does not blur the image around areas that have a high likelihood of being edges. This project report solves the Perona-Malik diffusion equation with the mimetic method. The mimetic method works for image processing. Furthermore, the output of the algorithm illustrates that the edges are significantly sharper and the other areas are more blurred.

1 Introduction

This report for the subject COMP670 is about solving the Perona-Malik diffusion equation with the mimetic method. The Perona-Malik diffusion is a non-linear anisotropic process. In principle, the equation blurs the image, except around image areas that are likely to have edges. In contrast to a linear diffusion, which blurs the entire image in the same way (such as the Gaussian diffusion), the widespread areas should be blurred while the edges remain sharp. For that reason, the equation is used to denoise or blur images [1], [2].

2 Equation with BC and IC

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

2.1 Perona Malik Equation

Perona Malik Equation [3]:

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

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	= Image at the iteration t
c	= diffusion coefficient [double]
K	= constant, controls sensitivity to edges [integer]
∇	= Gradient
$\boldsymbol{\nabla}\boldsymbol{\cdot}$	= 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 Lena image is used. After converting it to black and white, the image is available in its initial condition as an 8-bit grayscale image with values from 0 to 255, with black as 0 and white as 255.

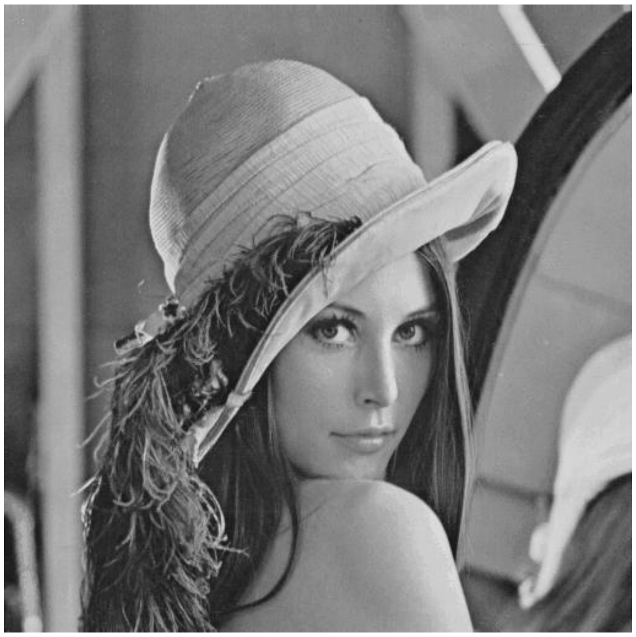

Figure 1: Lena image, black and white image, initial condition, 512x512px [4]

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.

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

where Ω denotes picture domain 512×512px [5].

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.

For the constant K, which controls the sensitivity to edges, K = 7 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 (see Matlab Code 3) from the paper [5]. The values were set to [a,b]×[c,d] = [0,1]×[0,1]. That results in a small dx and dy. For that reason, K = 7000 was found to obtain right image outputs.

```matlab

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
dx = (b-a)/m;
dy = (d-c)/n;

```

Listing 1: Matlab code part of peronamalik.m file, grid based on 0,1

The expected values ranges from 5 to 25 based on [6]. In this example K = 15 is used. To solve another project, the centered finite difference method is utilized. The same principle is present in the mimic method. 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 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, converts the image back into an uint8 image. Those steps are described in the following sections.

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 [7].

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 the Neumann stability factor 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 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 shows the code for peronamalik.m file.

```
% Perona Malik Equation with mimetic method
% Thomas Keller, COMP670, Project
clc; clear; close all
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% mole
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
addpath('../mole_MATLAB')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Read lena image
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
RGB = imread('lena_color.jpg');

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;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
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
    % Neumann stability criterion
    % dt = 0.1
    dt = dx^2/(10*alpha); % alpha = 1

    % IC
    I2 = IX; % Initial condition 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
    figure
    subplot(1,2,1)
    imshow(I)
    title('Original Image')
    subplot(1,2,2)
    imshow(I2)
    title('Perona Malik Image')

```

Listing 2: Matlab code for file peronamalik.m

6 Verification

6.1 Original Image

The result of the Perona Malik diffusion for an unmodified image as input is shown in the right image of Figure 2. The comparison to the original image illustrates that the edges are significantly sharper and the other areas are more blurred.

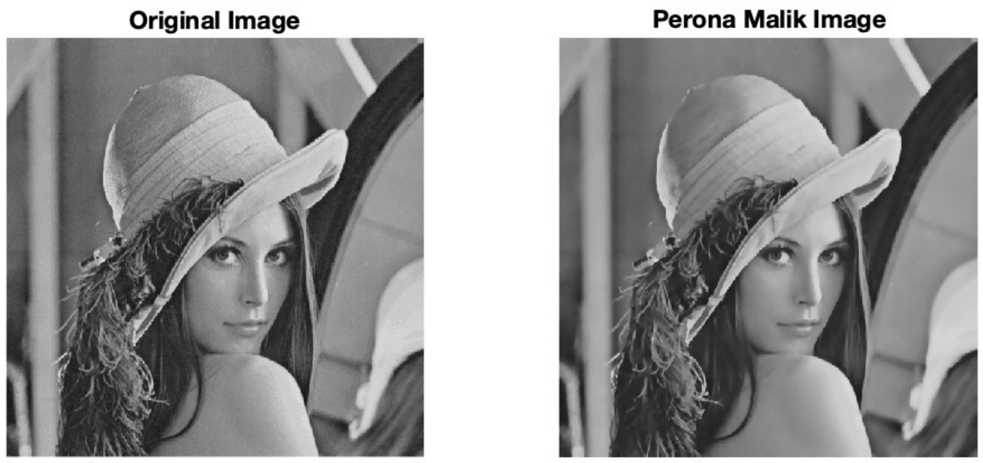

Figure 2: Lena black and white image: unmodified initial image and output image of MATLAB code, 20 iterations, K = 7 [4]

6.2 Noisy Image

The result of the Perona Malik diffusion for a noisy image as input is portrayed in the right image of Figure 3. The comparison to the original image highlights that the edges are sharper and the whole image is denoised. The number of iterations to denoise images is significantly greater than for blurring unmodified images.

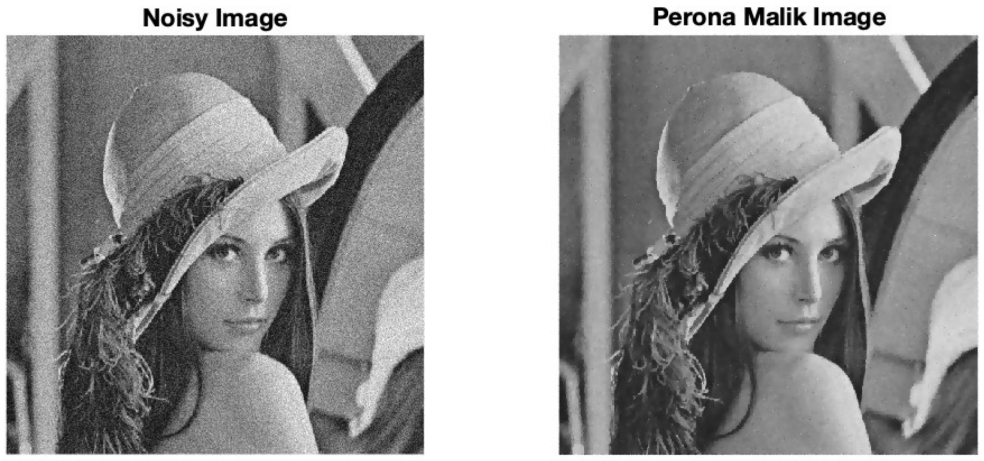

Figure 3: Lena black and white image: noisy initial image and output image of MATLAB code, 50 iterations, K = 7 [4]

References

[1] Wikipedia, "Anisotropic diffusion." [Online]. Available: https://en.wikipedia.org/wiki/Anisotropic_diffusion

[2] J. C. C. BAZAN, M. ABOUALI and P. BLOMGREN, "Mimetic finite difference methods in image processing," *Computational & Applied Mathematics*, 2011. [Online]. Available: https://pdfs.semanticscholar.org/2d8f/acea04124f05bb45779e3cfee477aff3ef82.pdf

[3] P. Perona and J.Malik, "Scale-space and edge detection using anisotropic diffusion." [Online]. Available: http://image.diku.dk/imagecanon/material/PeronaMalik1990.pdf

[4] Wikipedia, "Lenna." [Online]. Available: https://en.wikipedia.org/wiki/Lenna

[5] M. Wielgus, "Perona-malik equation and its numerical properties." [Online]. Available: https://arxiv.org/pdf/1412.6291.pdf

[6] Fubel, "Peronamalikdiffusion perona malik diffusion." [Online]. Available: https://github.com/fubel/PeronaMalikDiffusion

[7] MathWorks, "8-bit and 16-bit images." [Online]. Available: https://www.mathworks.com/help/matlab/creating_plots/working-with-8-bit-and-16-bit-images.html