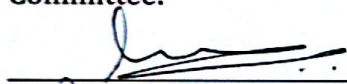


**MODELING, ANALYSIS, AND IMPLEMENTATION OF FINITE DIFFERENCE SCHEMES  
FOR NONLINEAR DIFFUSION WITH APPLICATIONS TO IMAGE PROCESSING**

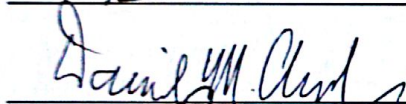
by

Armelle S. Franklin  
A Thesis  
Submitted to the  
Graduate Faculty  
of  
George Mason University  
in Partial Fulfillment of  
The Requirements for the Degree  
of  
Master of Science  
Mathematics

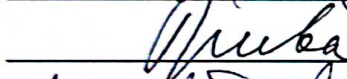
Committee:



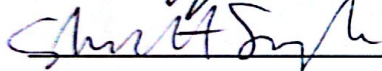
Dr. Padmanabhan Seshaiyer  
Thesis Director



Dr. Daniel Anderson, Committee Member



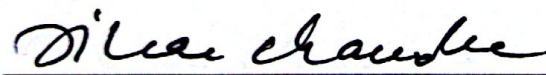
Dr. Igor Griva, Committee Member



Dr. Stephen Saperstone  
Department Chair



Dr. Timothy L. Born  
Associate Dean for Student and Academic  
Affairs, College of Science



Dr. Vikas Chandhoke  
Dean, College of Science

Date: May 2, 2013

Spring Semester 2013  
George Mason University  
Fairfax, VA

MODELING, ANALYSIS, AND IMPLEMENTATION OF FINITE DIFFERENCE SCHEMES  
FOR NONLINEAR DIFFUSION WITH APPLICATIONS TO IMAGE PROCESSING

A Thesis submitted in partial fulfillment of the requirements for the degree of  
Master of Science at George Mason University

By

Armelle S. Franklin  
Bachelor of Science  
Temple University, 2007

Director: Padmanabhan Seshaiyer, Professor  
Department of Mathematical Sciences

Spring Semester 2013  
George Mason University  
Fairfax, VA

© 2013, Armelle S. Franklin

All Rights Reserved

## **DEDICATION**

This is dedicated to my motivating mother, brother, boyfriend and to the Lockheed Martin IS&GS-National Engineering Leadership Development Program Class of 2011.

## **ACKNOWLEDGEMENTS**

I cannot thank Dr. Padmanabhan Seshaiyer (“Dr. Padhu”) enough for the extensive time, attention and aid that he dedicated to my research and to me throughout my efforts on this thesis. Special thanks to professors Dr. Daniel Anderson and Dr. Igor Griva for being members of my committee. I would also like to thank the many friends, relatives, and supporters who have made this happen. Many of my coworkers helped me choose a topic, offered words of advice and my supervisors were very flexible in allowing me time off to work on this thesis for which I am so grateful.

## TABLE OF CONTENTS

	Page
List of Figures .....	vii
List of Tables .....	x
List of Abbreviations and Symbols.....	xi
Abstract .....	xii
Chapter I – Introduction.....	1
Chapter II – Numerical Methods .....	5
2.1    Explicit Perona-Malik Second Order Difference .....	6
2.2    Implicit Perona-Malik Second Order Difference: .....	12
2.3    Initial and Boundary Conditions .....	12
2.3    Shorthand Notation and Other Numerical Methods.....	14
Chapter III – Stability and Convergence .....	17
3.1    Stability Analysis of the Explicit Numerical Scheme.....	17
Theorem 3.1 .....	18
Lemma 3.1.1 .....	18
Lemma 3.1.2: .....	23
Lemma 3.1.3 .....	25
3.2    Stability Analysis of the Implicit Numerical Scheme .....	32
Theorem 3.2.....	33
Lemma 3.2.1 .....	33
Lemma 3.2.2 .....	38
Chapter IV – MATLAB in a Literate Programming Style .....	42
4.1    Implementation of Explicit Perona-Malik Numerical Scheme .....	43
4.2    Implementation of Implicit Perona-Malik Numerical Scheme .....	49
4.3    Simulated Noise: Random (Additive), Gaussian and Speckle.....	59
Chapter V – Experimental Results.....	61

5.1	Performance Metrics .....	61
5.2	Computational Grid and Interpolation Methods .....	76
5.2.1	Finer Computational Grid: Bilinear vs. Spline Interpolation.....	77
5.2.2	Non-Uniform Computational Grid .....	81
Chapter VI – Conclusion and Future Work .....		86
Appendix A – MATLAB for Explicit PM Numerical Scheme .....		88
Appendix B – MATLAB for Implicit PM Numerical Scheme.....		93
Appendix C – MATLAB Script to run experiments with 1000 iterations.....		97
References .....		104

## LIST OF FIGURES

Figure	Page
Figure 1: Numerical Discretization Grid.....	5
Figure 2: MATLAB code for creating the mesh grid to overlay the computational grid space.....	42
Figure 3: MATLAB code for using bilinear interpolation to create the computational grid space (i.e. the computational image).....	43
Figure 4: MATLAB code for using spline interpolation to create the computational grid space (i.e. the computational image).....	43
Figure 5: MATLAB Flow for Explicit PM Numerical Scheme .....	43
Figure 6: MATLAB code to call function pm_explicit to run Explicit PM 2 <sup>nd</sup> Order Difference scheme .....	44
Figure 7: MATLAB code for maximum allowed time step, dtmax .....	44
Figure 8: MATLAB code for time step calculated as 99% of dtmax .....	44
Figure 9: MATLAB code for padding image .....	46
Figure 10: MATLAB code for calculating the gradient of the intensities.....	46
Figure 11: MATLAB Code for choice of g function – diffusivity function coefficient... ..	47
Figure 12: MATLAB code for calculating Explicit PM 2 <sup>nd</sup> Order Difference over each ((i, j)) pixel of the computational grid .....	48
Figure 13: MATLAB Code to read MATLAB built-in image - cameraman.tif.....	48
Figure 14: MATLAB code to define parameters in run script for inputs to pm_explicit. ....	49
Figure 15: MATLAB code to downsample the final image to original image grid space size (approximately).....	49
Figure 16: MATLAB Flow for Implicit PM Numerical Scheme .....	50
Figure 17: MATLAB Function for Implicit Matrix.....	50
Figure 18: MATLAB Code – pm_implicit_matrix.m.....	52
Figure 19: MATLAB Code – pm_implicit_matrix.m - 1 <sup>st</sup> Half of Loop.....	53
Figure 20: MATLAB Code – pm_implicit_matrix.m – 2 <sup>nd</sup> Half of Loop .....	55
Figure 21: MATLAB .....	56
Figure 22: MATLAB Code: pm_implicit.m – Bilinear Interpolation and Image Padding .....	56
Figure 23: MATLAB Code: Implicit PM Scheme (pm_implicit.m) – Part 1 of 3 .....	57
Figure 24: MATLAB code – pm_implicit.m – Part 2 of 3 .....	57
Figure 25: MATLAB Code: pm_implicit.m – Part 3 of 3 .....	58
Figure 26: Implicit PM Scheme (pm_implicit.m).....	58
Figure 27: MATLAB code to run Implicit Scheme.....	59
Figure 28: MATLAB Code for Adding Random Noise (reduced by factor of 5). .....	59



Figure 29: MATLAB built-in function imnoise for Gaussian and Multiplicative Noise (Speckle) .....	60
Figure 30: Cameraman Image (256x256 TIFF).....	61
Figure 31: Lena Image (512x512 TIFF) .....	61
Figure 32: Errors of Explicit (Red Star Dotted Line) and Implicit (Green Circle Solid Line) Methods wrt $\Delta t$ . Decrease in error with smaller values of $\Delta t$ validate the implementation of the algorithm.....	64
Figure 33: Explicit PM Visual Results for different values of $\Delta t$ for processing the original image without any noise added (i.e. “Noise” Image = “Original” Image).....	65
Figure 34: (ZOOM IN) Explicit PM Visual Results for different values of $\Delta t$ for processing the original image without any noise added (i.e. “Noise” Image = “Original” Image). .....	66
Figure 35: Implicit PM Visual Results for different values of $\Delta t$ for processing the original image without any noise added (i.e. “Noise” Image = “Original” Image).....	67
Figure 36: (ZOOM IN) Implicit PM Visual Results for different values of $\Delta t$ for processing the original image without any noise added (i.e. “Noise” Image = “Original” Image). .....	68
Figure 37: Explicit PM Visual Results for different values of $\Delta t$ for processing with RANDOM NOISE added. ....	69
Figure 38: Implicit PM Visual Results for different values of $\Delta t$ for processing with RANDOM NOISE added. ....	70
Figure 39: Explicit PM Visual Results for different values of $\Delta t$ for processing with GAUSSIAN NOISE added. ....	71
Figure 40: Implicit PM Visual Results for different values of $\Delta t$ for processing with GAUSSIAN NOISE added. ....	72
Figure 41: Explicit PM Visual Results for different values of $\Delta t$ for processing with simulated SPECKLE.....	73
Figure 42: Implicit PM Visual Results for different values of $\Delta t$ for processing with simulated SPECKLE.....	74
Figure 43: Errors of Explicit (circle) and Implicit (asterix) Methods wrt different $\Delta t$ . ....	75
Figure 44: MATLAB Profile to view time consumption when running MASTER_RUN_PERONA_MALIK.m script.....	76
Figure 45: Bilinear Interpolation of Original Image (BLUE) to 2x Finer Computational Grid (RED) with $dx = \frac{1}{2}$ and $dy = \frac{1}{2}$ and ZOOM (right). ....	77
Figure 46: Spline Interpolation of Original Image (BLUE) to transform to 2x Finer Computational Grid (RED) with $dx = \frac{1}{2}$ and $dy = \frac{1}{2}$ with ZOOM (right) .....	78
Figure 47: Explicit PM Visual Results with Bilinear Interpolation (2 <sup>nd</sup> row) and Spline Interpolation (3 <sup>rd</sup> Row) for $\Delta x = \Delta y = 1$ . No Noise Added. Each row has the final image then the downsampled image.....	79
Figure 48: Explicit PM Visual Results with Bilinear Interpolation (2 <sup>nd</sup> row) and Spline Interpolation (3 <sup>rd</sup> Row) for $\Delta x = \Delta y = 1/2$ . No Noise Added. Each row has the final image then the downsampled image.....	79

Figure 49: Explicit PM Visual Results with Bilinear Interpolation (2 <sup>nd</sup> row) and Spline Interpolation (3 <sup>rd</sup> Row) for $\Delta x = \Delta y = 1/10$ . No Noise Added. Each row has the final image then the downsampled image.....	80
Figure 50: Explicit PM Visual Results with Bilinear Interpolation (2 <sup>nd</sup> row) and Spline Interpolation (3 <sup>rd</sup> Row) for $\Delta x = \Delta y = 1/20$ . No Noise Added. Each row has the final image then the downsampled image.....	80
Figure 51: Explicit PM on Non-Uniform Computational Grid with $\Delta x = 1$ and $\Delta y = 1/2$ . Spline Interpolation. No Noise Added. Bottom Left is the final image (larger image (i.e. more elements since $dx$ and $dy$ are $< 1$ )). Bottom Right is the downsampled image (back to original size) .....	82
Figure 52: Explicit PM on Non-Uniform Computational Grid with $\Delta x = 1/2$ and $\Delta y = 1$ . Spline Interpolation. No Noise Added. Bottom Left is the final image (larger image (i.e. more elements since $dx$ and $dy$ are $< 1$ )). Bottom Right is the downsampled image (back to original size) .....	82
Figure 53: Explicit PM on Non-Uniform Computational Grid with $\Delta x = 1$ and $\Delta y = 1/3$ . Spline Interpolation. No Noise Added. Bottom Left is the final image (larger image (i.e. more elements since $dx$ and $dy$ are $< 1$ )). Bottom Right is the downsampled image (back to original size) .....	83
Figure 54: Explicit PM on Non-Uniform Computational Grid with $\Delta x = 1/3$ and $\Delta y = 1$ . Spline Interpolation. No Noise Added. Bottom Left is the final image (larger image (i.e. more elements since $dx$ and $dy$ are $< 1$ )). Bottom Right is the downsampled image (back to original size) .....	83
Figure 55: Explicit PM on Non-Uniform Computational Grid with $\Delta x = \Delta y = 1/20$ . Spline Interpolation. No Noise Added. Bottom Right is the final image (larger=more elements). Bottom Left is the downsampled image (back to original size) .....	84
Figure 56: Explicit PM on Non-Uniform Computational Grid with $\Delta x = 1$ and $\Delta y = 1/2$ . Spline Interpolation. No Noise Added. Bottom Left is the final image (larger image (i.e. more elements since $dx$ and $dy$ are $< 1$ )). Bottom Right is the downsampled image (back to original size) .....	84
Figure 57: MATLAB Function – pm_explicit_dt.m – Implementation of Explicit PM Numerical Scheme .....	92
Figure 58: MATLAB Function – pm_implicit_dt.m – Implementation of Implicit PM Numerical Scheme .....	96
Figure 59: MATLAB Script – MASTER_RUN_PERONA_MALIK_1000.m.....	99
Figure 60: MATLAB Script - PM_Explicit_Run.m.....	101
Figure 61: MATLAB Script - PM_Implicit_Run.m.....	103

## LIST OF TABLES

Table 1: Description of Fixed Parameters .....	62
Table 2: Explicit PM Results for different values of $\Delta t$ for processing the original image without any noise added.....	63
Table 3: Implicit PM Results for different values of $\Delta t$ for processing the original image without any noise added.....	64
Table 4: Errors and average time of Explicit and Implicit schemes wrt different $\Delta t$ for simulated noisy images .....	74
Table 5: Explicit PM Numerical Results for Bilinear and Spline Interpolation with different $\Delta x=\Delta y$ values .....	81
Table 6: Explicit PM Scheme on Non-Uniform Grid Spacing .....	85

## LIST OF ABBREVIATIONS AND SYMBOLS

Boundary Condition.....	BC
Contrast Parameter.....	K
Copy of Equation ##.....	(## <sub>cp</sub> )
Diffusion Coefficient Function .....	c, c(·)
Initial Condition.....	IC
Intensity (pixel brightness) .....	I
Left Hand Side (of equation) .....	LHS
Partial Derivative with respect to time, t .....	$\partial_t, \frac{\partial}{\partial t}$
Partial Derivative with respect to x.....	$\partial_x, \frac{\partial}{\partial x}$
Partial Derivative with respect to y.....	$\partial_y, \frac{\partial}{\partial y}$
Partial Differential Equation .....	PDE
Perona-Malik .....	PM
Right Hand Side (of equation) .....	RHS
Step size in the x direction .....	$\Delta x$
Step size in the y direction .....	$\Delta y$
Step size in time (t) .....	$\Delta t$
“With Respect To” .....	wrt

## **ABSTRACT**

MODELING, ANALYSIS, AND IMPLEMENTATION OF FINITE DIFFERENCE  
SCHEMES FOR NONLINEAR DIFFUSION WITH APPLICATIONS TO IMAGE  
PROCESSING

Armelle S. Franklin, M.S.

George Mason University, 2013

Thesis Director: Dr. Padmanabhan Seshaiyer

This thesis proposes to model, analyze and implement a nonlinear diffusion model problem for reduction in noise and speckle in image processing applications. Specifically, the Perona-Malik model equation that is widely studied in the image processing community is implemented via explicit and implicit finite difference algorithms. The solution methodology converts discrete image data onto a finer non-uniform grid space via interpolation techniques and applies the proposed numerical algorithms to reduce noise. These numerical algorithms are investigated analytically and computationally for appropriate choices of nonlinear diffusion coefficient functions. We derived conditions for stability and convergence of the proposed numerical algorithms. Numerical experiments are presented on benchmark problems that show the robustness and reliability of the proposed numerical schemes.

## CHAPTER I – INTRODUCTION

Algorithmic enhancements to imagery infected with noise or speckle are accomplished usually with application of a filter sometimes with a loss of accuracy. This loss of accuracy is synonymous to loss of resolution due to an attempt to smooth the image. The convolution of the infected imagery with a filter yields a lower resolution image but with less noise or speckle. A very difficult obstacle to overcome is the blurring of sharp edges in the scene content, a common artifact of linear filtering.

It is well-known that solutions to these denoised images satisfy nonlinear diffusion partial differential equations. These equations describe relationship between the Intensity values with respect to time and space. Over the past two decades there have been attempts to study and enhance denoised images through various partial differential equations.

The first of these PDEs was introduced by Perona & Malik in 1987 which was a nonlinear second order PDE. Namely, this model incorporated anisotropic diffusion filters into the PDEs to generate smoother images while preserving edge information (i.e. location). This nonlinear filter has been significant in processing, enhancement and scale-space analysis of imagery. The idea of scale-space is presented to correlate each solution of the PDE at each time step,  $t$ , to the next “smoother” image. They also examined the benefits of nonlinear filtering with a

second order PDE for not only smoothing, but also edge detection in an image in scale-space. The Perona-Malik equation is also known as anisotropic diffusion (in image processing or computer vision), where the diffusion coefficient is taken to be a function of the gradient of the intensity in the image. The anisotropic diffusion equation that is the basis of the Perona-Malik equation follows:

$$I_t = \frac{\partial I}{\partial t} = \text{div}(c(x, y, t)\nabla I) = c(x, y, t)\Delta I + \nabla c \cdot \nabla I \quad (1)$$

$c(x, y, t)$  is the diffusion coefficient at a given  $(x, y)$  and layer  $t$ . The  $\nabla$  denotes the gradient and  $\Delta$  denotes the Laplacian operator ( $\Delta = \nabla^2$ ). The unique feature of the Perona and Malik deduced that the optimal choice of  $c$  in order to preserve edge location accuracy is for  $c(x, y, t)$  to be a function of the gradient of the intensity,  $I$  at the given point:

$$c(x, y, t) = g(\|\nabla I(x, y, t)\|) \quad (2)$$

Reweriting the divergence operator gives:

$$I_t = \frac{\partial I}{\partial t} = \nabla \cdot (g(|\nabla I|)\nabla I) \quad (3)$$

Perona and Malik make careful selection of the  $g(\|\nabla I(x, y, t)\|)$  function specifically to be bounded between 0 and 1. A common choice for this g function is

$$g(z) = e^{-\left(\frac{z}{K}\right)^2} \quad (4)$$

Another traditional choice for the g function is

$$g(z) = \frac{1}{1 + \left(\frac{z}{K}\right)^2} \quad (5)$$

Note that the constant K is called a “contrast parameter”.

Perona and Malik’s research led to several updated models over the last few years. In 2000, You and Kaveh [2] introduced a fourth order PDE that would further enhance the smoothing with even less blur at the edge locations, but in this anisotropic diffusion model, the diffusion coefficient is taken to be a function of the Laplacian of the intensity in the image. In 2009, Hajiaboli [3] introduced a modification to the You-Kaveh anisotropic diffusion model, by maintaining the same function but returning to the Perona-Malik diffusion coefficient as a function of the gradient of the intensities in the image. The development of these equations have helped introduce a variety of nonlinear diffusion denoising models including fourth order PDEs, spatially regularized models (See [1], [2], [3] and the references



therein). Most of these models have either been experimentally validated or analytically studied. However, there is still a great need to develop numerically stable approximations to such models which is the focus of this thesis.

In this paper, explicit and implicit finite difference schemes are examined analytically and numerically for the Perona and Malik second order PDE. In Chapter 2 of this thesis we develop the finite differences of the Perona-Malik equation. In Chapter 3 we will introduce the explicit method and show that under a stability condition on the step size in time, the scheme is stable. We will also develop the implicit method in Chapter 3 which will be shown to be unconditionally stable. Chapter 4 introduces the reader to literate programming to summarize the MATLAB implementation of the numerical difference schemes. Chapter 5 presents the validation of the methods presented for a benchmark cameraman problem. Finally, Chapter 6 presents conclusion and future research.

## CHAPTER II – NUMERICAL METHODS

In this chapter we will develop the numerical methods for the associated equation (1) presented in Chapter 1 with nonlinear diffusion. These equations are implemented over finite grid spaces using the finite difference method for numerical approximation to the partial differential equation. Finite differences are a simple yet very powerful tool to approximate continuous derivatives on a discrete grid space. Hence, each equation will be discretized using a nearest neighbor approach.

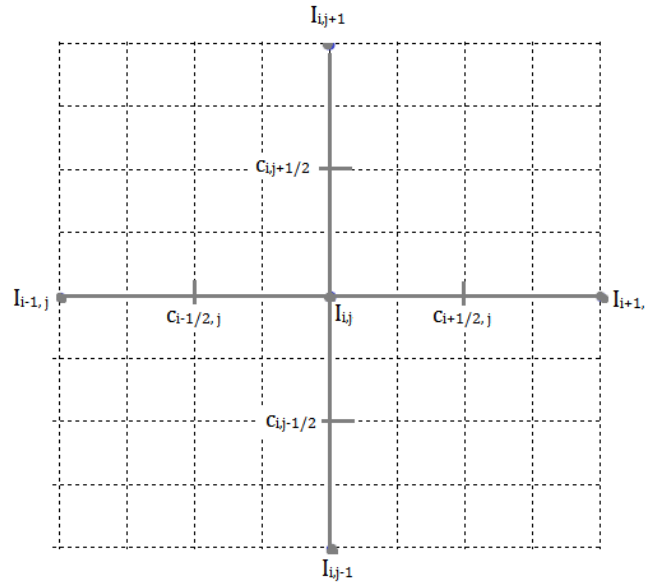


Figure 1: Numerical Discretization Grid

Consider the computational grid shown in figure (1). The intensity (or brightness) values,  $I_{i,j}$  are associated with the nodes of the lattice, North, South, East and West. The diffusion coefficients,  $c_{i,j}$ , are represented midway between lattice

nodes, as shown in figure (1). It may be noted that in by Perona-Malik [1], You-Kaveh [2] and Hajiaboli [3] and related papers, the grid spacing is always considered to be uniform. In this work we will extend this to non-uniform grid spacing.

## 2.1 Explicit Perona-Malik Second Order Difference

Consider the original nonlinear diffusion Perona-Malik equation (3). As previously mentioned, the diffusion coefficient is defined as a function,  $g$ , of the gradient of the intensities of the image, specifically, the norm of the gradient of  $I$ :

$$c(x, y, t) := g(\|\nabla I(x, y, t)\|) \approx c(\cdot) \quad (6)$$

Substituting the shorthand of the diffusion coefficient function (6) back into equation (3), we get:

$$I_t = \frac{\partial I}{\partial t} = \nabla \cdot (c(\cdot) \nabla I) \quad (7)$$

The intensity,  $I(x, y, t)$ , is a 2D function of space and thus the gradient will be with respect to  $x$  and  $y$ . Equation (7) then becomes:

$$I_t = (\partial_x, \partial_y) \cdot (c(\cdot) \partial_x I, c(\cdot) \partial_y I) \quad (8)$$

This simplifies to:

$$I_t = \partial_x(c(\cdot)\partial_x I) + \partial_y(c(\cdot)\partial_y I) \quad (9)$$

The resulting Perona-Malik equation in component form becomes:

$$\frac{\partial I}{\partial t} = \frac{\partial}{\partial x} \left( c(\cdot) \frac{\partial I}{\partial x} \right) + \frac{\partial}{\partial y} \left( c(\cdot) \frac{\partial I}{\partial y} \right) \quad (10)$$

Next, we examine the numerical approximation to each PDE term in equation (10). The term on the left can be approximated using a forward difference in time as follows:

$$I_t = \frac{\partial I}{\partial t} = \frac{I_{i,j}^{n+1} - I_{i,j}^n}{\Delta t} \quad (11)$$

Consider the forward difference for the first term of RHS of (10), in the x-direction:

$$\frac{\partial}{\partial x} \left( c(\cdot) \frac{\partial I}{\partial x} \right) \Big|_{i,j} = \frac{c(\cdot) \frac{\partial I}{\partial x} \Big|_{i+1,j} - c(\cdot) \frac{\partial I}{\partial x} \Big|_{i,j}}{\Delta x} \quad (12)$$

Backward differences are used to calculate the partial derivative of I wrt. X:

$$\frac{\partial I}{\partial x} \Big|_{i+1,j} = \frac{I_{i+1,j} - I_{i,j}}{\Delta x} \quad \text{and} \quad \frac{\partial I}{\partial x} \Big|_{i,j} = \frac{I_{i,j} - I_{i-1,j}}{\Delta x} \quad (13)$$

Substitute the above backward difference approximations of  $\frac{\partial I}{\partial x}$ :

$$\frac{\partial}{\partial x} \left( c(\cdot) \frac{\partial I}{\partial x} \right) = \frac{c(\cdot) \left( \frac{I_{i+1,j} - I_{i,j}}{\Delta x} \right) - c(\cdot) \left( \frac{I_{i,j} - I_{i-1,j}}{\Delta x} \right)}{\Delta x} \quad (14)$$

We approximate the diffusion coefficient,  $c$ , as an average at of the halfway between two nodes, denoted as  $i+1/2$  for  $c$  values between  $i+1$  and  $i$ , and  $i-1/2$  for  $c$  values between  $i$  and  $i-1$ :

$$\frac{\partial}{\partial x} \left( c(\cdot) \frac{\partial I}{\partial x} \right) = \frac{c_{i+\frac{1}{2},j} \left( \frac{I_{i+1,j} - I_{i,j}}{\Delta x} \right) - c_{i-\frac{1}{2},j} \left( \frac{I_{i,j} - I_{i-1,j}}{\Delta x} \right)}{\Delta x} \quad (15)$$

Multiplying out the terms results in

$$\frac{\partial}{\partial x} \left( c(\cdot) \frac{\partial I}{\partial x} \right) = \frac{c_{i+\frac{1}{2},j} I_{i+1,j} - c_{i+\frac{1}{2},j} I_{i,j} - \left[ c_{i-\frac{1}{2},j} I_{i,j} - c_{i-\frac{1}{2},j} I_{i-1,j} \right]}{\Delta x^2} \quad (16)$$

Rewriting (16) with  $c$  terms as coefficients yields:

$$\frac{\partial}{\partial x} \left( c(\cdot) \frac{\partial I}{\partial x} \right) = \frac{c_{i+\frac{1}{2},j} I_{i+1,j} - \left( c_{i+\frac{1}{2},j} + c_{i-\frac{1}{2},j} \right) I_{i,j} + c_{i-\frac{1}{2},j} I_{i-1,j}}{\Delta x^2} \quad (17)$$

Similarly, consider the finite differences in the y direction:

$$\frac{\partial}{\partial y} \left( \frac{c(\cdot) \partial I}{\partial y} \right) = \frac{c_{i,j+\frac{1}{2}} I_{i,j+1} - \left( c_{i,j+\frac{1}{2}} + c_{i,j-\frac{1}{2}} \right) I_{i,j} + c_{i,j-\frac{1}{2}} I_{i,j-1}}{\Delta y^2} \quad (18)$$

Substitute (11), (17), and (18) into equation (10) to get a model for the numerical approximation of the Perona Malik equation. Note that the diffusion coefficient,  $c(\cdot)$ , will always be evaluated at the current timestep,  $n$ :

$$\begin{aligned} \frac{I_{i,j}^{n+1} - I_{i,j}^n}{\Delta t} = & \frac{c_{i+\frac{1}{2},j}^n I_{i+1,j} - \left( c_{i+\frac{1}{2},j}^n + c_{i-\frac{1}{2},j}^n \right) I_{i,j} + c_{i-\frac{1}{2},j}^n I_{i-1,j}}{\Delta x^2} + \\ & \frac{c_{i,j+\frac{1}{2}}^n I_{i,j+1} - \left( c_{i,j+\frac{1}{2}}^n + c_{i,j-\frac{1}{2}}^n \right) I_{i,j} + c_{i,j-\frac{1}{2}}^n I_{i,j-1}}{\Delta y^2} \end{aligned} \quad (19)$$

For a numerical approximation to equation (6) of the diffusion coefficient,  $c(\cdot) = g(\|\nabla I(x, y, t)\|)$ , using the Euclidean (2-)norm, we get:

$$\|\nabla I(x, y, t)\| = \|(I_x, I_y)\|_2 = \sqrt{(I_x, I_y) \cdot (I_x, I_y)} = \sqrt{I_x^2 + I_y^2} \quad (20)$$

Substitute equation (20) into equation (6):

$$c(\cdot) = g\left(\sqrt{I_x^2 + I_y^2}\right) \quad (21)$$

Evaluate  $c(\cdot)$  at the  $(i, j)$  pixel:

$$c_{i,j} = [g(\|\nabla I\|)]_{i,j} = \left[ g\left(\sqrt{I_x^2 + I_y^2}\right) \right]_{i,j} \quad (22)$$

Consider the central differences of the intensity,  $I$ , in the  $x$  and  $y$  directions, using a full step to the next and previous points, which yields denominators of  $2\Delta x (= \Delta x + \Delta x)$  and  $2\Delta y (= \Delta y + \Delta y)$ , respectively:

$$I_x = \frac{\partial I}{\partial x} = \frac{I_{i+1,j} - I_{i-1,j}}{2\Delta x} \quad \text{and} \quad I_y = \frac{\partial I}{\partial y} = \frac{I_{i,j+1} - I_{i,j-1}}{2\Delta y} \quad (23)$$

Substituting the numerical approximations of  $I_x$  and  $I_y$  in (23) into (22) yields:

$$c_{i,j} = \left[ g\left(\sqrt{\left(\frac{I_{i+1,j} - I_{i-1,j}}{2\Delta x}\right)^2 + \left(\frac{I_{i,j+1} - I_{i,j-1}}{2\Delta y}\right)^2}\right) \right]_{i,j} \quad (24)$$

An average is taken to compute the diffusion coefficients at the halfway points:

$$\begin{aligned}
c_{i+\frac{1}{2},j} &= \frac{c_{i,j} + c_{i+1,j}}{2} \\
c_{i-\frac{1}{2},j} &= \frac{c_{i-1,j} + c_{i,j}}{2} \\
c_{i,j+\frac{1}{2}} &= \frac{c_{i,j} + c_{i,j+1}}{2} \\
c_{i,j-\frac{1}{2}} &= \frac{c_{i,j-1} + c_{i,j}}{2}
\end{aligned} \tag{25}$$

Let  $n$  be the superscript that denotes the timestep for the  $I_{i,j}$  terms in the implicit numerical scheme:

$$\begin{aligned}
\frac{I_{i,j}^{n+1} - I_{i,j}^n}{\Delta t} &= \frac{c_{i+\frac{1}{2},j}^n I_{i+1,j}^n - \left( c_{i+\frac{1}{2},j}^n + c_{i-\frac{1}{2},j}^n \right) I_{i,j}^n + c_{i-\frac{1}{2},j}^n I_{i-1,j}^n}{\Delta x^2} + \\
&\quad \frac{c_{i,j+\frac{1}{2}}^n I_{i,j+1}^n - \left( c_{i,j+\frac{1}{2}}^n + c_{i,j-\frac{1}{2}}^n \right) I_{i,j}^n + c_{i,j-\frac{1}{2}}^n I_{i,j-1}^n}{\Delta y^2}
\end{aligned} \tag{26}$$

Rewrite (26), given  $\lambda_1 = \frac{\Delta t}{\Delta x^2}$  and  $\lambda_2 = \frac{\Delta t}{\Delta y^2}$ :

$$\begin{aligned}
I_{i,j}^{n+1} &= \lambda_1 \left( c_{i+\frac{1}{2},j}^n I_{i+1,j}^n + c_{i-\frac{1}{2},j}^n I_{i-1,j}^n \right) + \lambda_2 \left( c_{i,j+\frac{1}{2}}^n I_{i,j+1}^n + c_{i,j-\frac{1}{2}}^n I_{i,j-1}^n \right) + \\
&\quad \left( 1 - \lambda_1 \left( c_{i+\frac{1}{2},j}^n + c_{i-\frac{1}{2},j}^n \right) - \lambda_2 \left( c_{i,j+\frac{1}{2}}^n + c_{i,j-\frac{1}{2}}^n \right) \right) I_{i,j}^n
\end{aligned} \tag{27}$$



## 2.2 Implicit Perona-Malik Second Order Difference:

Let  $(n+1)$  be the superscript that denotes the timestep of the  $I_{i,j}$  terms in (19):

$$\begin{aligned} \frac{I_{i,j}^{n+1} - I_{i,j}^n}{\Delta t} = & \frac{c_{i+\frac{1}{2},j}^n I_{i+1,j}^{n+1} - \left( c_{i+\frac{1}{2},j}^n + c_{i-\frac{1}{2},j}^n \right) I_{i,j}^{n+1} + c_{i-\frac{1}{2},j}^n I_{i-1,j}^{n+1}}{\Delta x^2} + \\ & \frac{c_{i,j+\frac{1}{2}}^n I_{i,j+1}^{n+1} - \left( c_{i,j+\frac{1}{2}}^n + c_{i,j-\frac{1}{2}}^n \right) I_{i,j}^{n+1} + c_{i,j-\frac{1}{2}}^n I_{i,j-1}^{n+1}}{\Delta y^2} \end{aligned} \quad (28)$$

Letting  $\lambda_1 = \frac{\Delta t}{\Delta x^2}$  and  $\lambda_2 = \frac{\Delta t}{\Delta y^2}$ :

$$\begin{aligned} & I_{i,j}^{n+1} \left( 1 + \lambda_1 \left( c_{i+\frac{1}{2},j}^n + c_{i-\frac{1}{2},j}^n \right) + \lambda_2 \left( c_{i,j+\frac{1}{2}}^n + c_{i,j-\frac{1}{2}}^n \right) \right) - \\ & \lambda_1 \left( c_{i+\frac{1}{2},j}^n I_{i+1,j}^{n+1} + c_{i-\frac{1}{2},j}^n I_{i-1,j}^{n+1} \right) - \lambda_2 \left( c_{i,j+\frac{1}{2}}^n I_{i,j+1}^{n+1} + c_{i,j-\frac{1}{2}}^n I_{i,j-1}^{n+1} \right) = I_{i,j}^n \end{aligned} \quad (29)$$

## 2.3 Initial and Boundary Conditions

The initial condition is chosen to be the original image:

$$I_{i,j}^0 = I_0(x_i, y_j) = I(x_i, y_j; t = 0) = \text{Original Image} \quad (30)$$

Neumann Boundary Conditions, wrt x and y, are used to define the values of the intensity and diffusion coefficient c outside the boundaries to be within the same as within the boundary. The Neumann BC imply  $\partial_x = 0$  and  $\partial_y = 0$ , i.e. at the boundaries we get:

$$\partial_x I = \frac{\partial I}{\partial x} = \frac{I_{i+1,j} - I_{i-1,j}}{2\Delta x} = 0 \quad \text{and} \quad \partial_y I = \frac{\partial I}{\partial y} = \frac{I_{i,j+1} - I_{i,j-1}}{2\Delta y} = 0 \quad (31)$$

Let  $i = 0, 1, 2, \dots, I$  and  $j = 0, 1, 2, \dots, J$  denote the step values in the x and y directions, respectively. At the lower boundary of x (i.e.  $i = 0$ ):

$$\left. \frac{\partial I}{\partial x} \right|_{0,j} = \frac{I_{1,j} - I_{-1,j}}{2\Delta x} = 0 \rightarrow I_{-1,j} = I_{1,j} \quad (32)$$

At the upper boundary for x (i.e.  $i = I$ ):

$$\left. \frac{\partial I}{\partial x} \right|_{I,j} = \frac{I_{I+1,j} - I_{I-1,j}}{2\Delta x} = 0 \rightarrow I_{I+1,j} = I_{I-1,j} \quad (33)$$

Similarly, at the lower boundary of y (i.e.  $j = 0$ ):

$$\left. \frac{\partial I}{\partial y} \right|_{i,0} = \frac{I_{i,1} - I_{i,-1}}{2\Delta y} = 0 \rightarrow I_{i,-1} = I_{i,1} \quad (34)$$

At the upper boundary for y (i.e.  $j = J$ ):

$$\left. \frac{\partial I}{\partial y} \right|_{i,J} = \frac{I_{i,J+1} - I_{i,J-1}}{2\Delta y} = 0 \rightarrow I_{i,J+1} = I_{i,J-1} \quad (35)$$

These boundary conditions affect the diffusion coefficients in a similar (25), the boundary conditions are as followed:

$$\begin{aligned} \left. \frac{\partial c}{\partial x} \right|_{0,j} &= \frac{c_{\frac{1}{2},j} - c_{-\frac{1}{2},j}}{\Delta x} = 0 \leftrightarrow c_{-\frac{1}{2},j} = c_{\frac{1}{2},j} \\ \left. \frac{\partial c}{\partial x} \right|_{I,j} &= \frac{c_{I+\frac{1}{2},j} - c_{I-\frac{1}{2},j}}{\Delta x} = 0 \leftrightarrow c_{I+\frac{1}{2},j} = c_{I-\frac{1}{2},j} \\ \left. \frac{\partial c}{\partial y} \right|_{i,0} &= \frac{c_{i,\frac{1}{2}} - c_{i,-\frac{1}{2}}}{\Delta x} = 0 \leftrightarrow c_{i,-\frac{1}{2}} = c_{i,\frac{1}{2}} \\ \left. \frac{\partial c}{\partial y} \right|_{i,J} &= \frac{c_{i,J+\frac{1}{2}} - c_{i,J-\frac{1}{2}}}{\Delta x} = 0 \leftrightarrow c_{i,J+\frac{1}{2}} = c_{i,J-\frac{1}{2}} \end{aligned} \quad (36)$$

### 2.3 Shorthand Notation and Other Numerical Methods

New terminology based on the idea of a central difference operator,  $\delta_p$  is introduced to represent the finite differences in shorthand notation. The shorthand notation for the finites difference of the Perona-Malik equation in the x-direction:

$$\delta_x^2(cI)_{i,j} = \frac{c_{i+\frac{1}{2},j}^n I_{i+1,j} - \left( c_{i+\frac{1}{2},j}^n + c_{i-\frac{1}{2},j}^n \right) I_{i,j} + c_{i-\frac{1}{2},j}^n I_{i-1,j}}{\Delta x^2} \quad (37)$$

and in the y-direction,

$$\delta_y^2(cI)_{i,j} = \frac{c_{i,j+\frac{1}{2}}^n I_{i,j+1} - \left( c_{i,j+\frac{1}{2}}^n + c_{i,j-\frac{1}{2}}^n \right) I_{i,j} + c_{i,j-\frac{1}{2}}^n I_{i,j-1}}{\Delta y^2} \quad (38)$$

With this notation, equations (27) and (29), the explicit and implicit numerical schemes for the Perona Malik Equation, become:

$$\frac{I_{i,j}^{n+1} - I_{i,j}^n}{\Delta t} = \delta_x^2(cI)_{i,j}^n + \delta_y^2(cI)_{i,j}^n \quad (39)$$

$$\frac{I_{i,j}^{n+1} - I_{i,j}^n}{\Delta t} = \delta_x^2(cI)_{i,j}^{n+1} + \delta_y^2(cI)_{i,j}^{n+1} \quad (40)$$

The shorthand notation can also be used to denote other numerical schemes which are not investigated in this paper, but may offer promising results if investigated in the future. The Crank-Nicolson numerical scheme of the Perona-Malik equation is essentially an average of the explicit and implicit schemes:

$$\frac{I_{i,j}^{n+1} - I_{i,j}^n}{\Delta t} = \frac{1}{2} \left( \delta_x^2(c) + \delta_y^2(c) \right) (I_{i,j}^{n+1} + I_{i,j}^n) \quad (41)$$

Note that  $\delta_x^2(c)I_{i,j} = \delta_x^2(cI)_{i,j}$ .

The Alternating Direction Implicit (ADI) Method, commonly used in petroleum manufacturing, allows the implementation of the numerical scheme to be computed in half-time-steps, and solving in only one direction, x or y, at a time.

When in the x-direction, solve for the  $I_{i,j}^{n+\frac{1}{2}}$  and when in the y direction, solve for  $I_{i,j}^{n+\frac{1}{2}}$  using the solution from the x-direction. So the ADI numerical scheme of the Perona-Malik equation can be represented as:

$$\begin{aligned} \frac{I_{i,j}^{n+\frac{1}{2}} - I_{i,j}^n}{\frac{\Delta t}{2}} &= \delta_x^2(\text{cl})_{i,j}^{n+\frac{1}{2}} + \delta_y^2(\text{cl})_{i,j}^n \\ \frac{I_{i,j}^{n+1} - I_{i,j}^{n+\frac{1}{2}}}{\frac{\Delta t}{2}} &= \delta_x^2(\text{cl})_{i,j}^{n+\frac{1}{2}} + \delta_y^2(\text{cl})_{i,j}^{n+1} \end{aligned} \quad (42)$$

## CHAPTER III – STABILITY AND CONVERGENCE

In this chapter we will discuss the stability of the explicit and implicit finite difference schemes to approximate the Perona-Malik equation numerically. The discussion is presented in one dimension (only with respect to  $x$ ), but can be directly extended to two-dimensions yielding similar results.

### 3.1 Stability Analysis of the Explicit Numerical Scheme

We start the discussion of the explicit numerical scheme for the Perona-Malik function by rewriting the Perona-Malik anisotropic diffusion equation in 1-D:

$$I_t = (c(I_x)I_x)_x \leftrightarrow \frac{\partial I}{\partial t} = \frac{\partial}{\partial x} \left( c \left( \frac{\partial I}{\partial x} \right) \frac{\partial I}{\partial x} \right) \quad (43)$$

The explicit scheme derived in Section 2.1 (27) can be rewritten as followed:

$$\frac{I_i^{n+1} - I_i^n}{\Delta t} = \frac{c_{i+\frac{1}{2}}^n \left( \frac{I_{i+1}^n - I_i^n}{\Delta x} \right) - c_{i-\frac{1}{2}}^n \left( \frac{I_i^n - I_{i-1}^n}{\Delta x} \right)}{\Delta x} \quad (44)$$

The above finite difference has an initial condition of

$$I_i^0 = I_0(x_i) \quad (45)$$

Define a dot product with respect to  $\Delta x$  of two N-dimensional vectors A and B to be

$$(A, B)_{\Delta x} = \frac{\Delta x}{2} A_0 B_0 + \sum_{i=1}^{N-1} \Delta x A_i B_i + \frac{\Delta x}{2} A_N B_N \quad (46)$$

From this definition of the dot product, we directly define a norm with respect to  $\Delta x$ , conventionally as follows:

$$\|A\|_{\Delta x} = (A, A)_{\Delta x}^{\frac{1}{2}} \quad (47)$$

We now prove the following theorem:

*Theorem 3.1:* *Let  $I_i^n$  be the numerical solution to equation (44). If  $\Delta t \leq \frac{(\Delta x)^2}{2 c_{\max}}$  where  $\Delta t$  and  $\Delta x$  are the respective temporal and spatial step sizes and  $c_{\max} = \max_{i,n} c_i^n$  we then have:*

$$\|I^{n+1}\|_{\Delta x} \leq \|I^n\|_{\Delta x}$$

To prove the theorem, we need the following lemmas:

*Lemma 3.1.1*  $\left(\frac{I^{n+1}-I^n}{\Delta t}, I^n\right)_{\Delta x} + \sum_{i=1}^N \Delta x c_{i-\frac{1}{2}}^n \left(\frac{I_i^n - I_{i-1}^n}{\Delta x}\right)^2 = 0$

*Proof of Lemma 3.1.1.* Multiply both sides of equation (44) by  $I_{i,j}^n$ :

$$\left(\frac{I_i^{n+1}-I_i^n}{\Delta t}\right) I_i^n = \left[ \frac{c_{i+\frac{1}{2}}^n \left(\frac{I_{i+1}^n - I_i^n}{\Delta x}\right) - c_{i-\frac{1}{2}}^n \left(\frac{I_i^n - I_{i-1}^n}{\Delta x}\right)}{\Delta x} \right] I_i^n \quad (48)$$

for  $i=0,1,\dots,N$ . These yield the following system:

$$\begin{aligned} \left(\frac{I_0^{n+1}-I_0^n}{\Delta t}\right) I_0^n &= \left[ \frac{c_{\frac{1}{2}}^n \left(\frac{I_1^n - I_0^n}{\Delta x}\right) - c_{-\frac{1}{2}}^n \left(\frac{I_0^n - I_{-1}^n}{\Delta x}\right)}{\Delta x} \right] I_0^n \\ \left(\frac{I_1^{n+1}-I_1^n}{\Delta t}\right) I_1^n &= \left[ \frac{c_{\frac{3}{2}}^n \left(\frac{I_2^n - I_1^n}{\Delta x}\right) - c_{\frac{1}{2}}^n \left(\frac{I_1^n - I_0^n}{\Delta x}\right)}{\Delta x} \right] I_1^n \\ &\vdots \quad \quad \quad \vdots \quad \quad \quad \vdots \quad \quad \quad \vdots \\ \left(\frac{I_{N-1}^{n+1}-I_{N-1}^n}{\Delta t}\right) I_{N-1}^n &= \left[ \frac{c_{N-\frac{1}{2}}^n \left(\frac{I_N^n - I_{N-1}^n}{\Delta x}\right) - c_{N-\frac{3}{2}}^n \left(\frac{I_{N-1}^n - I_{N-2}^n}{\Delta x}\right)}{\Delta x} \right] I_{N-1}^n \\ \left(\frac{I_N^{n+1}-I_N^n}{\Delta t}\right) I_N^n &= \left[ \frac{c_{N+\frac{1}{2}}^n \left(\frac{I_{N+1}^n - I_N^n}{\Delta x}\right) - c_{N-\frac{1}{2}}^n \left(\frac{I_N^n - I_{N-1}^n}{\Delta x}\right)}{\Delta x} \right] I_N^n \end{aligned} \quad (49)$$



Using the definition of the dot product, the terms on the left of (47) can be combined to yield:

$$\frac{\Delta x}{2} \left( \frac{I_0^{n+1} - I_0^n}{\Delta t} \right) I_0^n + \sum_{i=1}^{N-1} \Delta x \left( \frac{I_i^{n+1} - I_i^n}{\Delta t} \right) I_i^n + \frac{\Delta x}{2} \left( \frac{I_N^{n+1} - I_N^n}{\Delta t} \right) I_N^n = \left( \frac{I^{n+1} - I^n}{\Delta t}, I^n \right)_{\Delta x} \quad (50)$$

In a similar fashion, adding the terms of RHS of (47) we get:

$$\begin{aligned} & \frac{1}{2} \left[ c_{\frac{1}{2}}^n \left( \frac{I_1^n - I_0^n}{\Delta x} \right) - c_{-\frac{1}{2}}^n \left( \frac{I_0^n - I_{-1}^n}{\Delta x} \right) \right] I_0^n \\ & + \left[ c_{\frac{3}{2}}^n \left( \frac{I_2^n - I_1^n}{\Delta x} \right) - c_{\frac{1}{2}}^n \left( \frac{I_1^n - I_0^n}{\Delta x} \right) \right] I_1^n + \dots \dots \dots \\ & + \left[ c_{N-\frac{1}{2}}^n \left( \frac{I_N^n - I_{N-1}^n}{\Delta x} \right) - c_{N-\frac{3}{2}}^n \left( \frac{I_{N-1}^n - I_{N-2}^n}{\Delta x} \right) \right] I_{N-1}^n \\ & + \frac{1}{2} \left[ c_{N+\frac{1}{2}}^n \left( \frac{I_{N+1}^n - I_N^n}{\Delta x} \right) - c_{N-\frac{1}{2}}^n \left( \frac{I_N^n - I_{N-1}^n}{\Delta x} \right) \right] I_N^n \end{aligned} \quad (51)$$

Recall the boundary conditions from Section 2.3, but rewritten for the 1-D case wrt  $x$  only, i.e.  $i=0,1,\dots,N$ . Note that in the 1D case we have BC that affect the first term (i.e. when  $i=0$ ):

$$c_{-\frac{1}{2}} = c_{\frac{1}{2}} \quad \text{and} \quad I_{-1} = I_1 \quad (52)$$

Substituting (52) into the first term of (51) yields:

$$\frac{1}{2} \left[ c_{\frac{1}{2}}^n \left( \frac{I_1^n - I_0^n}{\Delta x} \right) - c_{-\frac{1}{2}}^n \left( \frac{I_0^n - I_1^n}{\Delta x} \right) \right] I_0^n = c_{\frac{1}{2}}^n \left( \frac{I_1^n - I_0^n}{\Delta x} \right) I_0^n \quad (53)$$

Also, note that in the 1D case we have BC that affect the last term (i.e. when  $i=N$ ):

$$c_{N+\frac{1}{2}} = c_{N-\frac{1}{2}} \quad \text{and} \quad I_{N+1} = I_{N-1} \quad (54)$$

Again, substituting (54) into the last term of (51), yields:

$$\frac{1}{2} \left[ c_{N-\frac{1}{2}}^n \left( \frac{I_{N-1}^n - I_N^n}{\Delta x} \right) - c_{N+\frac{1}{2}}^n \left( \frac{I_N^n - I_{N-1}^n}{\Delta x} \right) \right] I_N^n = -c_{N-\frac{1}{2}}^n \left( \frac{I_N^n - I_{N-1}^n}{\Delta x} \right) I_N^n \quad (55)$$

Substituting (53) and (55) into (51), we get:

$$\left( \frac{I^{n+1} - I^n}{\Delta t}, I^n \right)_{\Delta x} =$$

$$-c_{\frac{1}{2}}^n \left( \frac{I_1^n - I_0^n}{\Delta x} \right) I_0^n$$

$$\begin{aligned}
& -c_{\frac{3}{2}}^n \left( \frac{I_2^n - I_1^n}{\Delta x} \right) I_1^n - c_{\frac{1}{2}}^n \left( \frac{I_1^n - I_0^n}{\Delta x} \right) I_1^n \\
& -c_{\frac{5}{2}}^n \left( \frac{I_3^n - I_2^n}{\Delta x} \right) I_2^n - c_{\frac{3}{2}}^n \left( \frac{I_2^n - I_1^n}{\Delta x} \right) I_1^n \\
& \quad \vdots \quad \quad \quad \vdots \\
& -c_{N-\frac{3}{2}}^n \left( \frac{I_{N-1}^n - I_{N-2}^n}{\Delta x} \right) I_{N-2}^n - c_{N-\frac{5}{2}}^n \left( \frac{I_{N-2}^n - I_{N-3}^n}{\Delta x} \right) I_{N-2}^n \\
& -c_{N-\frac{1}{2}}^n \left( \frac{I_{N-1}^n - I_{N-2}^n}{\Delta x} \right) I_{N-1}^n - c_{N-\frac{3}{2}}^n \left( \frac{I_{N-1}^n - I_{N-2}^n}{\Delta x} \right) I_{N-1}^n \\
& -c_{N-\frac{1}{2}}^n \left( \frac{I_{N-1}^n - I_{N-2}^n}{\Delta x} \right) I_{N-1}^n
\end{aligned} \tag{56}$$

Collecting similar difference terms  $\left( \frac{I_i^n - I_{i-1}^n}{\Delta x} \right)$ , we get:

$$\begin{aligned}
& \left( \frac{I^{n+1} - I^n}{\Delta t}, I^n \right)_{\Delta x} = \\
& -c_{\frac{1}{2}}^n (I_1^n - I_0^n) \left( \frac{I_1^n - I_0^n}{\Delta x} \right) \\
& -c_{\frac{3}{2}}^n (I_2^n - I_1^n) \left( \frac{I_2^n - I_1^n}{\Delta x} \right) \\
& -c_{\frac{5}{2}}^n (I_3^n - I_2^n) \left( \frac{I_3^n - I_2^n}{\Delta x} \right) \\
& \quad \vdots \\
& -c_{N-\frac{3}{2}}^n (I_{N-1}^n - I_{N-2}^n) \left( \frac{I_{N-1}^n - I_{N-2}^n}{\Delta x} \right) \\
& -c_{N-\frac{1}{2}}^n (I_N^n - I_{N-1}^n) \left( \frac{I_N^n - I_{N-1}^n}{\Delta x} \right)
\end{aligned} \tag{57}$$

It is obvious that a repeated pattern (series) is revealed above. We multiply every term by  $\frac{\Delta x}{\Delta t}$  and collect the terms to result in a summation on the RHS. Thus

equation (56) is consolidated to the following:

$$\left(\frac{I^{n+1}-I^n}{\Delta t}, I^n\right)_{\Delta x} = -\sum_{i=1}^N \Delta x c_{i-\frac{1}{2}}^n \left(\frac{I_i^n - I_{i-1}^n}{\Delta x}\right) \left(\frac{I_i^n - I_{i-1}^n}{\Delta x}\right) \quad (58)$$

Which proves the result:

$$\left(\frac{I^{n+1}-I^n}{\Delta t}, I^n\right)_{\Delta x} + \sum_{i=1}^N \Delta x c_{i-\frac{1}{2}}^n \left(\frac{I_i^n - I_{i-1}^n}{\Delta x}\right)^2 = 0 \quad \square$$

$$\underline{\text{Lemma 3.1.2:}} \quad \left(\frac{I^{n+1}-I^n}{\Delta t}, I^n\right)_{\Delta x} = -\frac{\Delta t}{2} \left\| \frac{I^{n+1}-I^n}{\Delta t} \right\|_{\Delta x}^2 + \frac{\|I^{n+1}\|_{\Delta x}^2 - \|I^n\|_{\Delta x}^2}{2\Delta t}$$

*Proof.* Consider rewriting  $I^n$  as an expression of itself,  $I^{n+1}$ , and  $\Delta t$ :

$$I^n = \Delta t \left(-\frac{1}{2}\right) \left(\frac{I^{n+1}-I^n}{\Delta t}\right) + \frac{I^{n+1}+I^n}{2} \quad (59)$$

Substituting (60) into (58), we get:

$$\left(\frac{I^{n+1}-I^n}{\Delta t}, I^n\right)_{\Delta x} = \left(\frac{I^{n+1}-I^n}{\Delta t}, \Delta t \left(-\frac{1}{2}\right) \left(\frac{I^{n+1}-I^n}{\Delta t}\right) + \frac{I^{n+1}+I^n}{2}\right)_{\Delta x} \quad (60)$$

It is possible to move the scalar values outside dot product and use distribution:

$$\left(\frac{I^{n+1}-I^n}{\Delta t}, I^n\right)_{\Delta x} = \Delta t \left(-\frac{1}{2}\right) \left(\frac{I^{n+1}-I^n}{\Delta t}, \frac{I^{n+1}-I^n}{\Delta t}\right)_{\Delta x} + \left(\frac{I^{n+1}-I^n}{\Delta t}, \frac{I^{n+1}+I^n}{2}\right)_{\Delta x} \quad (61)$$

The first term on the RHS of (62) is the norm, according to the norm definition (49).

Again, scalar values can be extracted to the outside of the dot product in the second term above:

$$\left(\frac{I^{n+1}-I^n}{\Delta t}, I^n\right)_{\Delta x} = -\frac{\Delta t}{2} \left\| \frac{I^{n+1}-I^n}{\Delta t} \right\|_{\Delta x}^2 + \frac{1}{2\Delta t} (I^{n+1} - I^n, I^{n+1} + I^n)_{\Delta x} \quad (62)$$

Rewrite the dot product in the last term above as:

$$\begin{aligned} (I^{n+1} - I^n, I^{n+1} + I^n)_{\Delta x} &= (I^{n+1}, I^{n+1})_{\Delta x} + (I^{n+1}, I^n)_{\Delta x} - \\ &\quad (I^n, I^{n+1})_{\Delta x} - (I^n, I^n)_{\Delta x} \end{aligned} \quad (63)$$

The interior terms of (64) cancel out. Observe that the first and last terms are the square of the norms. Then (64) becomes:

$$\left(\frac{I^{n+1}-I^n}{\Delta t}, I^n\right)_{\Delta x} = -\frac{\Delta t}{2} \left\| \frac{I^{n+1}-I^n}{\Delta t} \right\|_{\Delta x}^2 + \frac{\|I^{n+1}\|_{\Delta x}^2 - \|I^n\|_{\Delta x}^2}{2\Delta t} \quad \square$$

Lemma 3.1.3  $\left\| \frac{I_N^{n+1} - I_N^n}{\Delta t} \right\|_{\Delta x}^2 \leq \frac{4}{(\Delta x)^2} c_{max} \sum_{i \geq 1}^N \Delta x c_{i-\frac{1}{2}}^n \left( \frac{I_i^n - I_{i-1}^n}{\Delta x} \right)^2$

*Proof.* Recall the explicit Perona-Malik numerical scheme for equation (44). At  $i=0$ , given the boundary conditions  $c_{-\frac{1}{2}} = c_{\frac{1}{2}}$  and  $I_{-1} = I_1$ . Equation (44) becomes:

$$\frac{I_0^{n+1} - I_0^n}{\Delta t} = \frac{c_{\frac{1}{2}}^n \left( \frac{I_1^n - I_0^n}{\Delta x} \right) - c_{-\frac{1}{2}}^n \left( \frac{I_0^n - I_{-1}^n}{\Delta x} \right)}{\Delta x} = \frac{2}{\Delta x} c_{\frac{1}{2}}^n \left( \frac{I_1^n - I_0^n}{\Delta x} \right) \quad (64)$$

When  $i=N$  and given the boundary conditions that  $c_{N+\frac{1}{2}} = c_{N-\frac{1}{2}}$  and  $I_{N+1} = I_{N-1}$

$$\frac{I_N^{n+1} - I_N^n}{\Delta t} = \frac{c_{N-\frac{1}{2}}^n \left( \frac{I_{N+1}^n - I_N^n}{\Delta x} \right) - c_{N+\frac{1}{2}}^n \left( \frac{I_N^n - I_{N-1}^n}{\Delta x} \right)}{\Delta x} = -\frac{2}{\Delta x} c_{N-\frac{1}{2}}^n \left( \frac{I_N^n - I_{N-1}^n}{\Delta x} \right) \quad (65)$$

The square of the defined  $\Delta x$ -norm is equal to the dot product of the vector with itself. So in the case of the finite difference wrt  $t$ , we have

$$\left\| \frac{I_N^{n+1} - I_N^n}{\Delta t} \right\|_{\Delta x}^2 = \left( \frac{I_N^{n+1} - I_N^n}{\Delta t}, \frac{I_N^{n+1} - I_N^n}{\Delta t} \right)_{\Delta x} \quad (66)$$

Incorporating the first and last and interior terms we can define the square of the  $\Delta x$ -norm of the finite difference wrt. t as followed

$$\left\| \frac{I_N^{n+1} - I_N^n}{\Delta t} \right\|_{\Delta x}^2 = \frac{\Delta x}{2} \left( \frac{I_0^{n+1} - I_0^n}{\Delta t} \right)^2 + \Delta x \left( \frac{I_1^{n+1} - I_1^n}{\Delta t} \right)^2 + \dots + \Delta x \left( \frac{I_{N-1}^{n+1} - I_{N-1}^n}{\Delta t} \right)^2 + \frac{\Delta x}{2} \left( \frac{I_N^{n+1} - I_N^n}{\Delta t} \right)^2 \quad (67)$$

Substitute the RHS of equations (68) and (69) that include BC and the explicit scheme RHS for each interior point (i=1,...,N-1) into the above equation (71):

$$\begin{aligned}
& \left\| \frac{I_N^{n+1} - I_N^n}{\Delta t} \right\|_{\Delta x}^2 \\
&= \frac{\Delta x}{2} \left( \frac{2}{\Delta x} c_{\frac{1}{2}}^n \left( \frac{I_1^n - I_0^n}{\Delta x} \right) \right)^2 + \Delta x \left( \frac{c_{\frac{3}{2}}^n \left( \frac{I_2^n - I_1^n}{\Delta x} \right) - c_{\frac{1}{2}}^n \left( \frac{I_1^n - I_0^n}{\Delta x} \right)}{\Delta x} \right)^2 \\
&+ \Delta x \left( \frac{c_{\frac{5}{2}}^n \left( \frac{I_3^n - I_2^n}{\Delta x} \right) - c_{\frac{3}{2}}^n \left( \frac{I_2^n - I_1^n}{\Delta x} \right)}{\Delta x} \right)^2 + \dots \\
&+ \Delta x \left( \frac{c_{N-\frac{3}{2}}^n \left( \frac{I_{N-1}^n - I_{N-2}^n}{\Delta x} \right) - c_{N-\frac{5}{2}}^n \left( \frac{I_{N-2}^n - I_{N-3}^n}{\Delta x} \right)}{\Delta x} \right)^2 \\
&+ \Delta x \left( \frac{c_{N-\frac{1}{2}}^n \left( \frac{I_N^n - I_{N-1}^n}{\Delta x} \right) - c_{N-\frac{3}{2}}^n \left( \frac{I_{N-1}^n - I_{N-2}^n}{\Delta x} \right)}{\Delta x} \right)^2 \\
&+ \frac{\Delta x}{2} \left( -\frac{2}{\Delta x} c_{N-\frac{1}{2}}^n \left( \frac{I_N^n - I_{N-1}^n}{\Delta x} \right) \right)^2
\end{aligned} \tag{68}$$

Cancelling out the  $\Delta x$  where necessary, we end up with a sum of squares of differences, i.e.  $\left( (..) - (..) \right)^2$ , except for first and last terms, which were expanded:



$$\begin{aligned}
& \left\| \frac{I_N^{n+1} - I_N^n}{\Delta t} \right\|_{\Delta x}^2 = \\
& \frac{\Delta x}{2} \left( \frac{4}{(\Delta x)^2} \left( c_{\frac{1}{2}}^n \right)^2 \left( \frac{I_1^n - I_0^n}{\Delta x} \right)^2 \right) + \frac{1}{\Delta x} \left( c_{\frac{3}{2}}^n \left( \frac{I_2^n - I_1^n}{\Delta x} \right) - c_{\frac{1}{2}}^n \left( \frac{I_1^n - I_0^n}{\Delta x} \right) \right)^2 + \\
& \frac{1}{\Delta x} \left( c_{\frac{5}{2}}^n \left( \frac{I_3^n - I_2^n}{\Delta x} \right) - c_{\frac{3}{2}}^n \left( \frac{I_2^n - I_1^n}{\Delta x} \right) \right)^2 + \dots + \frac{1}{\Delta x} \left( c_{N-\frac{3}{2}}^n \left( \frac{I_{N-1}^n - I_{N-2}^n}{\Delta x} \right) - \right. \\
& \left. c_{N-\frac{5}{2}}^n \left( \frac{I_{N-2}^n - I_{N-3}^n}{\Delta x} \right) \right)^2 + \frac{1}{\Delta x} \left( c_{N-\frac{1}{2}}^n \left( \frac{I_N^n - I_{N-1}^n}{\Delta x} \right) - c_{N-\frac{3}{2}}^n \left( \frac{I_{N-1}^n - I_{N-2}^n}{\Delta x} \right) \right)^2 + \frac{\Delta x}{2} \quad (69)
\end{aligned}$$

*Claim:*  $(x - y)^2 \leq 2x^2 + 2y^2$  .

*Proof of Claim.*  $(x - y)^2 = x^2 - 2xy + y^2 \leq 2x^2 + 2y^2$

$$\Leftrightarrow x^2 + y^2 \leq 2x^2 + 2xy + 2y^2$$

$$\Leftrightarrow 0 \leq x^2 + 2xy + y^2$$

$$\Leftrightarrow 0 \leq (x + y)^2$$

$$(x - y)^2 \leq 2x^2 + 2y^2$$

□

By the claim, we can employ this inequality on the RHS of (71). Multiplication of the interior terms by  $\frac{\Delta x}{\Delta x}$  yields:

$$\begin{aligned}
\left\| \frac{I_N^{n+1} - I_N^n}{\Delta t} \right\|_{\Delta x}^2 &= \frac{\Delta x}{(\Delta x)^2} \left( 2 \left( c_{\frac{1}{2}}^n \right)^2 \left( \frac{I_1^n - I_0^n}{\Delta x} \right)^2 \right) + \frac{\Delta x}{(\Delta x)^2} \left( 2 \left( c_{\frac{3}{2}}^n \right)^2 \left( \frac{I_2^n - I_1^n}{\Delta x} \right)^2 + \right. \\
&2 \left( c_{\frac{1}{2}}^n \right)^2 \left( \frac{I_1^n - I_0^n}{\Delta x} \right)^2 \left. \right) + \frac{\Delta x}{(\Delta x)^2} \left( 2 \left( c_{\frac{5}{2}}^n \right)^2 \left( \frac{I_3^n - I_2^n}{\Delta x} \right)^2 + 2 \left( c_{\frac{3}{2}}^n \right)^2 \left( \frac{I_2^n - I_1^n}{\Delta x} \right)^2 \right) + \\
&\dots + \frac{\Delta x}{(\Delta x)^2} \left( 2 \left( c_{N-\frac{3}{2}}^n \right)^2 \left( \frac{I_{N-1}^n - I_{N-2}^n}{\Delta x} \right)^2 + 2 \left( c_{N-\frac{5}{2}}^n \right)^2 \left( \frac{I_{N-2}^n - I_{N-3}^n}{\Delta x} \right)^2 \right) + \\
&\frac{\Delta x}{(\Delta x)^2} \left( 2 \left( c_{N-\frac{1}{2}}^n \right)^2 \left( \frac{I_N^n - I_{N-1}^n}{\Delta x} \right)^2 + 2 \left( c_{N-\frac{3}{2}}^n \right)^2 \left( \frac{I_{N-1}^n - I_{N-2}^n}{\Delta x} \right)^2 \right) + \\
&\frac{\Delta x}{(\Delta x)^2} \left( 2 \left( c_{N-\frac{1}{2}}^n \right)^2 \left( \frac{I_N^n - I_{N-1}^n}{\Delta x} \right)^2 \right) \tag{70}
\end{aligned}$$

The RHS above collapses into a sum of terms with a repeated pattern that can thus be consolidated into a single summation term (after adding like terms):

$$\left\| \frac{I_N^{n+1} - I_N^n}{\Delta t} \right\|_{\Delta x}^2 = \frac{4}{(\Delta x)^2} \sum_{i=1}^N \Delta x \left( c_{i-\frac{1}{2}}^n \right)^2 \left( \frac{I_i^n - I_{i-1}^n}{\Delta x} \right)^2 \tag{71}$$

Select the maximum value of the  $c$  function and remove from the product, defined

$c_{max}$  to be defined as  $c_{max} := \max_i \left( c_{i-\frac{1}{2}}^n \right)$  to introduce an inequality:

$$\left\| \frac{I_N^{n+1} - I_N^n}{\Delta t} \right\|_{\Delta x}^2 \leq \frac{4}{(\Delta x)^2} c_{max} \sum_{i=1}^N \Delta x c_{i-\frac{1}{2}}^n \left( \frac{I_i^n - I_{i-1}^n}{\Delta x} \right)^2 \quad \square$$

*Proof of Theorem 3.1.*

Using Lemmas 3.1 and 3.2, we get,

$$-\frac{\Delta t}{2} \left\| \frac{I^{n+1} - I^n}{\Delta t} \right\|_{\Delta x}^2 + \frac{\|I^{n+1}\|_{\Delta x}^2 - \|I^n\|_{\Delta x}^2}{2\Delta t} + \sum_{i=1}^N \Delta x c_{i-\frac{1}{2}}^n \left( \frac{I_i^n - I_{i-1}^n}{\Delta x} \right)^2 = 0 \quad (72)$$

Adding the first term on the LHS to the RHS in (64) removes the negative sign:

$$\frac{\|I^{n+1}\|_{\Delta x}^2 - \|I^n\|_{\Delta x}^2}{2\Delta t} + \sum_{i=1}^N \Delta x c_{i-\frac{1}{2}}^n \left( \frac{I_i^n - I_{i-1}^n}{\Delta x} \right)^2 = \frac{\Delta t}{2} \left\| \frac{I^{n+1} - I^n}{\Delta t} \right\|_{\Delta x}^2 \quad (73)$$

Using Lemma 3.3 for the RHS of (75), we get:

$$\begin{aligned} \frac{\|I^{n+1}\|_{\Delta x}^2 - \|I^n\|_{\Delta x}^2}{2\Delta t} + \sum_{i=1}^N \Delta x c_{i-\frac{1}{2}}^n \left( \frac{I_i^n - I_{i-1}^n}{\Delta x} \right)^2 \leq \\ \frac{\Delta t}{2} \left[ \frac{4}{(\Delta x)^2} c_{max} \sum_{i \geq 1}^N \Delta x c_{i-\frac{1}{2}}^n \left( \frac{I_i^n - I_{i-1}^n}{\Delta x} \right)^2 \right] \end{aligned} \quad (74)$$

We can then collect the coefficients for the summation on the LHS:

$$\frac{\|I^{n+1}\|_{\Delta x}^2 - \|I^n\|_{\Delta x}^2}{2\Delta t} + \left( 1 - \frac{2\Delta t}{(\Delta x)^2} c_{max} \right) \sum_{i=1}^N \Delta x c_{i-\frac{1}{2}}^n \left( \frac{I_i^n - I_{i-1}^n}{\Delta x} \right)^2 \leq 0 \quad (75)$$

An assumption made by Perona and Malik in their 1990 paper, is that the choice of the  $g$  function that describes the diffusion coefficient,  $c$ , is restricted to a domain of  $[0,1]$ . This special choice of the  $g$  function forces the diffusion coefficient,  $c$ , to remain within this domain. Then since  $\Delta x > 0$  and the square of the difference term is greater than or equal to 0, the summation becomes positive. For the above equation to hold true, given the second term is positive, the first term is forced to be less than or equal to 0, which implies

$$\frac{\|I^{n+1}\|_{\Delta x}^2 - \|I^n\|_{\Delta x}^2}{2\Delta t} \leq 0 \quad (76)$$

Cross-multiplying the  $2\Delta t$  by zero gives:

$$\|I^{n+1}\|_{\Delta x}^2 - \|I^n\|_{\Delta x}^2 \leq 0 \quad \leftrightarrow \quad \|I^{n+1}\|_{\Delta x}^2 \leq \|I^n\|_{\Delta x}^2 \quad \square$$

*Remark.* The proof of Theorem 3.1 proves stability, i.e. the  $(n+1)$ -step is bounded by the  $n$ -step, *but with a condition on  $\Delta t$* . Hence the explicit numerical scheme for the Perona-Malik equation in 1D (and in higher dimensions) is stable if and only if the following stability condition holds for  $\Delta t$ ,  $\Delta x$ , and  $c_{max}$ :

$$1 - \frac{2\Delta t}{(\Delta x)^2} c_{max} \geq 0$$

$$\leftrightarrow \frac{2\Delta t}{(\Delta x)^2} c_{max} \leq 1$$

$$\leftrightarrow \Delta t \leq \frac{(\Delta x)^2}{2 c_{max}} \quad (77)$$

When extended to the 2D case, the stability condition on  $\Delta t$  becomes:

$$\Delta t \leq \frac{(\Delta x)^2}{2 c_{\max (i)}} + \frac{(\Delta y)^2}{2 c_{\max (j)}} \quad (78)$$

When dealing with the 2D case when the diffusion coefficient is  $c=1$  or when the maximum value of  $c$  is  $c_{\max (i)} = c_{\max (j)} = 1$ , then the stability condition on  $\Delta t$  becomes:

$$\Delta t \leq \frac{(\Delta x)^2 + (\Delta y)^2}{2} \quad (79)$$

### 3.2 Stability Analysis of the Implicit Numerical Scheme

To examine the stability of the implicit numerical scheme of the Perona-Malik equation, we consider the following implicit discretization:

$$\frac{I_i^{n+1} - I_i^n}{\Delta t} = \frac{c_{i+\frac{1}{2}}^n \left( \frac{I_{i+1}^{n+1} - I_i^{n+1}}{\Delta x} \right) - c_{i-\frac{1}{2}}^n \left( \frac{I_i^{n+1} - I_{i-1}^{n+1}}{\Delta x} \right)}{\Delta x} \quad (80)$$

The above finite difference is considered with an initial condition of  $I_i^0 = I_0(x_i)$ .

Theorem 3.2: The scheme represented in (80) is unconditionally stable.

Moreover:

$$\|I^{n+1}\|_{\Delta x} \leq \|I^n\|_{\Delta x}$$

To prove the theorem, we need the following lemmas:

Lemma 3.2.1  $\left(\frac{I^{n+1}-I^n}{\Delta t}, I^{n+1}\right)_{\Delta x} + \sum_{i=1}^N \Delta x c_{i-\frac{1}{2}}^n \left(\frac{I_i^{n+1}-I_{i-1}^{n+1}}{\Delta x}\right)^2 = 0$

*Proof.* Multiply both sides of equation (80) by  $I_{i,j}^{n+1}$ :

$$\left(\frac{I_i^{n+1}-I_i^n}{\Delta t}\right) I_i^{n+1} = \left[ \frac{c_{i+\frac{1}{2}}^n \left(\frac{I_{i+1}^{n+1}-I_i^{n+1}}{\Delta x}\right) - c_{i-\frac{1}{2}}^n \left(\frac{I_i^{n+1}-I_{i-1}^{n+1}}{\Delta x}\right)}{\Delta x} \right] I_i^{n+1} \quad (81)$$

For  $i=0,1,\dots,N$  to generate the following list of equations:

$$\begin{aligned} \left(\frac{I_0^{n+1}-I_0^n}{\Delta t}\right) I_0^{n+1} &= \left[ \frac{c_{\frac{1}{2}}^n \left(\frac{I_1^{n+1}-I_0^{n+1}}{\Delta x}\right) - c_{-\frac{1}{2}}^n \left(\frac{I_0^{n+1}-I_{-1}^{n+1}}{\Delta x}\right)}{\Delta x} \right] I_0^{n+1} \\ \left(\frac{I_1^{n+1}-I_1^n}{\Delta t}\right) I_1^{n+1} &= \left[ \frac{c_{\frac{3}{2}}^n \left(\frac{I_2^{n+1}-I_1^{n+1}}{\Delta x}\right) - c_{\frac{1}{2}}^n \left(\frac{I_1^{n+1}-I_0^{n+1}}{\Delta x}\right)}{\Delta x} \right] I_1^{n+1} \end{aligned}$$

$$\begin{aligned}
& \vdots \qquad \qquad \qquad \vdots \qquad \qquad \qquad \vdots \qquad \qquad \qquad \vdots \\
& \left( \frac{I_{N-1}^{n+1} - I_{N-1}^n}{\Delta t} \right) I_{N-1}^{n+1} = \left[ \frac{c_{N-\frac{1}{2}}^n \left( \frac{I_N^{n+1} - I_{N-1}^{n+1}}{\Delta x} \right) - c_{N-\frac{3}{2}}^n \left( \frac{I_{N-1}^{n+1} - I_{N-2}^{n+1}}{\Delta x} \right)}{\Delta x} \right] I_{N-1}^{n+1} \\
& \left( \frac{I_N^{n+1} - I_N^n}{\Delta t} \right) I_N^{n+1} = \left[ \frac{c_{N+\frac{1}{2}}^n \left( \frac{I_{N+1}^{n+1} - I_N^{n+1}}{\Delta x} \right) - c_{N-\frac{1}{2}}^n \left( \frac{I_N^{n+1} - I_{N-1}^{n+1}}{\Delta x} \right)}{\Delta x} \right] I_N^{n+1} \quad (82)
\end{aligned}$$

Using the definitions for the dot product (46) and the norm (47), the LHS of (82) for  $i=0,1,\dots,N$  in terms of the dot product which yields:

$$\begin{aligned}
\frac{\Delta x}{2} \left( \frac{I_0^{n+1} - I_0^n}{\Delta t} \right) I_0^{n+1} + \sum_{i=1}^{N-1} \Delta x \left( \frac{I_i^{n+1} - I_i^n}{\Delta t} \right) I_i^{n+1} + \frac{\Delta x}{2} \left( \frac{I_N^{n+1} - I_N^n}{\Delta t} \right) I_N^{n+1} = \\
\left( \frac{I^{n+1} - I^n}{\Delta t}, I^{n+1} \right)_{\Delta x} \quad (83)
\end{aligned}$$

In a similar fashion, add up the RHS of equation block (89) for  $i=0,1,\dots,N$  in terms of the dot product which yields:

$$\begin{aligned}
& \frac{1}{2} \left[ c_{\frac{1}{2}}^n \left( \frac{I_1^{n+1} - I_0^{n+1}}{\Delta x} \right) - c_{-\frac{1}{2}}^n \left( \frac{I_0^{n+1} - I_{-1}^{n+1}}{\Delta x} \right) \right] I_0^{n+1} \\
& + \left[ c_{\frac{3}{2}}^n \left( \frac{I_2^{n+1} - I_1^{n+1}}{\Delta x} \right) - c_{\frac{1}{2}}^n \left( \frac{I_1^{n+1} - I_0^{n+1}}{\Delta x} \right) \right] I_1^{n+1} + \dots \dots \dots \\
& + \left[ c_{N-\frac{1}{2}}^n \left( \frac{I_N^{n+1} - I_{N-1}^{n+1}}{\Delta x} \right) - c_{N-\frac{3}{2}}^n \left( \frac{I_{N-1}^{n+1} - I_{N-2}^{n+1}}{\Delta x} \right) \right] I_{N-1}^{n+1} \\
& + \frac{1}{2} \left[ c_{N+\frac{1}{2}}^n \left( \frac{I_{N+1}^{n+1} - I_N^{n+1}}{\Delta x} \right) - c_{N-\frac{1}{2}}^n \left( \frac{I_N^{n+1} - I_{N-1}^{n+1}}{\Delta x} \right) \right] I_N^{n+1}
\end{aligned} \tag{84}$$

Using the boundary conditions from Section 2.3, for the 1-D case,  $c_{-\frac{1}{2}} = c_{\frac{1}{2}}$

and  $I_{-1} = I_1$  and also  $c_{N+\frac{1}{2}} = c_{N-\frac{1}{2}}$  and  $I_{N+1} = I_{N-1}$ , the first and last terms (i.e.

when  $i=0$  and  $i=N$ ) of the above equation (84) become:

$$\frac{1}{2} \left[ c_{\frac{1}{2}}^n \left( \frac{I_1^{n+1} - I_0^{n+1}}{\Delta x} \right) - c_{-\frac{1}{2}}^n \left( \frac{I_0^{n+1} - I_{-1}^{n+1}}{\Delta x} \right) \right] I_0^{n+1} = c_{\frac{1}{2}}^n \left( \frac{I_1^{n+1} - I_0^{n+1}}{\Delta x} \right) I_0^{n+1} \tag{85}$$

$$\frac{1}{2} \left[ c_{N-\frac{1}{2}}^n \left( \frac{I_N^{n+1} - I_{N-1}^{n+1}}{\Delta x} \right) - c_{N+\frac{1}{2}}^n \left( \frac{I_{N+1}^{n+1} - I_N^{n+1}}{\Delta x} \right) \right] I_N^{n+1} = -c_{N-\frac{1}{2}}^n \left( \frac{I_N^{n+1} - I_{N-1}^{n+1}}{\Delta x} \right) \tag{86}$$



With these terms, we get:

$$\begin{aligned}
\left(\frac{I^{n+1}-I^n}{\Delta t}, I^{n+1}\right)_{\Delta x} = & \\
& -c_{\frac{1}{2}}^n \left(\frac{I_1^{n+1}-I_0^{n+1}}{\Delta x}\right) I_0^{n+1} \\
& -c_{\frac{3}{2}}^n \left(\frac{I_2^{n+1}-I_1^{n+1}}{\Delta x}\right) I_1^{n+1} - c_{\frac{1}{2}}^n \left(\frac{I_1^{n+1}-I_0^{n+1}}{\Delta x}\right) I_1^{n+1} \\
& -c_{\frac{5}{2}}^n \left(\frac{I_3^{n+1}-I_2^{n+1}}{\Delta x}\right) I_2^n - c_{\frac{3}{2}}^n \left(\frac{I_2^{n+1}-I_1^{n+1}}{\Delta x}\right) I_1^{n+1} \\
& \quad \quad \quad \vdots \quad \quad \quad \vdots \\
& -c_{N-\frac{3}{2}}^n \left(\frac{I_{N-1}^{n+1}-I_{N-2}^{n+1}}{\Delta x}\right) I_{N-2}^{n+1} - c_{N-\frac{5}{2}}^n \left(\frac{I_{N-2}^{n+1}-I_{N-3}^{n+1}}{\Delta x}\right) I_{N-2}^{n+1} \\
& -c_{N-\frac{1}{2}}^n \left(\frac{I_N^{n+1}-I_{N-1}^{n+1}}{\Delta x}\right) I_{N-1}^{n+1} - c_{N-\frac{3}{2}}^n \left(\frac{I_{N-1}^{n+1}-I_{N-2}^{n+1}}{\Delta x}\right) I_{N-1}^{n+1} \\
& -c_{N-\frac{1}{2}}^n \left(\frac{I_N^{n+1}-I_{N-1}^{n+1}}{\Delta x}\right) I_{N-1}^{n+1} \tag{87}
\end{aligned}$$

Collecting similar difference terms  $\left(\frac{I_p^{n+1}-I_{p-1}^{n+1}}{\Delta x}\right)$ , we get the following

consolidated equation:

$$\left(\frac{I^{n+1}-I^n}{\Delta t}, I^n\right)_{\Delta x} =$$

$$\begin{aligned}
& -c_{\frac{1}{2}}^n (I_1^{n+1} - I_0^{n+1}) \left( \frac{I_1^{n+1} - I_0^{n+1}}{\Delta x} \right) \\
& -c_{\frac{3}{2}}^n (I_2^{n+1} - I_1^{n+1}) \left( \frac{I_2^{n+1} - I_1^{n+1}}{\Delta x} \right) \\
& -c_{\frac{5}{2}}^n (I_3^{n+1} - I_2^{n+1}) \left( \frac{I_3^{n+1} - I_2^{n+1}}{\Delta x} \right) \\
& \vdots \\
& -c_{N-\frac{3}{2}}^n (I_{N-1}^{n+1} - I_{N-2}^{n+1}) \left( \frac{I_{N-1}^{n+1} - I_{N-2}^{n+1}}{\Delta x} \right) \\
& -c_{N-\frac{1}{2}}^n (I_N^{n+1} - I_{N-1}^{n+1}) \left( \frac{I_N^{n+1} - I_{N-1}^{n+1}}{\Delta x} \right)
\end{aligned} \tag{88}$$

Multiply every term by  $\frac{\Delta x}{\Delta t}$  and collect the terms to result in a summation on the RHS. Thus equation (95) is consolidated to the following:

$$\left( \frac{I^{n+1} - I^n}{\Delta t}, I^{n+1} \right)_{\Delta x} = - \sum_{i=1}^N \Delta x c_{i-\frac{1}{2}}^n \left( \frac{I_i^{n+1} - I_{i-1}^{n+1}}{\Delta x} \right) \left( \frac{I_i^{n+1} - I_{i-1}^{n+1}}{\Delta x} \right) \tag{89}$$

We add the summation on the RHS to the LHS to result in an equation equal to zero:

$$\left( \frac{I^{n+1} - I^n}{\Delta t}, I^{n+1} \right)_{\Delta x} + \sum_{i=1}^N \Delta x c_{i-\frac{1}{2}}^n \left( \frac{I_i^{n+1} - I_{i-1}^{n+1}}{\Delta x} \right)^2 = 0 \quad \square$$

Lemma 3.2.2  $\left(\frac{I^{n+1}-I^n}{\Delta t}, I^{n+1}\right)_{\Delta x} = \frac{\Delta t}{2} \left\| \frac{I^{n+1}-I^n}{\Delta t} \right\|_{\Delta x}^2 + \frac{\|I^{n+1}\|_{\Delta x}^2 - \|I^n\|_{\Delta x}^2}{2\Delta t}$

Now consider rewriting  $I^{n+1}$  as an expression of itself,  $I^n$ , and  $\Delta t$ :

$$I^{n+1} = \Delta t \left(\frac{1}{2}\right) \left(\frac{I^{n+1}-I^n}{\Delta t}\right) + \frac{I^{n+1}+I^n}{2} \quad (90)$$

Substituting this expression for  $I^{n+1}$  into the dot product (first term) in equation (97) gives:

$$\left(\frac{I^{n+1}-I^n}{\Delta t}, I^{n+1}\right)_{\Delta x} = \left(\frac{I^{n+1}-I^n}{\Delta t}, \Delta t \left(\frac{1}{2}\right) \left(\frac{I^{n+1}-I^n}{\Delta t}\right) + \frac{I^{n+1}+I^n}{2}\right)_{\Delta x} \quad (91)$$

By the properties of a dot product, we can move the scalar outside of the dot product operation and distribute over the sum to get a sum of two dot products:

$$\left(\frac{I^{n+1}-I^n}{\Delta t}, I^{n+1}\right)_{\Delta x} = \Delta t \left(\frac{1}{2}\right) \left(\frac{I^{n+1}-I^n}{\Delta t}, \frac{I^{n+1}-I^n}{\Delta t}\right)_{\Delta x} + \left(\frac{I^{n+1}-I^n}{\Delta t}, \frac{I^{n+1}+I^n}{2}\right)_{\Delta x} \quad (92)$$

There is a norm introduced in the first term above, according to the definition of a norm in equation (49). Also, the scalar values can be extracted to the outside of the dot product in the second term above:

$$\left(\frac{I^{n+1}-I^n}{\Delta t}, I^n\right)_{\Delta x} = \frac{\Delta t}{2} \left\| \frac{I^{n+1}-I^n}{\Delta t} \right\|_{\Delta x}^2 + \frac{1}{2\Delta t} (I^{n+1} - I^n, I^{n+1} + I^n)_{\Delta x} \quad (93)$$

By carefully following the properties of distribution wrt dot products, we can rewrite the dot product in the last term above as:

$$\begin{aligned} (I^{n+1} - I^n, I^{n+1} + I^n)_{\Delta x} &= (I^{n+1}, I^{n+1})_{\Delta x} + (I^{n+1}, I^n)_{\Delta x} - \\ &\quad (I^n, I^{n+1})_{\Delta x} - (I^n, I^n)_{\Delta x} \end{aligned} \quad (94)$$

The interior terms of (97) cancel out. Observe that the first and last terms are the square of the norms. Then (97) becomes:

$$\left(\frac{I^{n+1}-I^n}{\Delta t}, I^{n+1}\right)_{\Delta x} = \frac{\Delta t}{2} \left\| \frac{I^{n+1}-I^n}{\Delta t} \right\|_{\Delta x}^2 + \frac{\|I^{n+1}\|_{\Delta x}^2 - \|I^n\|_{\Delta x}^2}{2\Delta t} \quad \square$$

*Proof of Theorem 3.2.* Substituting Lemma 3.2.2 into Lemma 3.2.1, we get:

$$\frac{\Delta t}{2} \left\| \frac{I^{n+1}-I^n}{\Delta t} \right\|_{\Delta x}^2 + \frac{\|I^{n+1}\|_{\Delta x}^2 - \|I^n\|_{\Delta x}^2}{2\Delta t} + \sum_{i=1}^N \Delta x c_{i-\frac{1}{2}}^n \left( \frac{I_i^{n+1} - I_{i-1}^{n+1}}{\Delta x} \right)^2 = 0 \quad (95)$$

The first term of (99) is positive since the scalar is positive because  $\Delta t \geq 0 \rightarrow \frac{\Delta t}{2} \geq 0$  and the squared norm is positive. Now skip to the third term (summation), we have that  $\Delta x \geq 0$  and the squared difference is greater than zero. We force the  $c$  values to be within  $[0,1]$  by choice of the  $g$  function, therefore  $c_{i-\frac{1}{2}}^n \geq 0$  and thus the summation is greater than or equal to zero. Now in order for the three terms above to sum to 0, given two positive terms, it must be true that the second term is less than or equal to zero. Consider that fact:

$$\frac{\|I^{n+1}\|_{\Delta x}^2 - \|I^n\|_{\Delta x}^2}{2\Delta t} \leq 0 \quad (96)$$

Cross-multiplying the  $2\Delta t$  by zero gives:

$$\|I^{n+1}\|_{\Delta x}^2 - \|I^n\|_{\Delta x}^2 \leq 0 \leftrightarrow \|I^{n+1}\|_{\Delta x}^2 \leq \|I^n\|_{\Delta x}^2 \quad \square$$

*Remark.* Then this brings us directly to the definition of stability and convergence, i.e. we proved that the  $(n+1)$ -step is bounded by the  $n$ -step, *without* any conditions on  $\Delta t$  or any other parameter. The only careful choice that has to be made is of the  $g$  function that determines  $c$ , with the safest choice being one that is bounded by 0 and 1. In summary, we are confident that given a careful choice of the

diffusion coefficient function  $c$ , the Implicit Numerical Scheme for the Perona-Malik function will always converge to stable solution regardless of choice of  $\Delta t$ .

## CHAPTER IV – MATLAB IN A LITERATE PROGRAMMING STYLE

Using the dissertation style of Van Den Boomgaard , we will examine how MATLAB was used to implement the above numerical approximations, directly within this paper to allow the reader to understand the implementation step-by-step and reproduce similar results. All of the MATLAB implementations allow processing on a non-uniform grid space which we call the *computational grid space* as opposed to the *image grid space* which is the pixel space of the original image. A mesh grid is created based on user defined step sizes for x and y,  $dx (= \Delta x)$  and  $dy (= \Delta y)$ , respectively:

```
[im_compX, im_compY] = meshgrid(dx:dx:Iorig_max, dy:dy:Jorig_max);
```

Figure 2: MATLAB code for creating the mesh grid to overlay the computational grid space

In order to create the computational grid space, an interpolation must be performed to fill in the values between pixels. The built-in MATLAB function `interp2` was used to execute a bilinear interpolation:

```
im_comp =  
    interp2(im_origX, im_origY, im_orig, im_compX, im_compY, 'linear');
```

Figure 3: MATLAB code for using bilinear interpolation to create the computational grid space (i.e. the computational image)

The function `interp2` was also used to examine spline interpolation:

```
im_comp =  
    interp2(im_origX,im_origY,im_orig,im_compX,im_compY,'spline');
```

Figure 4: MATLAB code for using spline interpolation to create the computational grid space (i.e. the computational image)

If  $dx = 1$  and  $dy = 1$  then the computational grid space will be equal to the image grid space and the processing will be done on a pixel by pixel basis.

#### 4.1 Implementation of Explicit Perona-Malik Numerical Scheme

The MATLAB implementation of the explicit PM numerical scheme is illustrated in figure 5:

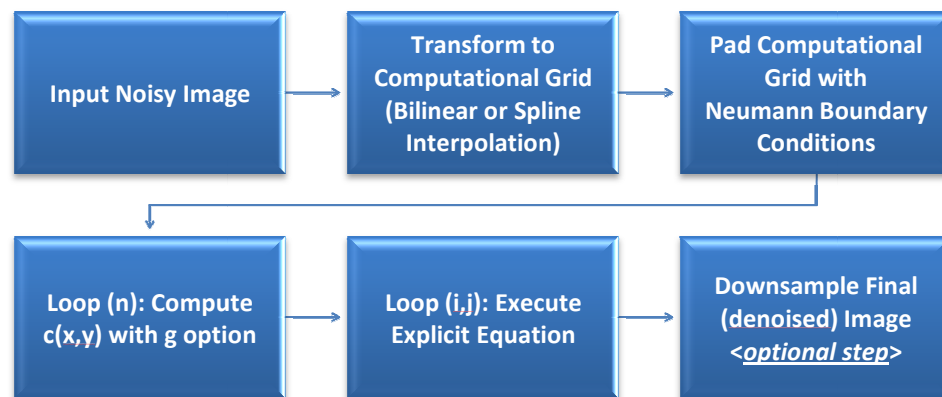


Figure 5: MATLAB Flow for Explicit PM Numerical Scheme



A function named `pm_explicit.m` was created to execute Perona-Malik explicit numerical scheme:

```
function [im_final varargout] =
    pm_explicit(im_orig,kval,dx,dy,dt,nsteps,tol,option_g,varargin)
```

**Figure 6: MATLAB code to call function `pm_explicit` to run Explicit PM 2<sup>nd</sup> Order Difference scheme**

The function outputs the result of the Perona-Malik equation. The inputs are the original image, `im_orig`, the  $K$  value from the  $g$  functions (4) and (5), the step size in the  $x$  direction, `dx`, the step size in the  $y$  direction, `dy` and the choice of the  $g$  function, `option_g`. The first step involves calculating the upper bound on the step size in time per the stability condition discussion in Section 3.1.

```
dtmax = (dx^2)*(dy^2)/(2*(dx^2 + dy^2));
```

**Figure 7: MATLAB code for maximum allowed time step, `dtmax`**

We choose the time step size to be just 1% less than the upper bound:

```
dt = dtmax - abs(dtmax)/100;
```

**Figure 8: MATLAB code for time step calculated as 99% of `dtmax`**

All of the processing is done on the computational grid space, which we will also refer to as the image (specifically, `im_comp`, in the MATLAB code). The computational image is created as mentioned in the discussion above and is then padded on each side, based on the Neumann Boundary Conditions wrt to  $x$  and  $y$ , i.e. at the boundaries we get:

$$\frac{\partial I}{\partial x} = \frac{I_{i+1,j} - I_{i-1,j}}{2\Delta x} = 0 \quad \text{and} \quad \frac{\partial I}{\partial y} = \frac{I_{i,j+1} - I_{i,j-1}}{2\Delta y} = 0 \quad (97)$$

It is important to keep in mind that in MATLAB all indices must start with 1 (not 0). So let  $i = 1, 2, \dots, I_{\max}$  and  $j = 1, 2, \dots, J_{\max}$  denote the step values in the x and y directions, respectively. At the lower boundary of x in MATLAB (i.e.  $i = 1$ ):

$$\left. \frac{\partial I}{\partial x} \right|_{1,j} = \frac{I_{2,j} - I_{0,j}}{2\Delta x} = 0 \rightarrow I_{0,j} = I_{2,j} \quad (98)$$

At the upper boundary for x (i.e.  $i = I_{\max}$ ):

$$\left. \frac{\partial I}{\partial x} \right|_{I_{\max},j} = \frac{I_{I_{\max}+1,j} - I_{I_{\max}-1,j}}{2\Delta x} = 0 \rightarrow I_{I_{\max}+1,j} = I_{I_{\max}-1,j} \quad (99)$$

Similarly, at the lower boundary of y in MATLAB (i.e.  $j = 1$ ):

$$\left. \frac{\partial I}{\partial y} \right|_{i,1} = \frac{I_{i,2} - I_{i,0}}{2\Delta y} = 0 \rightarrow I_{i,0} = I_{i,2} \quad (100)$$

At the upper boundary for y (i.e.  $j = J_{\max}$ ):

$$\left. \frac{\partial I}{\partial y} \right|_{i,J_{\max}} = \frac{I_{i,J_{\max}+1} - I_{i,J_{\max}-1}}{2\Delta y} = 0 \rightarrow I_{i,J_{\max}+1} = I_{i,J_{\max}-1} \quad (101)$$

In MATLAB, the padding is executed with the following statements:

```
im_pad = zeros(Imax+2,Jmax+2);
im_pad(2:Imax+1,2:Jmax+1) = im_comp;
im_pad(1,2:Jmax+1)         = im_pad(3,2:Jmax+1);
im_pad(Imax+2,2:Jmax+1)    = im_pad(Imax,2:Jmax+1);
im_pad(2:Imax+1,1)         = im_pad(2:Imax+1,3);
im_pad(2:Imax+1,Jmax+2)    = im_pad(2:Imax+1,Jmax);
```

**Figure 9: MATLAB code for padding image**

Then at each  $(i, j)$  coordinate within the computational grid-space we compute the numerical approximation to the gradient of the intensities:

```
% Solve for new intensity using only interior of padded computational
grid
for n=1:nsteps

    % Find c at each point, in order to find c at each point we need

    % Calculate the gradient of the intensities (of the padded image)
    gradI_x = zeros(size(im_pad));
    gradI_y = zeros(size(im_pad));

    for i=2:Imax+1
        for j=2:Jmax+1
            gradI_x(i,j) = (im_pad(i+1,j)-im_pad(i-1,j))/(2*dx);
            gradI_y(i,j) = (im_pad(i,j+1)-im_pad(i,j-1))/(2*dy);
        end
    end

    ... loop to be continued in Figure 11
```

**Figure 10: MATLAB code for calculating the gradient of the intensities**

The Euclidean norm of the gradient is computed and passed into the chosen g function:

```
... loop continued from Figure 10

% Calculate the 2-norm of the gradient of the intensities
nrm_gradI = sqrt(gradI_x.^2 + gradI_y.^2);

% Determine and evaluate the g function
switch option_g
    case 1
        gfun = exp(-(nrm_gradI/kval).^2);
    case 2
        gfun = 1./(1+(nrm_gradI/kval).^2);
    otherwise
        gfun = ones(size(nrm_gradI)); % Gaussian Kernel
end

cval = gfun; % Is this step necessary???

... to be continued in Figure 12
```

**Figure 11: MATLAB Code for choice of g function – diffusivity function coefficient**

We iterate over the difference time steps to compute equation (20) iteratively up to nsteps (number of time steps):

```

... loop continued from Figure 11

% CHECK for no change between previous and current timesteps (< tol)
im_prev = im_pad;

for i=2:Imax+1
    for j=2:Jmax+1
        % Perform all calculations wrt X
        A1 = (cval(i+1,j)+cval(i,j))*(im_pad(i+1,j)-im_pad(i,j))/2;
        A2 = (cval(i,j)+cval(i-1,j))*(im_pad(i,j)-im_pad(i-1,j))/2;
        A = dt*(A1 - A2)/dx^2;

        % Perform all calculations wrt Y
        B1 = (cval(i,j+1)+cval(i,j))*(im_pad(i,j+1)-im_pad(i,j))/2;
        B2 = (cval(i,j)+cval(i,j-1))*(im_pad(i,j)-im_pad(i,j-1))/2;
        B = dt*(B1 - B2)/dy^2;

        im_pad(i,j) = im_pad(i,j) + A + B;
    end
end

iter_err = norm(im_prev - im_pad)/norm(im_prev);
if iter_err < tol && n > 2
    break
end

end

```

**Figure 12: MATLAB code for calculating Explicit PM 2<sup>nd</sup> Order Difference over each ((i,j)) pixel of the computational grid**

To run the pm\_explicit.m function code, first read in an image, e.g. built-in MATLAB image, cameraman.tif:

```

im_orig = imread('cameraman.tif');

```

**Figure 13: MATLAB Code to read MATLAB built-in image - cameraman.tif**

Then use the following commands to set up the parameters and execute the `pm_explicit` function to run the Explicit PM Numerical Scheme:

```
kval    = 1/7;  
nsteps  = 7;  
option  = 1;  
dx       = 0.5;  
dy       = 0.5;  
dt       = 0.25;  
tol      = 1e-8;  
[im_final exp_dt exp_n exp_ierr] =  
    pm_explicit(im_noise,kval,dx,dy,dt,nsteps,tol,option);
```

**Figure 14: MATLAB code to define parameters in run script for inputs to `pm_explicit`**

The final result is in the processed computational grid (without padding). Therefore you may want to downsample the final image by the  $dx$  and  $dy$  factors in order to return to the image grid space for comparison purposes. The following MATLAB command can be run (note, for optimal return to same image grid space size, use  $dx = 1/M$  and  $dy = 1/N$ , i.e. the reciprocal of an integer):

```
im_down = downsample(downsample(im_final',ceil(1/dx))',ceil(1/dy));
```

**Figure 15: MATLAB code to downsample the final image to original image grid space size (approximately)**

## 4.2 Implementation of Implicit Perona-Malik Numerical Scheme

The MATLAB implementation for the Implicit Perona-Malik Numerical Scheme is very similar to the Explicit Perona-Malik Numerical MATLAB

implementation. The only change will be the section of code that iterates over the underlying equation. Instead in the implicit method a function, `pm_implicit_matrix.m`, is executed to generate the matrix operator that represents the coefficient matrix for the system of equations of the implicit method.

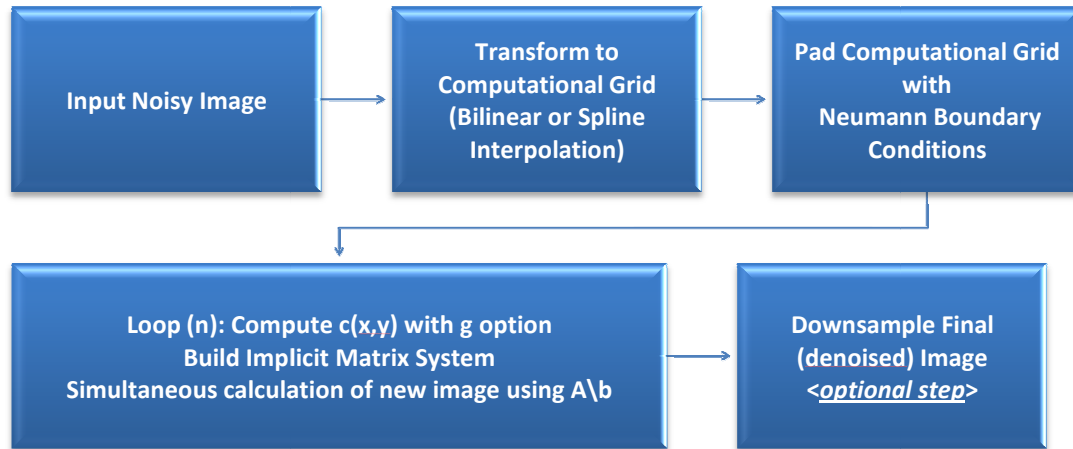


Figure 16: MATLAB Flow for Implicit PM Numerical Scheme

A function named `pm_implicit.m` was created to execute Perona-Malik implicit numerical scheme:

```
function A = pm_implicit_matrix(Imax,Jmax,dt,dx,dy,cval)
```

Figure 17: MATLAB Function for Implicit Matrix

The implicit scheme requires this intermediate solver to solve for the  $n+1$  term. Also, per the stability discussion in Section 3.2, there is no need to restrict the

choice of the time step,  $\Delta t$ , which no longer depends on the  $x$  and  $y$  step sizes. For the implicit numerical scheme of the Perona-Malik function we take equation (29):

$$I_{i,j}^{n+1} \left( 1 + \lambda_1 \left( c_{i+\frac{1}{2},j}^n + c_{i-\frac{1}{2},j}^n \right) + \lambda_2 \left( c_{i,j+\frac{1}{2}}^n + c_{i,j-\frac{1}{2}}^n \right) \right) - \lambda_1 \left( c_{i+\frac{1}{2},j}^n I_{i+1,j}^{n+1} + c_{i-\frac{1}{2},j}^n I_{i-1,j}^{n+1} \right) - \lambda_2 \left( c_{i,j+\frac{1}{2}}^n I_{i,j+1}^{n+1} + c_{i,j-\frac{1}{2}}^n I_{i,j-1}^{n+1} \right) = I_{i,j}^n \quad (102)$$

Rewrite equation (29) to be a system of unknown  $I^{n+1}$  terms:

$$- I_{i+1,j}^{n+1} \left( \lambda_1 c_{i+\frac{1}{2},j}^n \right) - I_{i-1,j}^{n+1} \left( \lambda_1 c_{i-\frac{1}{2},j}^n \right) + I_{i,j}^{n+1} \left( 1 + \lambda_1 \left( c_{i+\frac{1}{2},j}^n + c_{i-\frac{1}{2},j}^n \right) + \lambda_2 \left( c_{i,j+\frac{1}{2}}^n + c_{i,j-\frac{1}{2}}^n \right) \right) - I_{i,j+1}^{n+1} \left( \lambda_2 c_{i,j+\frac{1}{2}}^n \right) - I_{i,j-1}^{n+1} \left( \lambda_2 c_{i,j-\frac{1}{2}}^n \right) = I_{i,j}^n \quad (103)$$

Coefficient for the  $I_{i,j}^{n+1}$  term:

$$\gamma = \left( 1 + \lambda_1 \left( c_{i+\frac{1}{2},j}^n + c_{i-\frac{1}{2},j}^n \right) + \lambda_2 \left( c_{i,j+\frac{1}{2}}^n + c_{i,j-\frac{1}{2}}^n \right) \right) \quad (104)$$



Coefficients for the  $I_{i+1,j}^{n+1}$  and  $I_{i-1,j}^{n+1}$  terms, respectively:

$$\alpha_1 = -\left(\lambda_1 c_{i+\frac{1}{2},j}^n\right), \quad \alpha_2 = -\left(\lambda_1 c_{i-\frac{1}{2},j}^n\right), \quad (\text{Note: } \alpha_{1+2} = \alpha_1 + \alpha_2) \quad (105)$$

Coefficients for the  $I_{i,j+1}^{n+1}$  and  $I_{i,j-1}^{n+1}$  terms, respectively:

$$\beta_1 = -\left(\lambda_2 c_{i,j+\frac{1}{2}}^n\right), \quad \beta_2 = -\left(\lambda_2 c_{i,j-\frac{1}{2}}^n\right), \quad (\text{Note: } \beta_{1+2} = \beta_1 + \beta_2) \quad (106)$$

The first step in pm\_implicit\_matrix.m calculates the  $\lambda_1$  and  $\lambda_2$  values:

```
lambda1 = dt/(dx^2);
lambda2 = dt/(dy^2);
```

**Figure 18: MATLAB Code – pm\_implicit\_matrix.m**

The coefficients are in terms of the c values at the halves, hence those constants are explicitly calculated first. The cvals matrix variable of the c values is passed into pm\_implicit\_matrix.m function. The coefficients  $\gamma$ ,  $\alpha_1$ ,  $\alpha_2$ ,  $\beta_1$ , and  $\beta_2$  are then calculated for each i and j at each iteration in time. The first portion of the MATLAB to calculate all those values follows:

```

r = 0;
for j = 2:Jmax+1
    for i = 2:Imax+1
        r = r+1;

        ci1 = (cval(i+1,j)+cval(i,j))/2;
        ci2 = (cval(i,j)+cval(i-1,j))/2;
        cj1 = (cval(i,j+1)+cval(i,j))/2;
        cj2 = (cval(i,j)+cval(i,j-1))/2;

        gamma = 1 + lambda1*(ci1+ci2) +lambda2*(cj1+cj2);
        alpha1 = -lambda1*ci1;
        alpha2 = -lambda1*ci2;
        beta1 = -lambda2*cj1;
        beta2 = -lambda2*cj2;

... to be continued in Figure 20

```

**Figure 19: MATLAB Code – pm\_implicit\_matrix.m - 1<sup>st</sup> Half of Loop**

Now the system can be expressed in terms of the above coefficients  $\gamma$ ,  $\alpha_1$ ,  $\alpha_2$ ,  $\beta_1$ , and  $\beta_2$ . All terms are summed because the variables include the negative signs:

$$\alpha_1 I_{i+1,j}^{n+1} + \alpha_2 I_{i-1,j}^{n+1} + \gamma I_{i,j}^{n+1} + \beta_1 I_{i,j+1}^{n+1} + \beta_2 I_{i,j-1}^{n+1} = I_{i,j}^n \quad (107)$$

Consider holding  $n = 0$ , fix the  $j$  term, then for  $i=1,2,...,M=Imax$  we get

$$\begin{aligned}
 \alpha_1 I_{2,j}^1 + \alpha_2 I_{0,j}^1 + \gamma I_{1,j}^1 + \beta_1 I_{1,j+1}^1 + \beta_2 I_{1,j-1}^1 &= I_{1,j}^0 \\
 \alpha_1 I_{3,j}^1 + \alpha_2 I_{1,j}^1 + \gamma I_{2,j}^1 + \beta_1 I_{2,j+1}^1 + \beta_2 I_{2,j-1}^1 &= I_{2,j}^0 \\
 \vdots &\vdots
 \end{aligned}$$

$$\begin{aligned}
\alpha_1 I_{M,j}^1 + \alpha_2 I_{M-2,j}^1 + \gamma I_{M-1,j}^1 + \beta_1 I_{M-1,j+1}^1 + \beta_2 I_{M-1,j-1}^1 &= I_{M-1,j}^0 \\
\alpha_1 I_{M+1,j}^1 + \alpha_2 I_{M-1,j}^1 + \gamma I_{M,j}^1 + \beta_1 I_{M,j+1}^1 + \beta_2 I_{M,j-1}^1 &= I_{M,j}^0 \quad (108)
\end{aligned}$$

This will create a system of M (= I<sub>max</sub>) equations for M unknowns. Let N = J<sub>max</sub>,

then the above system is repeated N times, resulting in an M\*N equations. List the

intensities to be solved as one single (MN x 1)-dimensional vector, listing all values

of i(=1,2,...,M=I<sub>max</sub>) for each j(=1,2,...,N=J<sub>max</sub>). The final system can be represented

by a (MN x MN) matrix A, left-multiplied by the MN-dimension vector of all

intensities. We illustrate with a 4x4 system:

$$\begin{bmatrix}
\gamma & \alpha_{1+2} & 0 & \beta_{1+2} & 0 & 0 & 0 & 0 & 0 \\
\alpha_2 & \gamma & \alpha_1 & 0 & \beta_{1+2} & 0 & 0 & 0 & 0 \\
0 & \alpha_2 & \gamma & \alpha_1 & 0 & \beta_{1+2} & 0 & 0 & 0 \\
\beta_2 & 0 & 0 & \gamma & \alpha_{1+2} & 0 & \beta_1 & 0 & 0 \\
0 & \beta_2 & 0 & \alpha_2 & \gamma & \alpha_1 & 0 & \beta_1 & 0 \\
0 & 0 & \beta_2 & 0 & \alpha_2 & \gamma & \alpha_1 & 0 & \beta_1 \\
0 & 0 & 0 & \beta_{1+2} & 0 & 0 & \gamma & \alpha_{1+2} & 0 \\
0 & 0 & 0 & 0 & \beta_{1+2} & 0 & \alpha_2 & \gamma & \alpha_1 \\
0 & 0 & 0 & 0 & 0 & \beta_{1+2} & 0 & \alpha_2 & \gamma
\end{bmatrix} \cdot \begin{bmatrix} I_{1,1}^{n+1} \\ I_{2,1}^{n+1} \\ I_{3,1}^{n+1} \\ I_{1,2}^{n+1} \\ I_{2,2}^{n+1} \\ I_{3,2}^{n+1} \\ I_{1,3j}^{n+1} \\ I_{2,3}^{n+1} \\ I_{3,3}^{n+1} \end{bmatrix} = \begin{bmatrix} I_{1,1}^n \\ I_{2,1}^n \\ I_{3,1}^n \\ I_{1,2}^n \\ I_{2,2}^n \\ I_{3,2}^n \\ I_{1,3j}^n \\ I_{2,3}^n \\ I_{3,3}^n \end{bmatrix} \quad (109)$$

Recall that boundary conditions exist whenever  $i=1$  or  $i=M=I_{\max}$  or whenever  $j=1$  or  $j=N=J_{\max}$  (see discussion in previous section surrounding equations (108)-(111)). This is what causes the combined coefficients of  $\alpha_{1+2}$  and  $\beta_{1+2}$ . The following excerpt of code shows the second half of the loop that sets up the matrix and updates the coefficients when at the boundaries.

```

...continued from Figure 19

    if (i-1) == 1
        alpha1 = alpha1 + alpha2;
        alpha2 = 0;
    end

    if (j-1) == 1
        beta1 = beta1 + beta2;
    end

    if (j-1) == Jmax
        beta2 = beta1 + beta2;
    end

    A(r,r) = gamma;
    if r < Imax*Jmax
        A(r,r+1) = alpha1;
    end

    if r > 1
        A(r,r-1) = alpha2;
    end

    if r <= Imax*Jmax-Jmax
        A(r,r+Jmax) = beta1;
    end

    if r > Jmax
        A(r,r-Jmax) = beta2;
    end
end
end

```

**Figure 20: MATLAB Code – pm\_implicit\_matrix.m – 2<sup>nd</sup> Half of Loop**

The result is a standard matrix system,  $Ax=b$ . This is used in the parent function:

```
function [im_final varargout] =
    pm_implicit(im_orig,kval,dx,dy,dt,nsteps,tol,option_g,varargin)
```

**Figure 21: MATLAB**

As mentioned before the padding and transformation to the computational grid is the same as that for the pm\_explicit.m function. For completeness, all the first steps are included again here:

```
% Create mesh of original image space and create mesh based on dx,dy
steps
[im_origX,im_origY] = meshgrid(1:Iorig_max,1:Jorig_max);
[im_compX,im_compY] = meshgrid(1:dx:Iorig_max,1:dy:Jorig_max);

% Use interpolation to create the intensities of the computational grid
im_comp =
interp2(im_origX,im_origY,im_orig,im_compX,im_compY,'linear');

im_pad = zeros(Imax+2,Jmax+2);
im_pad(2:Imax+1,2:Jmax+1) = im_comp;
im_pad(1,2:Jmax+1) = im_pad(3,2:Jmax+1);
im_pad(Imax+2,2:Jmax+1) = im_pad(Imax,2:Jmax+1);
im_pad(2:Imax+1,1) = im_pad(2:Imax+1,3);
im_pad(2:Imax+1,Jmax+2) = im_pad(2:Imax+1,Jmax);
```

**Figure 22: MATLAB Code: pm\_implicit.m – Bilinear Interpolation and Image Padding**

After interpolation and padding, the iterations in time space are commenced, prior to calling the pm\_implicit\_matrix.m, the c function values must be calculated.

```
for n=1:nsteps

    % Find c at each point, in order to find c at each point we need to
    % calculate the gradient of the intensities (of the padded image)
    gradI_x = zeros(size(im_pad));
    gradI_y = zeros(size(im_pad));

    for i=2:Imax+1
        for j=2:Jmax+1
            gradI_x(i,j) = (im_pad(i+1,j)-im_pad(i-1,j))/(2*dx);
            gradI_y(i,j) = (im_pad(i,j+1)-im_pad(i,j-1))/(2*dy);
```

```

        end
    end

    % Calculate the 2-norm of the gradient of the intensities
    nrm_gradI = sqrt(gradI_x.^2 + gradI_y.^2);

    % Determine and evaluate the g function
    switch option_g
        case 1
            gfun = exp(-(nrm_gradI/kval).^2);
        case 2
            gfun = 1./(1+(nrm_gradI/kval).^2);
        otherwise
            gfun = ones(size(nrm_gradI)); % Gaussian Kernel
    end

    cval = gfun;

... to be continued in Figure 24

```

**Figure 23: MATLAB Code: Implicit PM Scheme (pm\_implicit.m) – Part 1 of 3**

The A matrix is then created per the discussion at the beginning of this section:

```

... continued from Figure 23

% Generate IJxIJ matrix to solve implicit system
A = pm_implicit_matrix(Imax,Jmax,dt,dx,dy,cval);

... loop to be continued in Figure 25

```

**Figure 24: MATLAB code – pm\_implicit.m – Part 2 of 3**

In order to create the b vector, which is simply one long vector depicting the intensity values at the current time step, the built-in MATLAB function “reshape” was used. Since MATLAB is a column-major programming language, the reshape function will traverse the intensity matrix by column which directly corresponds to how the vector is describe above (fix j, select all I values). Once the matrix A and vector b are created the system is solved with the built-in MATLAB command “\”:

```

... continued from Figure 24

% CHECK for no change between previous and current timesteps (< tol)
im_prev = im_pad;
%toc

im_bvec = reshape(im_pad(2:Imax+1,2:Jmax+1),Imax*Jmax,1);
%toc

im_xvec = A \ im_bvec;
%toc

im_pad(2:Imax+1,2:Jmax+1) = reshape(im_xvec,Imax,Jmax);
toc

iter_err = norm(im_prev - im_pad)/norm(im_prev);
if iter_err < tol && n > 2
    break
end

end

```

**Figure 25: MATLAB Code: pm\_implicit.m – Part 3 of 3**

The final image is taken to be the interior of the padded image to remove the logical boundary conditions:

```

im_final = im_pad(2:Imax+1,2:Jmax+1);

```

**Figure 26: Implicit PM Scheme (pm\_implicit.m)**

Below are commands to set up the parameters and execute the pm\_implicit function to run the Implicit PM Numerical Scheme:

```

kval    = 1/7;
nsteps  = 100;
option  = 2;
dt       = 0.01;
dx       = 1;
dy       = 1;
tol      = 1e-8;
[im_final imp_dt imp_n imp_ierr] =

```

```
pm_implicit(im_noise,kval,dx,dy,dt,nsteps,tol,option);
```

**Figure 27: MATLAB code to run Implicit Scheme**

### **4.3 Simulated Noise: Random (Additive), Gaussian and Speckle**

In order to simulate a noisy image, random values are directly added to the original image, hence why it is addressed as additive random noise. The built-in MATLAB function “rand” is used which produces random values between [0,1]. The noise is reduced or increased based on a factor. If it is a fraction then the noise is reduced, if it is greater than 1, then it is increased. The added noise is saved to a variable called “noise\_floor” to store exactly what was added to the image for simulated degradation:

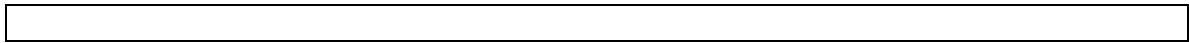
```
noise_factor = 1/5;  
noise_floor = rand(size(im_orig))*noise_factor;  
im_noise = im2double(im_orig) + noise_floor;
```

**Figure 28: MATLAB Code for Adding Random Noise (reduced by factor of 5).**

The MATLAB Image Processing Toolbox (available in Student Edition) contains a very convenient built-in function called “imnoise” to simulate various types of noise. In this thesis Gaussian noise and Speckle were also examined as sources of simulated noise for degrading the image. The speckle is multiplicative noise created by  $\text{im\_noise} = \text{im\_orig} + n \cdot \text{im\_orig}$ , where  $n$  is random between [0,1] and with default variance of 0.04) See implementations below:

```
im_gaussian = imnoise(im_orig,'gaussian');  
im_speckle = imnoise(im_orig,'speckle');
```





**Figure 29: MATLAB built-in function imnoise for Gaussian and Multiplicative Noise (Speckle)**

A very useful resource is the MATLAB Help or [www.mathworks.com](http://www.mathworks.com) for more information on all MATLAB built-in functions. The full MATLAB functions are included in the supplemental material with this thesis.

## CHAPTER V – EXPERIMENTAL RESULTS

All experiments were performed with benchmark images in image processing. Specifically, the “Camera Man” and “Lena” images were considered, although majority of the work was investigated with the camera man image.



Figure 30: Cameraman Image (256x256 TIFF)



Figure 31: Lena Image (512x512 TIFF)

### 5.1 Performance Metrics

The first experimental analysis that was performed took into consideration the theoretical attributes of the numerical schemes. Mainly, holding all parameters the same and decreasing  $\Delta t$ , the relative error of the solution to the original image

should decrease as well. It was noticed that this indeed was the case for both the explicit and implicit MATLAB implementations, pm\_explicit.m and pm\_implicit.m, respectively. In order to speed up the experiments, a subset of the cameraman.tif image was used. The parameters (held constant) used for running the experiment are listed in Table 1.

Parameter	MATLAB Variable Name	Value
Contrast Parameter, K	kval	1/7
Step size in x direction, $\Delta x$	dx	1
Step size in y direction, $\Delta y$	dy	1
Choice of g function	option	1
Maximum number of iterations	nsteps	1000
Tolerance (between successive iterations – stopping criteria)	tol	1e-9
Camera Man Image – Subset Range	subX subY	(30, 130) (80, 180)

**Table 1: Description of Fixed Parameters**

The maximum number of iterations is set in case the tolerance is not reached.

The tolerance is checked against the relative error between iterations:

$$Iteration\ Error = \frac{\|I^n - I^{n-1}\|}{\|I^n\|} < Tolerance \quad (110)$$

To determine the overall accuracy of the scheme, the relative error was calculated against the original image and the final image:

$$\text{Relative (Final) Error} = \frac{\|(Original Image)-(Final PM Image)\|}{\|Original Image\|} \quad (111)$$

The explicit PM numerical scheme processing time averaged at about 38-39 seconds and the relative error was within one order of error compared with the  $\Delta t$ . In other words, if  $\Delta t = 1E-N$ , then the relative error would be  $1E-(N-1)$  or better.

Explicit - Original Image - No Noise Added				
Delta-T	Number Iterations	Relative Error	Iteration Error	Time (sec)
2.475E-01	1000	2.823E-01	9.020E-05	39
1.000E-01	1000	2.122E-01	1.198E-04	37
1.000E-02	1000	8.721E-02	6.624E-05	39
1.000E-03	1000	3.368E-02	1.863E-05	38
1.000E-04	1000	5.939E-03	5.372E-06	38
1.000E-05	1000	6.342E-04	6.128E-07	38
1.000E-06	1000	6.384E-05	6.209E-08	38

Table 2: Explicit PM Results for different values of  $\Delta t$  for processing the original image without any noise added.

The implicit PM numerical scheme processing time averaged at about 17-18 minutes. Hence the implicit method was approximately 27 times slower compared to the explicit method. The relative error was within two orders of error compared with the  $\Delta t$  (i.e. if  $\Delta t = 1E-N$ , then the relative error would be  $1E-(N-2)$  or better).

Implicit - Original Image - No Noise Added				
Delta-T	Number Iterations	Relative Error	Iteration Error	Time (sec)
1.000E+00	1000	4.449E-01	9.770E-05	945
1.000E-01	1000	2.691E-01	1.555E-04	941
1.000E-02	1000	1.278E-01	6.465E-05	964
1.000E-03	1000	5.990E-02	3.445E-05	1066
1.000E-04	1000	1.019E-02	9.086E-06	1135
1.000E-05	1000	1.104E-03	1.065E-06	1166
1.000E-06	1000	1.113E-04	1.082E-07	1163

Table 3: Implicit PM Results for different values of  $\Delta t$  for processing the original image without any noise added.

Graphically we should see an increasing pattern, since theory says that the  $\Delta t$  is positively proportional to the error. This was confirmed in the experiment:

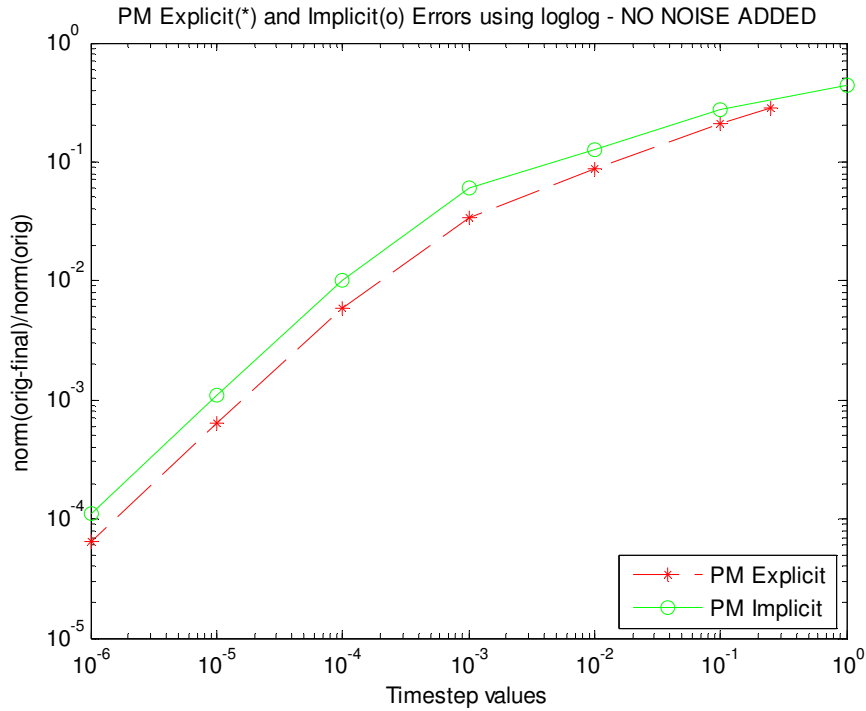
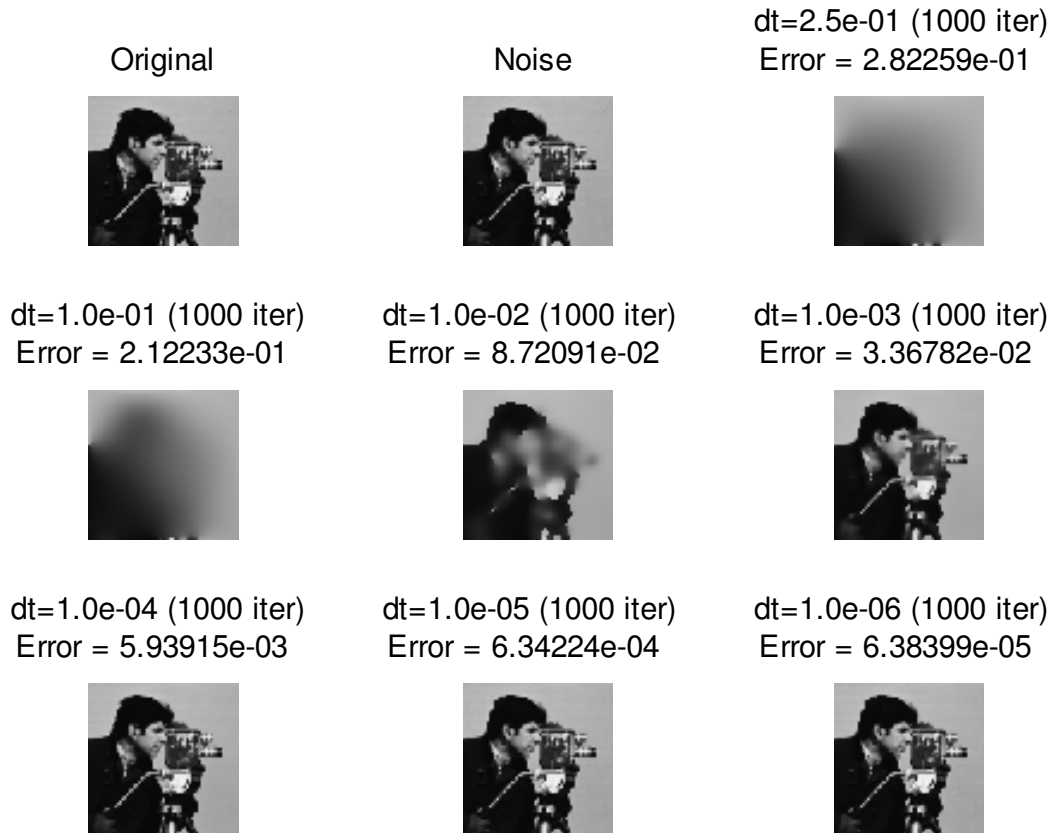
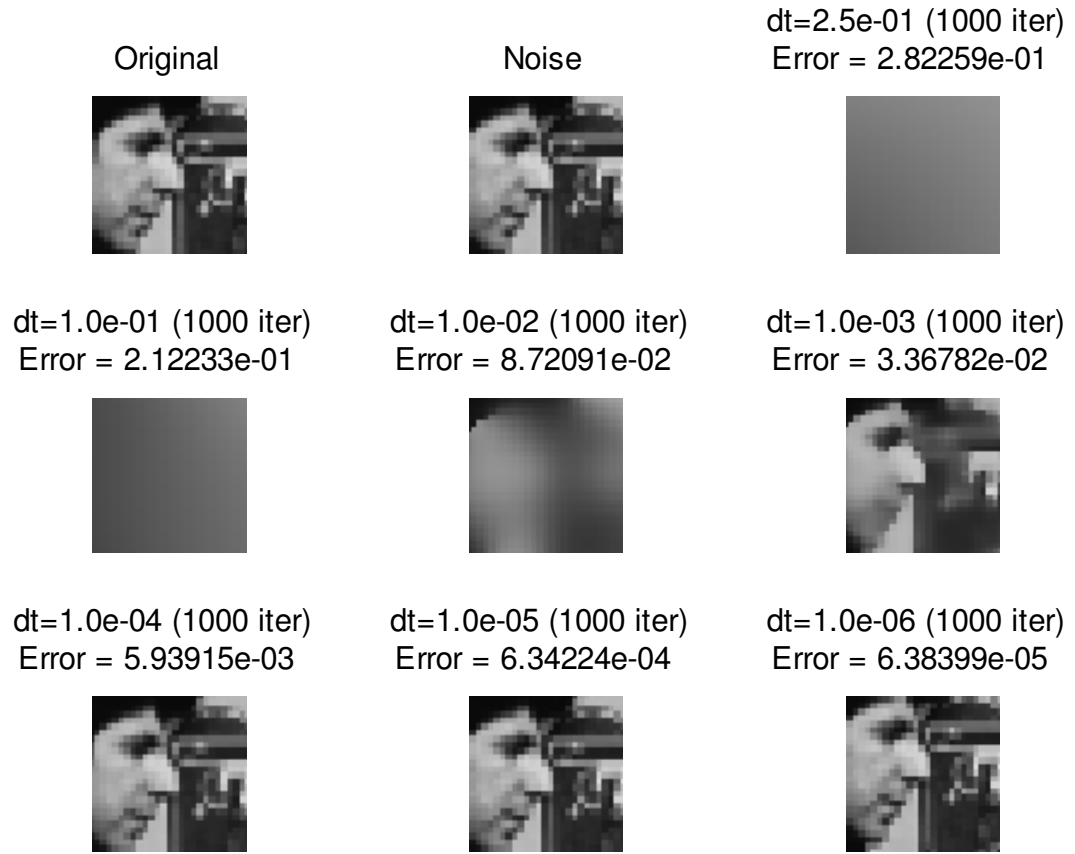


Figure 32: Errors of Explicit (Red Star Dotted Line) and Implicit (Green Circle Solid Line) Methods wrt  $\Delta t$ . Decrease in error with smaller values of  $\Delta t$  validate the implementation of the algorithm

From a numerical standpoint the methods are accurate. But in image processing the visual context is more significant than the numerical errors. Like Perona and Malik mentioned in their paper [1], the goal is to emphasize the “semantically meaningful” aspects of the image. In visual analysis of the results, only can one make the decision if the numerical scheme “worked”. These experiments also help determine an optimal choice of  $\Delta t$ .

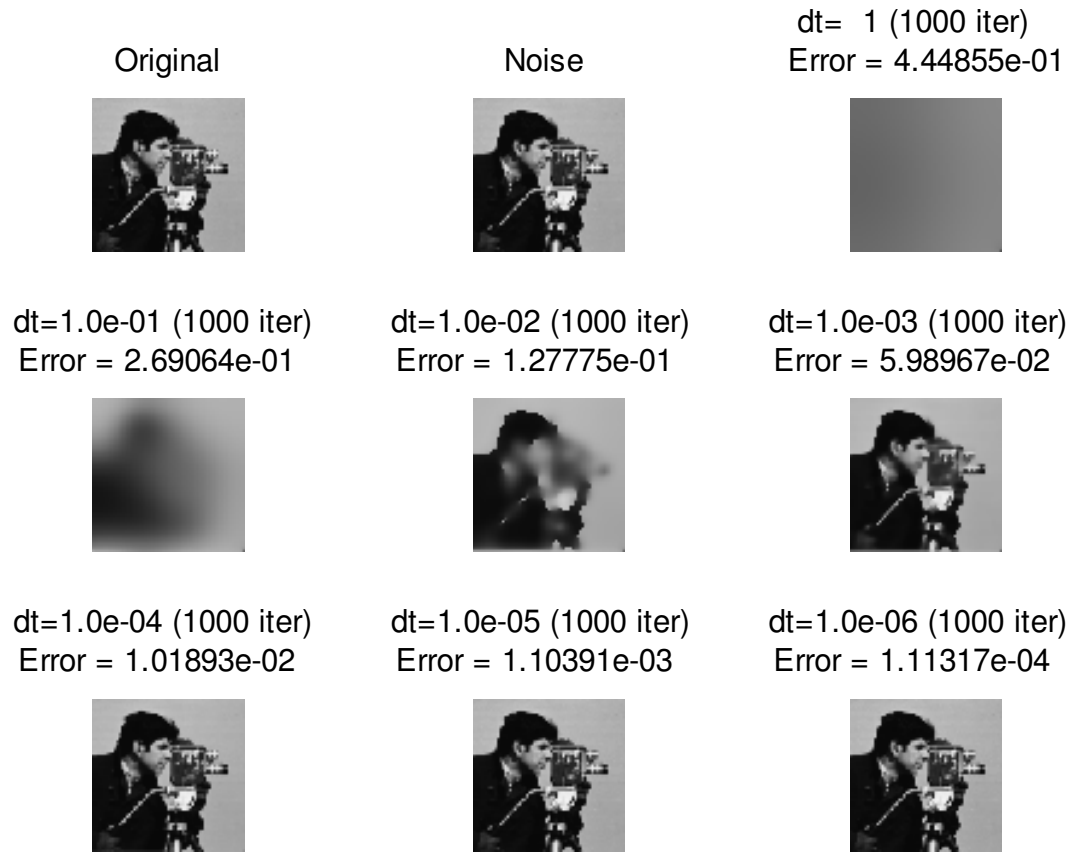


**Figure 33: Explicit PM Visual Results for different values of  $\Delta t$  for processing the original image without any noise added (i.e. “Noise” Image = “Original” Image).**



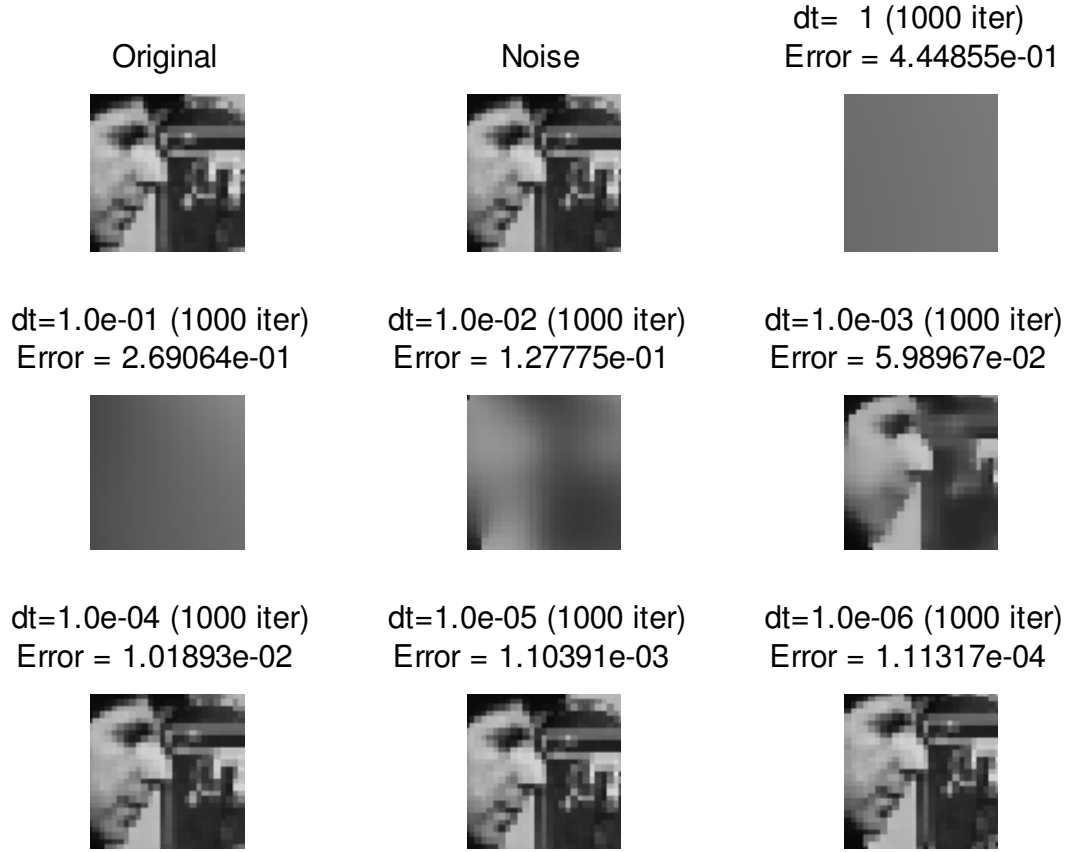
**Figure 34: (ZOOM IN) Explicit PM Visual Results for different values of  $\Delta t$  for processing the original image without any noise added (i.e. “Noise” Image = “Original” Image).**

It quickly becomes clear that for the Explicit PM method with the parameters set in Table 1, the image has no meaning until  $\Delta t$  is at least  $1E-2$  or smaller. The Implicit Method demonstrated similar results.



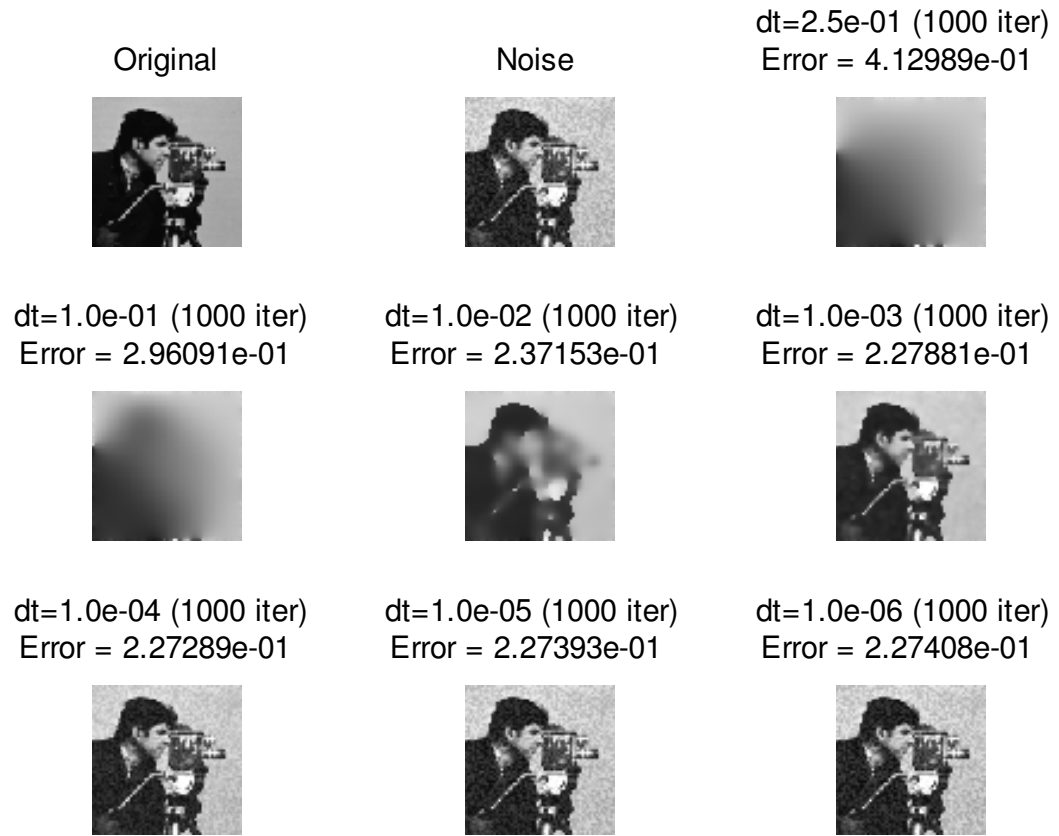
**Figure 35: Implicit PM Visual Results for different values of  $\Delta t$  for processing the original image without any noise added (i.e. “Noise” Image = “Original” Image).**



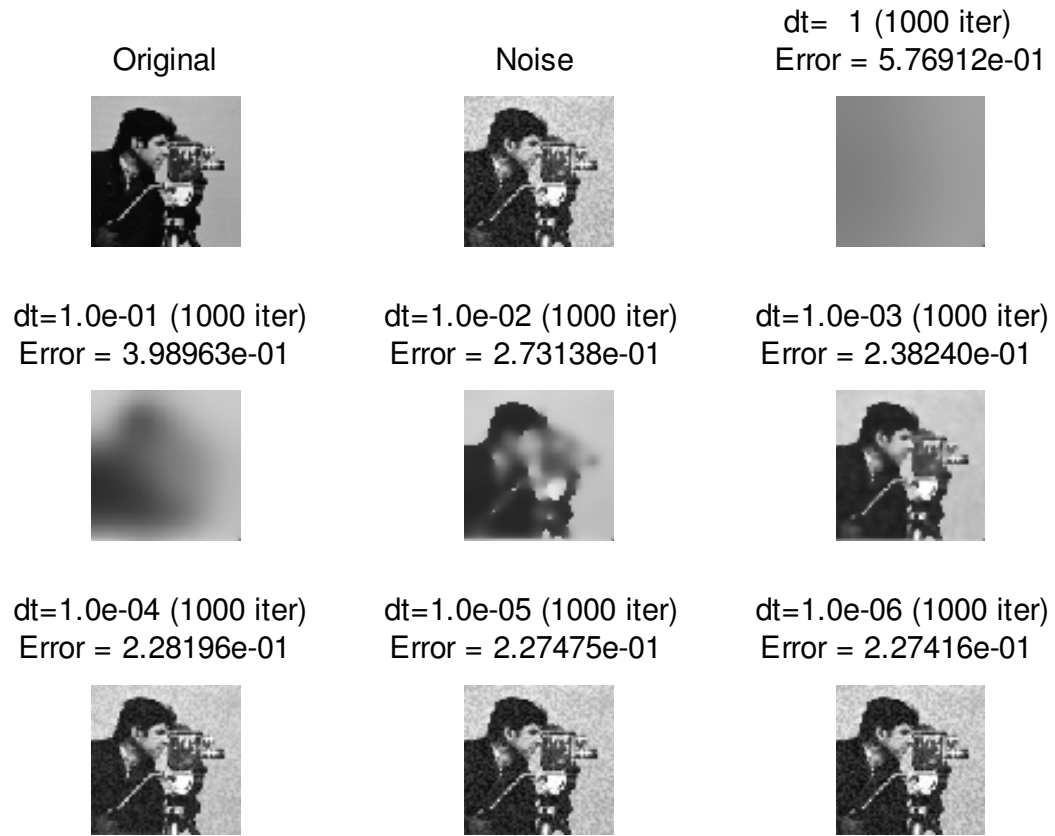


**Figure 36: (ZOOM IN) Implicit PM Visual Results for different values of  $\Delta t$  for processing the original image without any noise added (i.e. “Noise” Image = “Original” Image).**

Given processing on the original image only, it that the optimal choice of  $\Delta t$  wrt error, runtime and visual analysis, is  $\Delta t=1E-04$ . Although, the stability of the scheme has been confirmed and the results are promising, it is essential to examine the same results given degraded imagery. See Section 4.3 for details on the different noise simulations. Our first degradation of the image is by adding random noise:

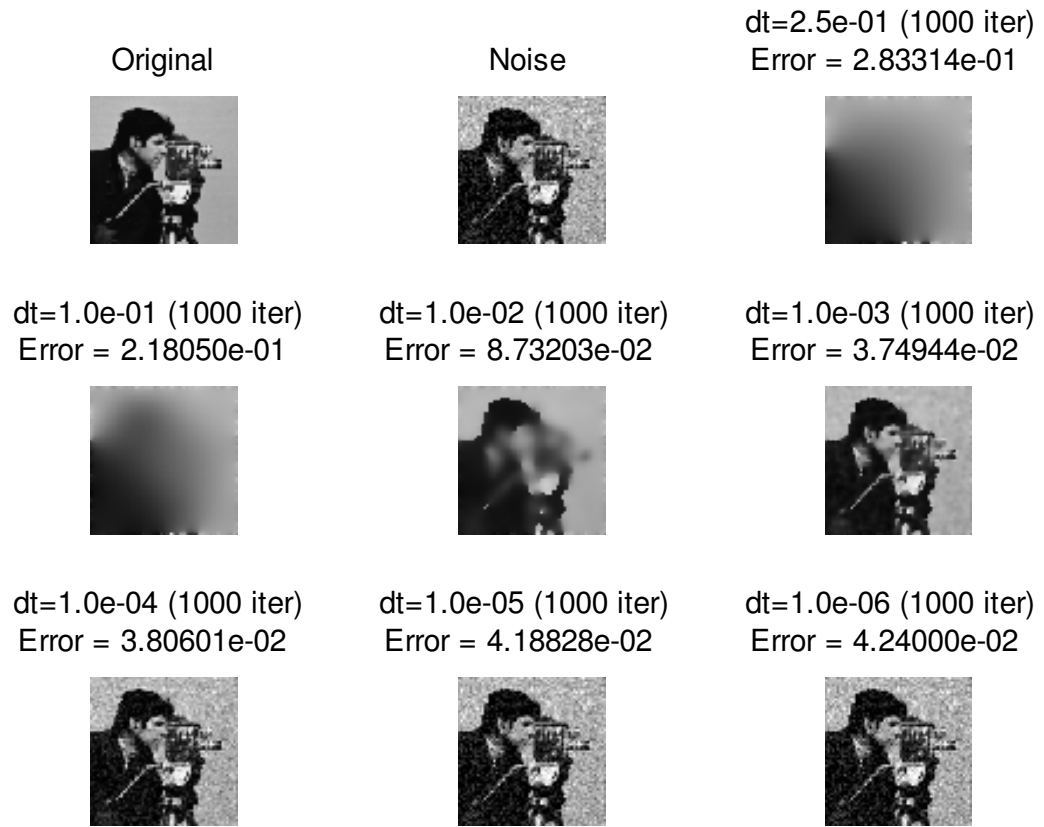


**Figure 37: Explicit PM Visual Results for different values of  $\Delta t$  for processing with RANDOM NOISE added.**

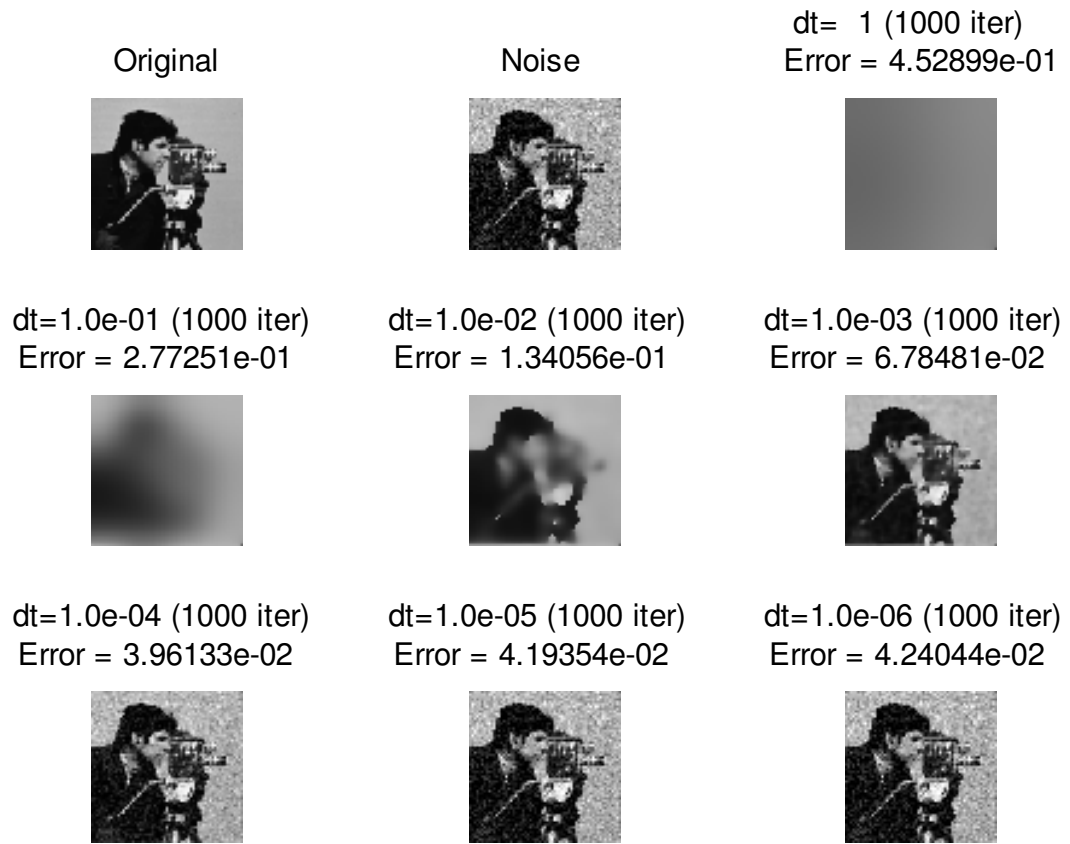


**Figure 38: Implicit PM Visual Results for different values of  $\Delta t$  for processing with RANDOM NOISE added.**

The second method to degrade the image is to add Gaussian noise to the image:

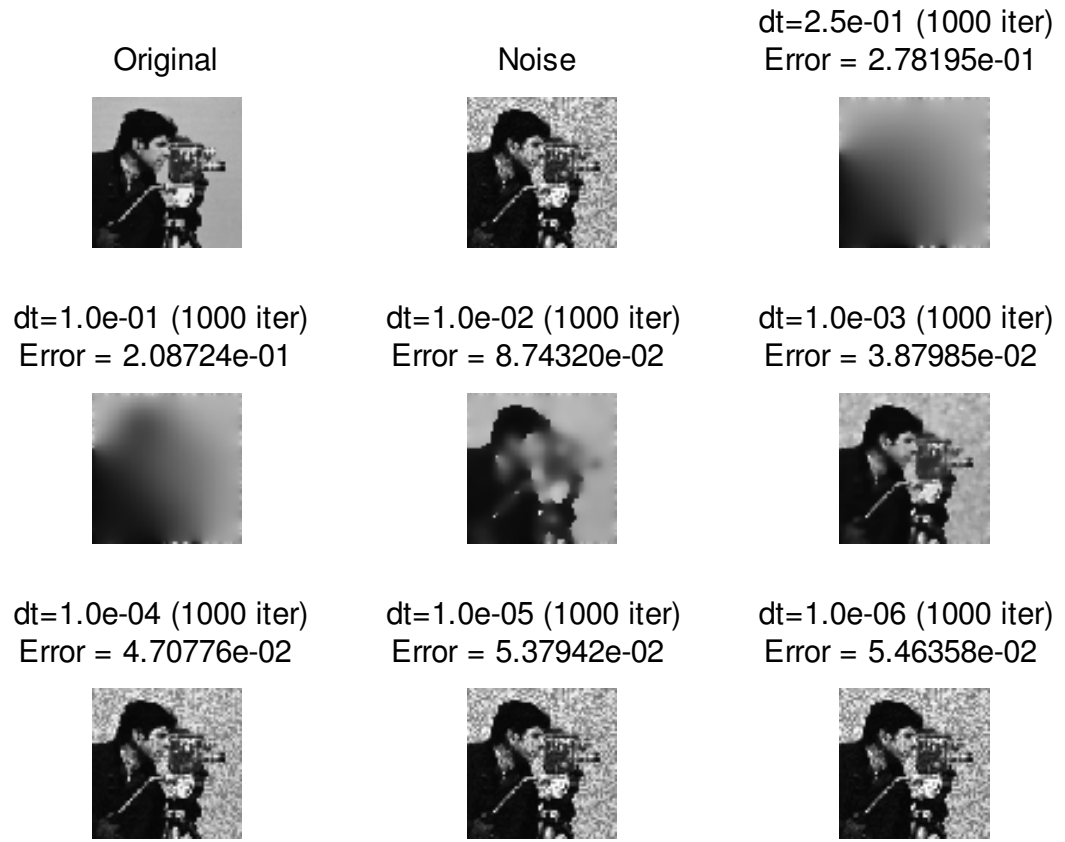


**Figure 39: Explicit PM Visual Results for different values of  $\Delta t$  for processing with GAUSSIAN NOISE added.**



**Figure 40: Implicit PM Visual Results for different values of  $\Delta t$  for processing with GAUSSIAN NOISE added.**

The last degradation of the image is by simulating speckle by multiplying the image by random noise:



**Figure 41: Explicit PM Visual Results for different values of  $\Delta t$  for processing with simulated SPECKLE.**

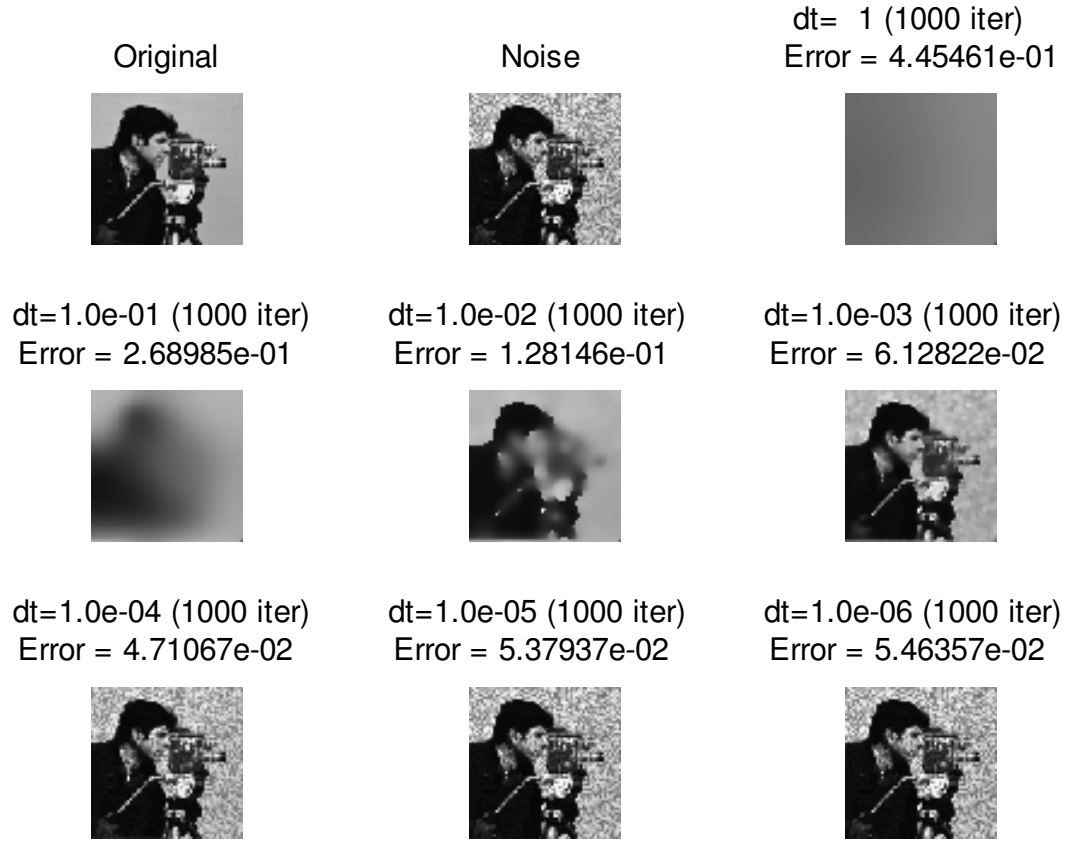


Figure 42: Implicit PM Visual Results for different values of  $\Delta t$  for processing with with simulated SPECKLE.

Experiments to investigate noise used the parameters in Table 1. Visual analysis of the final image for both explicit and implicit put preference on  $\Delta t = 1E-3$ .

dt	Relative Errors of Explicit and Implicit PM on Noisy Images					
	Random Noise		Gaussian Noise		Speckle	
	Explicit	Implicit	Explicit	Implicit	Explicit	Implicit
1E-01	2.96E-01	3.99E-01	2.18E-01	2.77E-01	2.09E-01	2.69E-01
1E-02	2.37E-01	2.73E-01	8.73E-02	1.34E-01	8.74E-02	1.28E-01
1E-03	2.28E-01	2.38E-01	3.75E-02	6.78E-02	3.88E-02	6.13E-02
1E-04	2.27E-01	2.28E-01	3.81E-02	3.96E-02	4.71E-02	4.71E-02
1E-05	2.27E-01	2.27E-01	4.19E-02	4.19E-02	5.38E-02	5.38E-02
1E-06	2.27E-01	2.27E-01	4.24E-02	4.24E-02	5.46E-02	5.46E-02
Avg Time (s)	40	1056	38	1064	37	1072

Table 4: Errors and average time of Explicit and Implicit schemes wrt different  $\Delta t$  for simulated noisy images

In contrast to the experiment using the original image demonstrating the theory that the error decreases with decreasing  $\Delta t$  without lower bound is not validated in the cases processing the noisy images. In fact, with  $\Delta t < 1\text{E-}3$ , the error stays the same or gets worse. Also, looking at the different errors for a specific noise source, it shows that the Perona-Malik numerical schemes are most effective against Gaussian and Speckle noise, hence the use in reduction in speckle for Synthetic Aperture Radar (SAR) and ultrasound imagery, where speckle is a common artifact.

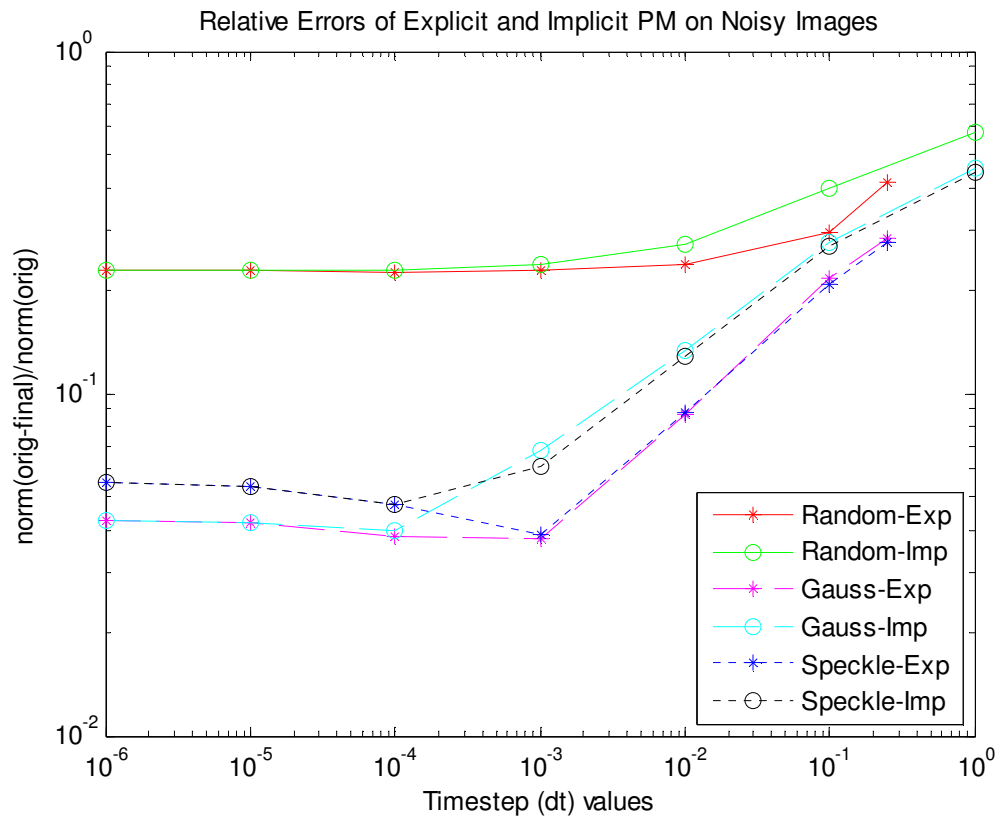


Figure 43: Errors of Explicit (circle) and Implicit (asterix) Methods wrt different  $\Delta t$ .



So now with all experiments reviewed, the optimal (suggested) choice for a time step to run the Explicit and Implicit PM Numerical Schemes is with  $\Delta t = 1E-3$  and not to far less.

Overall run time is an extremely significant factor in the discussion of explicit versus implicit computational schemes. The MATLAB Profiler application was used to examine the timing of the programs. The Implicit PM scheme always takes more time and the majority of the time is spent creating implicit scheme coefficients matrix within pm\_implicit\_matrix.m. Approximately 78% of the total runtime is spent on creating this matrix of coefficients. Future efforts can be made on optimizing the run time for computing this matrix, possibly starting with vectorization rather than looping thru each element.

#### Profile Summary

Generated 27-Apr-2013 00:16:27 using cpu time.






Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
<a href="#">PM_Implicit_Converge</a>	4	2322.314 s	3.387 s	
<a href="#">pm_implicit_dt</a>	28	2317.296 s	498.277 s	
<a href="#">pm_implicit_matrix</a>	28	1818.634 s	1818.634 s	
<a href="#">PM_Explicit_Converge</a>	5	87.419 s	3.370 s	
<a href="#">pm_explicit_dt</a>	29	82.247 s	81.847 s	

Figure 44: MATLAB Profile to view time consumption when running MASTER\_RUN\_PERONA\_MALIK.m script

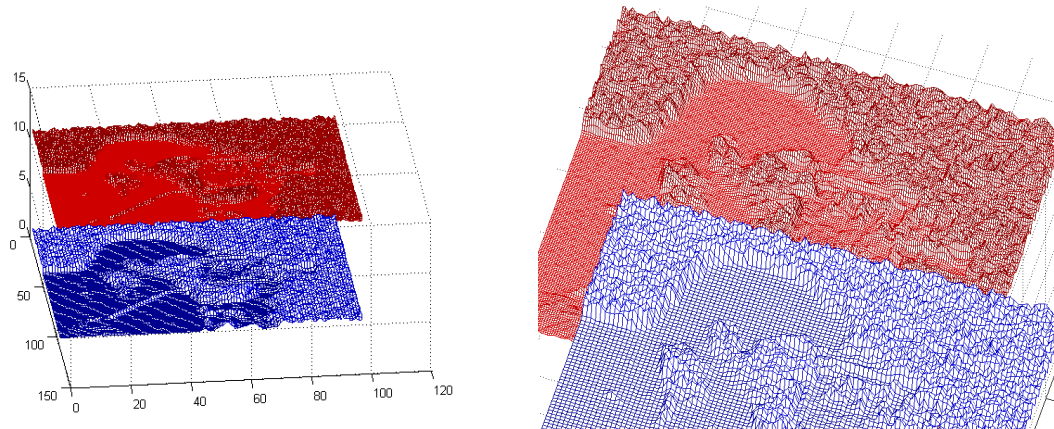
## 5.2 Computational Grid and Interpolation Methods

The numerical schemes were written to allow for potential investigation of different values of  $\Delta x$  and  $\Delta y$  to transform an original input image onto a

computational grid. The methods imply reciprocal fractions, i.e. given integers M and N,  $\Delta x=1/M$  and  $\Delta y=1/N$ .

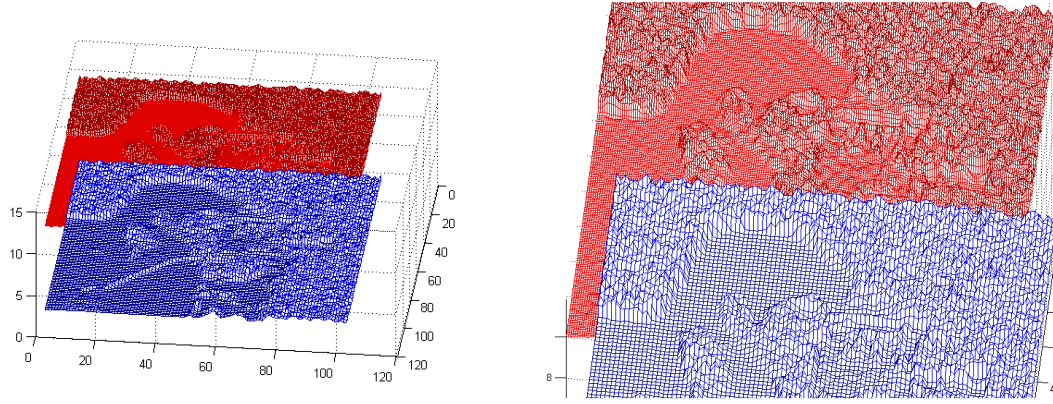
### 5.2.1 Finer Computational Grid: Bilinear vs. Spline Interpolation

For examining different interpolation methods all parameters were held constant (see Table 1) with  $\Delta t=1E-3$ . In order to run the interpolation schemes on a finer grid, a mesh grid space must first be created to step in the x direction from 1 to I<sub>max</sub> and step in the y direction from 1 to J<sub>max</sub>. See below for a visual of the original image as a mesh plotted next to the mesh of the computational grid for the bilinear interpolation:



**Figure 45: Bilinear Interpolation of Original Image (BLUE) to 2x Finer Computational Grid (RED) with  $dx=1/2$  and  $dy=1/2$  and ZOOM (right).**

See below for a visual of the original image as a mesh plotted next to the mesh of the computational grid for the spline interpolation:

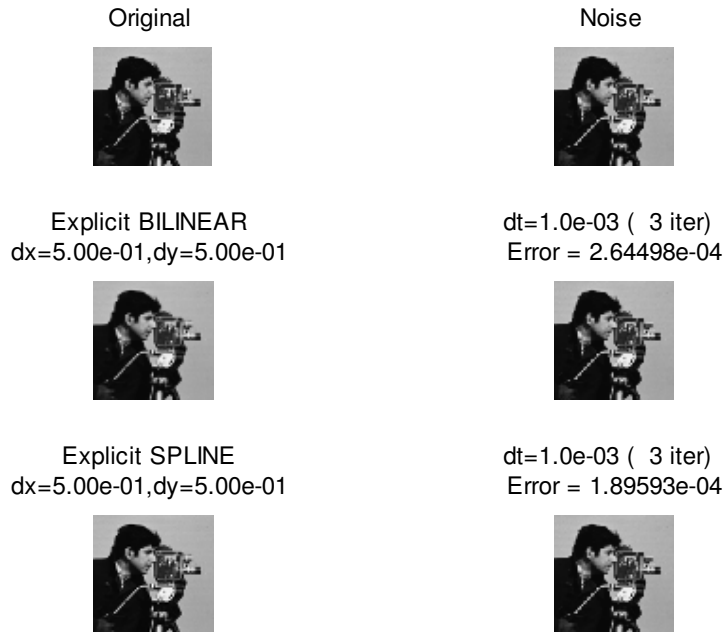


**Figure 46: Spline Interpolation of Original Image (BLUE) to transform to 2x Finer Computational Grid (RED) with  $dx = \frac{1}{2}$  and  $dy = \frac{1}{2}$  with ZOOM (right)**

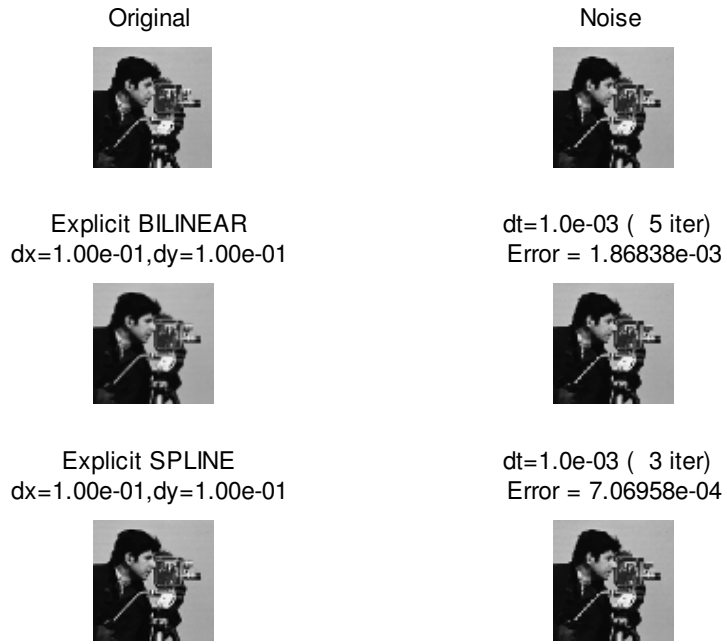
It is not easy to visually see the differences of the computational grids generated from bilinear versus spline interpolation visually. Experiments were run with the original image (no noise added) to compare error results using either bilinear or spline interpolation with fixed  $\Delta t = 1, 1/2, 1/10$ , and  $1/20$ . This analysis was performed with the Explicit PM Method:



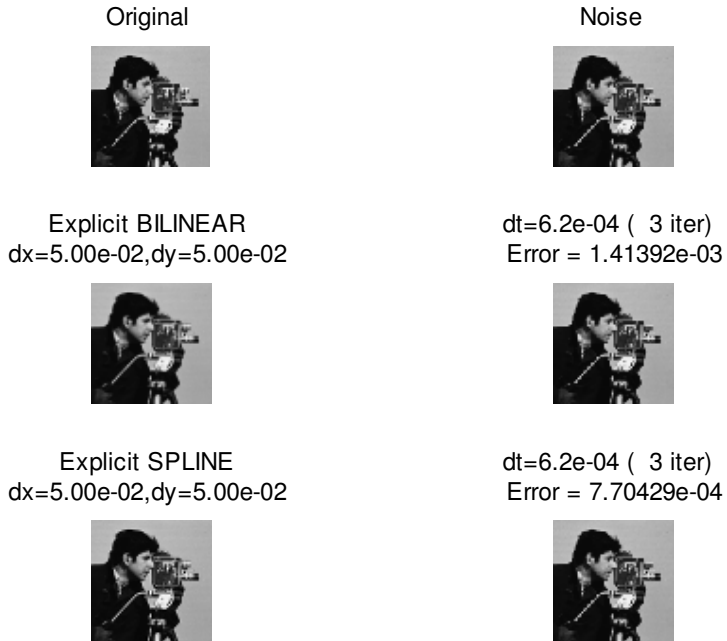
**Figure 47: Explicit PM Visual Results with Bilinear Interpolation (2<sup>nd</sup> row) and Spline Interpolation (3<sup>rd</sup> Row) for  $\Delta x=\Delta y=1$ . No Noise Added. Each row has the final image then the downsampled image.**



**Figure 48: Explicit PM Visual Results with Bilinear Interpolation (2<sup>nd</sup> row) and Spline Interpolation (3<sup>rd</sup> Row) for  $\Delta x=\Delta y=1/2$ . No Noise Added. Each row has the final image then the downsampled image.**



**Figure 49: Explicit PM Visual Results with Bilinear Interpolation (2<sup>nd</sup> row) and Spline Interpolation (3<sup>rd</sup> Row) for  $\Delta x=\Delta y=1/10$ . No Noise Added. Each row has the final image then the downsampled image.**



**Figure 50: Explicit PM Visual Results with Bilinear Interpolation (2<sup>nd</sup> row) and Spline Interpolation (3<sup>rd</sup> Row) for  $\Delta x=\Delta y=1/20$ . No Noise Added. Each row has the final image then the downsampled image.**

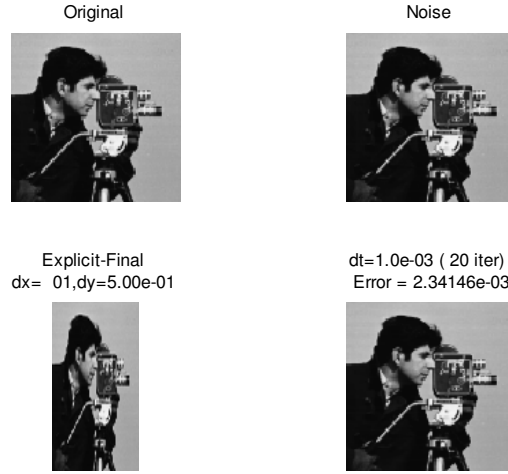
Below are the numerical results consolidated into a table. From the results, we can see that for processing the original image with  $\Delta t = 1\text{E-}3$ , the SPLINE interpolation yields better results for the Explicit PM Scheme.

I <sub>max</sub>	J <sub>max</sub>	dx=dy	BILINEAR		SPLINE	
			Rel. Error	Time (s)	Rel. Error	Time (s)
101	101	1	1.9141E-04	0.2	1.9141E-04	0.2
201	201	1/2	2.6450E-04	0.6	1.8959E-04	0.5
1001	1001	1/10	1.8684E-03	27.2	7.0696E-04	16.4
2001	2001	1/20	1.4139E-03	101.1	7.7043E-04	101.0

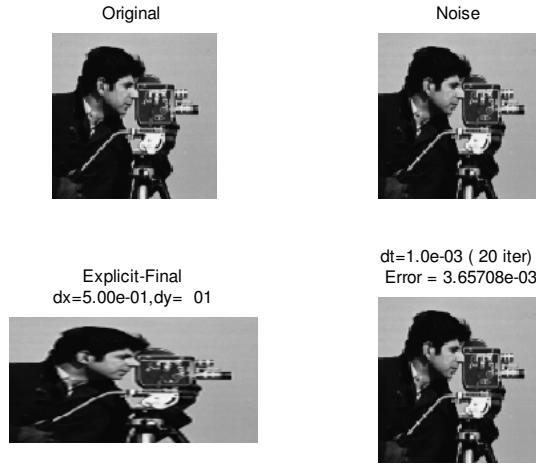
Table 5: Explicit PM Numerical Results for Bilinear and Spline Interpolation with different  $\Delta x = \Delta y$  values

### 5.2.2 Non-Uniform Computational Grid

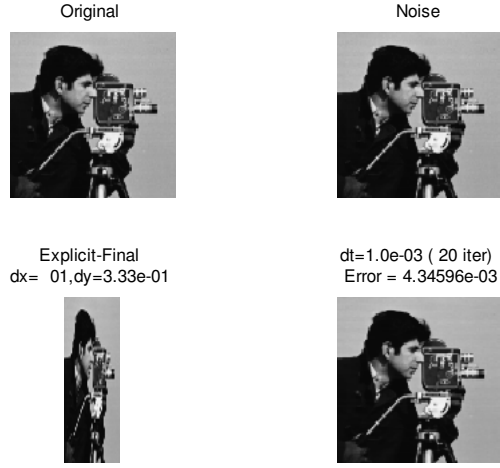
In this thesis the focus was on uniform grid spaces but all schemes were written to allow for running experiments on a non-uniform grid space (i.e. where  $\Delta x$  does not equal  $\Delta y$ ). See examples below for preliminary results, to demonstrate the functionality using SPLINE interpolation and  $\Delta t = 1\text{E-}3$ :



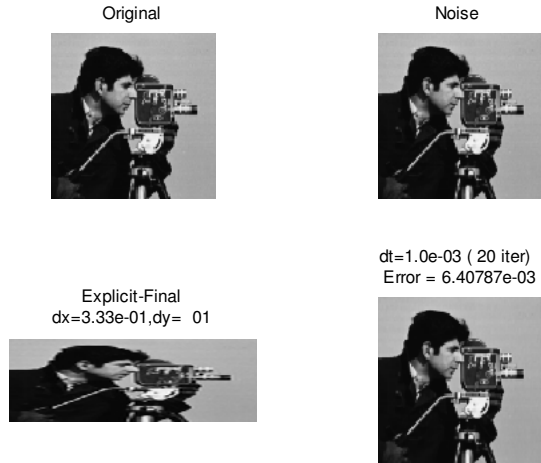
**Figure 51: Explicit PM on Non-Uniform Computational Grid with  $\Delta x=1$  and  $\Delta y=1/2$ . Spline Interpolation. No Noise Added. Bottom Left is the final image (larger image (i.e. more elements since dx and dy are  $< 1$ )). Bottom Right is the downsampled image (back to original size)**



**Figure 52: Explicit PM on Non-Uniform Computational Grid with  $\Delta x=1/2$  and  $\Delta y=1$ . Spline Interpolation. No Noise Added. Bottom Left is the final image (larger image (i.e. more elements since dx and dy are  $< 1$ )). Bottom Right is the downsampled image (back to original size)**

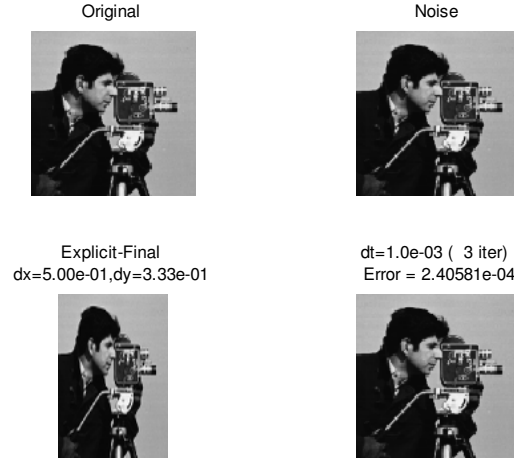


**Figure 53: Explicit PM on Non-Uniform Computational Grid with  $\Delta x=1$  and  $\Delta y=1/3$ . Spline Interpolation. No Noise Added. Bottom Left is the final image (larger image (i.e. more elements since  $dx$  and  $dy$  are  $< 1$ )). Bottom Right is the downsampled image (back to original size)**

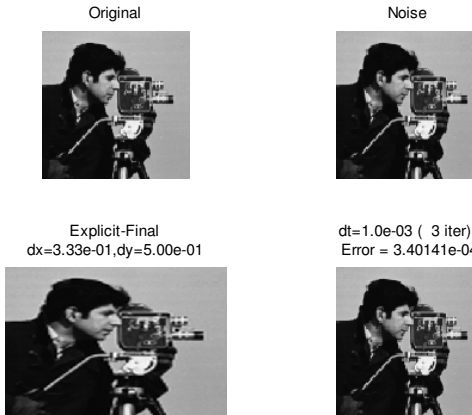


**Figure 54: Explicit PM on Non-Uniform Computational Grid with  $\Delta x=1/3$  and  $\Delta y=1$ . Spline Interpolation. No Noise Added. Bottom Left is the final image (larger image (i.e. more elements since  $dx$  and  $dy$  are  $< 1$ )). Bottom Right is the downsampled image (back to original size)**





**Figure 55: Explicit PM on Non-Uniform Computational Grid with  $\Delta x = \Delta y = 1/20$ . Spline Interpolation. No Noise Added. Bottom Right is the final image (larger=more elements). Bottom Left is the downsampled image (back to original size)**



**Figure 56: Explicit PM on Non-Uniform Computational Grid with  $\Delta x = 1$  and  $\Delta y = 1/2$ . Spline Interpolation. No Noise Added. Bottom Left is the final image (larger image (i.e. more elements since dx and dy are  $< 1$ )). Bottom Right is the downsampled image (back to original size)**

The optimal choice of a non-uniform grid spacing will depend on the details of the original input image or what dimension needs to be emphasized in computation. Analysis of the few results summarized below shows that for this

subset of the cameraman.tif image, the Explicit PM numerical scheme with SPLINE interpolation, generally more accurate when  $\Delta x > \Delta y$ .

<b>I<sub>max</sub></b>	<b>J<sub>max</sub></b>	<b>dx</b>	<b>dy</b>	<b>dt</b>	<b># Iter</b>	<b>Rel. Error</b>	<b>Iter Error</b>	<b>Time (s)</b>
201	101	1	1/2	0.001	20	2.3415E-03	1.1080E-04	5.2
101	201	1/2	1	0.001	20	3.6571E-03	1.6896E-04	3.5
301	101	1	1/3	0.001	20	4.3460E-03	1.9623E-04	3.7
101	301	1/3	1	0.001	20	6.4079E-03	2.7931E-04	3.5
301	201	1/2	1/3	0.001	3	2.4058E-04	7.2920E-05	0.8
201	301	1/3	1/2	0.001	3	3.4014E-04	9.5883E-05	1.2

**Table 6: Explicit PM Scheme on Non-Uniform Grid Spacing**

## CHAPTER VI – CONCLUSION AND FUTURE WORK

Two numerical methods were investigated for implementing the Perona-Malik equation for image denoising. The Explicit PM Scheme that is widely used in the image processing world, is well-known. In this paper we were able to identify exactly what factors contributed to a stable solution when running the explicit method. The implicit numerical scheme that was examined proved to have much more theoretically sound stability without any conditions on  $\Delta t$ . Unfortunately, with regards to computational time metrics, it is extremely hard to make the argument to employ the implicit method due to its extremely high cost in terms of computation time. It is a very slow method. Dependent upon the desired end result, parameters chosen and computing platform, a user may be willing to expend the time cost for the more stable solution from the implicit scheme. Future work on porting the implicit computation onto a high performance computing platform may hold promise in bringing down the runtimes. Extending the research of the Perona-Malik equation and other nonlinear diffusion functions via other stable methods such as Crank-Nicolson may yield a better trade-off between quality, stability and run-time., especially if implementing the Alternating Direction Implicit Method. Running on user-defined computational grid spaces that are finer, coarser or non-uniform also opens doors to a wide array of experiments tailorable to the unique attributes of the

input image. Understanding the theoretical stability and convergence properties of the numerical schemes being implemented will help shape expectations of various algorithms. The immediate plan after this work would be to examine the implicit numerical schemes for the fourth order nonlinear diffusion models for image noise reduction, especially those presented by You & Kaveh [2] and Hajiaboli [3].

## APPENDIX A – MATLAB FOR EXPLICIT PM NUMERICAL SCHEME

```
function [im_final varargout] =
pm_explicit(im_orig,kval,dx,dy,dt,nsteps,tol,option_g,varargin)
%
% Perona-Malik Explicit Code
%
% FUNCTION      PM_EXPLICIT_LOOP
%
% USAGE         [im_final <dt> <n> iter_err]=pm_explicit_loop(im_orig,
%                                                             kval,
%                                                             dx,dy,dt,
%                                                             nsteps,tol
%                                                             option_g,
%                                                             <verbose>)
%
% INPUTS      im_orig      = original image that requires smoothing,
%                               denoising, despeckle, etc
%              kval        = constant that scales down the variable of the
%                               g function
%              dx          = step size in x direction (delta x) used to
%                               create computational grid, currently a
%                               scalar (may be a function in future)
%              dy          = step size in y direction (delta y) used to
%                               create computational grid, currently a
%                               scalar (may be a function in future)
%              dt          = step size in time (delta t) used to create
%                               matrix operator used to solve the implicit
%                               system.
%                               Restricted to (dx^2)*(dy^2)/(2*(dx^2 + dy^2))
%              nsteps      = number of steps in t, i.e. the maximum
%                               allowed number of iterations (or solutions
%                               to the PDE), if the image does not converge
%              tol         = tolerance for image error between iterations
%                               since ||image(n+1)|| < ||image(n)||, so as
%                               the iterations continue the image doesn't
%                               changes as much. We stop as soon as the rel
%                               error < tolerance
%              option_g    = choice of g function (hard-coded)
%                               1 ==> exp(-(nrm_gradI/kval).^2)
%                               2 ==> 1./(1+(nrm_gradI/kval).^2)
%                               otherwise gaussian filter with coeffiecents
%                               = 1
% (optional) verbose      = 'verbose' or 'v' to display plots of the
```

```

%                                     various images created in the PM process
%
% OUTPUTS      im_final      = original image that requires smoothing,
%                               denoising, despecklye, etc
% (optional) dt      = return the dt used, in case dt was greater
%                               than stability condition, it will be set to
%                               99% of the maximum allowed value
% (optional) n      = final number of iterations run to reach tol
%                               or if tol was not reached, n = nsteps
% (optional) iter_err  = relative error of last two images at final
%                               iteration = norm(im_prev-im_pad)/norm(im_prev)
%
%
verbose = 'none';

if length(varargin) >= 1
    verbose = varargin{1};
end

% Calculate delta-time
dtmax = (dx^2)*(dy^2)/(2*(dx^2 + dy^2));

if dt >= dtmax
    dtbad = dt;
    dt = dtmax - abs(dtmax)/100;
    dterr=sprintf('dt = %d > dtmax = %d, default dt = %d (99pct of
dtmax)',...
    dtbad,dtmax,dt);
    disp(dterr)
end

% lambda1 = dt/(dx^2);
% lambda2 = dt/(dy^2);

% Convert image to double
im_orig = im2double(im_orig);

% Get dimensions of the image
[Iorig_max Jorig_max] = size(im_orig);

if strcmpi(verbose,'v') || strcmpi(verbose,'verbose')
    % In verbose mode, create figure of different images created
    figure(1), imshow(im_orig), title('Original')
    % figure, subplot 231, imshow(im_orig), title('Original')
end

% Create mesh of original image space and create mesh based on dx,dy
steps
[im_origX,im_origY] = meshgrid(1:Iorig_max,1:Jorig_max);
[im_compX,im_compY] = meshgrid(1:dx:Iorig_max,1:dy:Jorig_max);

% Use interpolation to create the intensities of the computational grid

```

```

im_comp =
interp2(im_origX,im_origY,im_orig,im_compX,im_compY,'linear');

% figure,mesh(im_origX,im_origY,im_orig)
% hold on
% mesh(im_compX,im_compY,im_comp+50)
% hold off

[Imax,Jmax] = size(im_comp);

if strcmpi(verbose,'v') || strcmpi(verbose,'verbose')
    % In verbose mode, create figure of different images created
    figure(2), imshow(im_comp), title('Comput. Grid')
end

im_pad = zeros(Imax+2,Jmax+2);
im_pad(2:Imax+1,2:Jmax+1) = im_comp;
im_pad(1,2:Jmax+1) = im_pad(3,2:Jmax+1);
im_pad(Imax+2,2:Jmax+1) = im_pad(Imax,2:Jmax+1);
im_pad(2:Imax+1,1) = im_pad(2:Imax+1,3);
im_pad(2:Imax+1,Jmax+2) = im_pad(2:Imax+1,Jmax);

if strcmpi(verbose,'v') || strcmpi(verbose,'verbose')
    % In verbose mode, create figure of different images created
    figure(3), imshow(im_pad), title('Padded CompGrid')
end

% Solve for new intensity using only interior of padded computational
grid
for n=1:nsteps

    % Find c at each point, in order to find c at each point we need

    % Calculate the gradient of the intensities (of the padded image)
    gradI_x = zeros(size(im_pad));
    gradI_y = zeros(size(im_pad));

    for i=2:Imax+1
        for j=2:Jmax+1
            gradI_x(i,j) = (im_pad(i+1,j)-im_pad(i-1,j))/(2*dx);
            gradI_y(i,j) = (im_pad(i,j+1)-im_pad(i,j-1))/(2*dy);
        end
    end

    % Calculate the 2-norm of the gradient of the intensities
    nrm_gradI = sqrt(gradI_x.^2 + gradI_y.^2);

    % Determine and evaluate the g function
    switch option_g
        case 1
            gfun = exp(-(nrm_gradI/kval).^2);

```

```

        case 2
            gfun = 1./(1+(nrm_gradI/kval).^2);
        otherwise
            gfun = ones(size(nrm_gradI));           % Gaussian Kernel
        end

        cval = gfun;           % Is this step necessary???

        if strcmpi(verbose,'v') || strcmpi(verbose,'verbose')
            % In verbose mode, create figure of different images created
            figure(4), imshow(cval), title('C Values')
        end

        % CHECK for no change between previous and current timesteps (<
tol)
        im_prev = im_pad;

        for i=2:Imax+1
            for j=2:Jmax+1
                % Perform all calculations wrt X
                A1 = (cval(i+1,j)+cval(i,j))*(im_pad(i+1,j)-im_pad(i,j))/2;
                A2 = (cval(i,j)+cval(i-1,j))*(im_pad(i,j)-im_pad(i-1,j))/2;
                A = dt*(A1 - A2)/dx^2;

                % Perform all calculations wrt Y
                B1 = (cval(i,j+1)+cval(i,j))*(im_pad(i,j+1)-im_pad(i,j))/2;
                B2 = (cval(i,j)+cval(i,j-1))*(im_pad(i,j)-im_pad(i,j-1))/2;
                B = dt*(B1 - B2)/dy^2;

                im_pad(i,j) = im_pad(i,j) + A + B;
            end
        end

        iter_err = norm(im_prev - im_pad)/norm(im_prev);
        if iter_err < tol && n > 2
            break
        end
    end

    im_final = im_pad(2:Imax+1,2:Jmax+1);
    results=sprintf('Total Iterations = %3d (dt = %3.1d), Iteration Error =
    %6.4d (< tol = %5.3d)',...
        n,dt,iter_err,tol);
    disp(results);
    if strcmpi(verbose,'v') || strcmpi(verbose,'verbose')
        % In verbose mode, create figure of different images created
        figure(5), imshow(im_final), title('Final PM')
    end

    varargout{1} = dt;
    varargout{2} = n;

```



```
varargout{3} = iter_err;
```

**Figure 57: MATLAB Function – pm\_explicit\_dt.m – Implementation of Explicit PM Numerical Scheme**

## APPENDIX B - MATLAB FOR IMPLICIT PM NUMERICAL SCHEME

```

function [im_final varargout] =
pm_implicit(im_orig,kval,dx,dy,dt,nsteps,tol,option_g,varargin)
%
% Perona-Malik Implicit Code
%
% FUNCTION          PM_IMPLICIT_LOOP
%
% USAGE             [im_final <dt> <n> <iter_err>]=pm_implicit_loop(im_orig,
%                                                                    kval,
%                                                                    dx,dy,dt,
%                                                                    nsteps,tol
%                                                                    option_g,
%                                                                    <verbose>)
%
% INPUTS            im_orig      = original image that requires smoothing,
%                                denoising, despecklye, etc
%                                kval      = constant that scales down the variable of the
%                                g function
%                                dx        = step size in x direction (delta x) used to
%                                create computational grid, currently a
%                                scalar (may be a function in future)
%                                dy        = step size in y direction (delta y) used to
%                                create computational grid, currently a
%                                scalar (may be a function in future)
%                                dt        = step size in time (delta t) used to create
%                                matrix operator used to solve the implicit
%                                system. No restriction except (0 < dt < 1)
%                                nsteps    = number of steps in t, i.e. the maximum
%                                allowed number of iterations (or solutions
%                                to the PDE), if the image does not converge
%                                tol       = tolerance for image error between iterations
%                                since ||image(n+1)|| < ||image(n)||, so as
%                                the iterations continue the image doesn't
%                                changes as much. We stop as soon as the rel
%                                error < tolerance
%                                option_g  = choice of g function (hard-coded)
%                                1 ==>  exp(-(nrm_gradI/kval).^2)
%                                2 ==>  1./(1+(nrm_gradI/kval).^2)
%                                otherwise gaussian filter with coeffieicients
%                                = 1
% (optional) verbose = 'verbose' or 'v' to display plots of the
%                    various images created in the PM process
% OUTPUTS            im_final    = original image that requires smoothing,

```

```

%                               denoising, despecklye, etc
%(optional) dt                 = same dt as above (kept here for consistency
%                               with pm_explicit_dt.m
%(optional) n                   = final number of iterations run to reach tol
%                               or if tol was not reached, n = nsteps
%(optional) iter_err            = relative error of last two images at final
%                               iteration = norm(im_prev-im_pad)/norm(im_prev)
%

verbose = 'none';

if length(varargin) >= 1
    verbose = varargin{1};
end

% DO NOT HAVE RESTRICTION ON DELTA-T

% Convert image to double
im_orig = im2double(im_orig);

% Get dimensions of the image
[Iorig_max Jorig_max] = size(im_orig);

if strcmpi(verbose,'v') || strcmpi(verbose,'verbose')
    % In verbose mode, create figure of different images created
    figure(1), imshow(im_orig), title('Original')
    %   figure, subplot 231, imshow(im_orig), title('Original')
end

% Create mesh of original image space and create mesh based on dx,dy
steps
[im_origX,im_origY] = meshgrid(1:Iorig_max,1:Jorig_max);
[im_compX,im_compY] = meshgrid(1:dx:Iorig_max,1:dy:Jorig_max);

% Use interpolation to create the intensities of the computational grid
im_comp =
interp2(im_origX,im_origY,im_orig,im_compX,im_compY,'linear');

% figure,mesh(im_origX,im_origY,im_orig)
% hold on
% mesh(im_compX,im_compY,im_comp+50)
% hold off

[Imax,Jmax] = size(im_comp);

if strcmpi(verbose,'v') || strcmpi(verbose,'verbose')
    % In verbose mode, create figure of different images created
    figure(2), imshow(im_comp), title('Comput. Grid')
    %   subplot 232, imshow(im_comp), title('Comput. Grid')
end

```

```

im_pad = zeros(Imax+2,Jmax+2);
im_pad(2:Imax+1,2:Jmax+1) = im_comp;
im_pad(1,2:Jmax+1)      = im_pad(3,2:Jmax+1);
im_pad(Imax+2,2:Jmax+1) = im_pad(Imax,2:Jmax+1);
im_pad(2:Imax+1,1)      = im_pad(2:Imax+1,3);
im_pad(2:Imax+1,Jmax+2) = im_pad(2:Imax+1,Jmax);

if strcmpi(verbose,'v') || strcmpi(verbose,'verbose')
    % In verbose mode, create figure of different images created
    figure(3), imshow(im_pad), title('Padded CompGrid')
    % subplot 233, imshow(im_pad), title('Padded CompGrid')
end

% Do we need im_pad anymore? can we simply use im_comp at this point
% Solve for new intensity by solving  $x = A \backslash b$ 

for n=1:nsteps

    % Find c at each point, in order to find c at each point we need to
    % calculate the gradient of the intensities (of the padded image)
    gradI_x = zeros(size(im_pad));
    gradI_y = zeros(size(im_pad));

    for i=2:Imax+1
        for j=2:Jmax+1
            gradI_x(i,j) = (im_pad(i+1,j)-im_pad(i-1,j))/(2*dx);
            gradI_y(i,j) = (im_pad(i,j+1)-im_pad(i,j-1))/(2*dy);
        end
    end

    % Calculate the 2-norm of the gradient of the intensities
    nrm_gradI = sqrt(gradI_x.^2 + gradI_y.^2);

    % Determine and evaluate the g function
    switch option_g
        case 1
            gfun = exp(-(nrm_gradI/kval).^2);
        case 2
            gfun = 1./(1+(nrm_gradI/kval).^2);
        otherwise
            gfun = ones(size(nrm_gradI)); % Gaussian Kernel
    end

    cval = gfun; % Is this step necessary???

    if strcmpi(verbose,'v') || strcmpi(verbose,'verbose')
        % In verbose mode, create figure of different images created
        figure(4), imshow(cval), title('C Values')
        % subplot 234, imshow(cval), title('C Values')
    end
end
tic

```

```

    % Generate IJxIJ matrix to solve implicit system
    A = pm_implicit_matrix(Imax,Jmax,dt,dx,dy,cval);
    toc

    % CHECK for no change between previous and current timesteps (<
tol)
    im_prev = im_pad;
    %toc

    im_bvec = reshape(im_pad(2:Imax+1,2:Jmax+1),Imax*Jmax,1);
    %toc

    im_xvec = A \ im_bvec;
    %toc

    im_pad(2:Imax+1,2:Jmax+1) = reshape(im_xvec,Imax,Jmax);
    toc

    iter_err = norm(im_prev - im_pad)/norm(im_prev);
    if iter_err < tol && n > 2
        break
    end

end
clear A;

im_final = im_pad(2:Imax+1,2:Jmax+1);
results=sprintf('Total Iterations = %3d (dt = %3.1d), Iteration Error =
%6.4d (< tol = %5.3d)',...
    n,dt,iter_err,tol);
disp(results);
if strcmpi(verbose,'v') || strcmpi(verbose,'verbose')
    % In verbose mode, create figure of different images created
    figure(5), imshow(im_final), title('Final PM')
    % subplot 235, imshow(im_final), title('Final PM')
end

varargout{1} = dt;
varargout{2} = n;
varargout{3} = iter_err;

```

**Figure 58: MATLAB Function – pm\_implicit\_dt.m – Implementation of Implicit PM Numerical Scheme**

## APPENDIX C – MATLAB SCRIPT TO RUN EXPERIMENTS WITH 1000 ITERATIONS

```
% Script: MASTER_RUN_PERONA_MALIK_1000_2.m
% Master script to run various

clear all, close all, diary off;
%%

profile on

diary MASTER_RUN_PERONA_MALIK_Run_1000.txt
format longeng

%%
nsteps = 1000;
tol = 1e-9;
im_cam = imread('cameraman.tif');

subX = 30:130;
subY = 80:180;
im_orig = im_cam(subX,subY);
mypmresults_sub.im_orig = im_orig;

figure, imshow(im_orig)

im_cam_dbl = im2double(im_cam);

noise_factor = 1/5;
noise_floor = rand(size(im_cam_dbl))*noise_factor;
im_cam_noise = im_cam_dbl + noise_floor;

im_cam_gaussian = imnoise(im_cam, 'gaussian');

im_cam_speckle = imnoise(im_cam, 'speckle');

%% Run ORIGINAL IMAGE
fignum = 500;
PM_Explicit_Run
exp_results{1} = errvals;

clear im_noise
fignum = 550;
```

```

PM_Implicit_Run
imp_results{1} = errvals;

% Random Additive Noise directly to chip

im_noise = im_cam_noise(subX,subY);
mypmresults_sub.im_addnoise = im_noise;

fignum = 600;
PM_Explicit_Run
figure(gcf),title('PM Explicit: loglog(error) with RANDOM NOISE (CHIP-
ADD)')
exp_results{2} = errvals;

fignum = 650;
PM_Implicit_Run
figure(gcf),title('PM Implicit: loglog(error) with RANDOM NOISE (CHIP-
ADD)')
imp_results{2} = errvals;

%% Gaussian noise
im_noise = im_cam_gaussian(subX,subY);
mypmresults_sub.im_gaussian = im_noise;

fignum = 700;
PM_Explicit_Run
figure(gcf),title('PM Explicit: loglog(error) with GAUSSIAN NOISE')
exp_results{3} = errvals;

fignum = 750;
PM_Implicit_Run
figure(gcf),title('PM Implicit: loglog(error) with GAUSSIAN NOISE')
imp_results{3} = errvals;

%% Speckle (Multiplicative Noise)
im_noise = im_cam_speckle(subX,subY);
mypmresults_sub.im_speckle = im_noise;

fignum = 800;
PM_Explicit_Run
figure(gcf),title('PM Explicit: loglog(error) with SPECKLE NOISE')
exp_results{4} = errvals;

fignum = 850;
PM_Implicit_Run
figure(gcf),title('PM Explicit: loglog(error) with SPECKLE NOISE')
imp_results{4} = errvals;

mypmresults_sub.explicit = exp_results;
mypmresults_sub.implicit = imp_results;

```

```
diary off

profile viewer
```

**Figure 59: MATLAB Script – MASTER\_RUN\_PERONA\_MALIK\_1000.m**

```
diary PM_Explicit_Run_1000.txt

disp('=====');
disp(' ');
disp('Running PM_Explicit_Converge.m Script to view decline in error
rates');

% Check that the im_noise image exists, if not, use std cameraman image
if exist('im_orig') == 0
    im_orig = imread('cameraman.tif');
    im_orig = im_orig(1:174,1:174);
    disp('No original image was identified. Using cameraman.tif');
else
    disp('An original image in memory was identified. Using im_orig');
end

if exist('im_noise') == 0
    im_noise = im_orig;
    disp('No noisy image was identified. Using im_orig directly');
    myloglogtitle = sprintf('PM Explicit: loglog(error) with NO NOISE -
original');
    %fignum = 100;
else
    disp('A noisy image in memory was identified. Using im_noise');
    myloglogtitle = sprintf('PM Explicit: loglog(error) with NOISE
added');
    %fignum = 200;
end

% Set parameters for Perona Malik Explicit Scheme
if exist('nsteps') == 0
    nsteps = 10;
end
if exist('tol') == 0
    tol = 1e-8;
end
```



```

kval    = 1/7;
option  = 1;

dx      = 1;
dy      = 1;

disp(sprintf('Parameters:\n - K = %4.2d\n - Max N Steps = %4d\n -
Option %1d (choice of g)\n - Stepsizes: dx = %d, dy = %d',...
    kval,nsteps,option,dx,dy));
disp(sprintf(' - Tolerance = %d (between successive iterations)',tol));
disp('-----');

dvals = 0:6;
num_dt = length(dvals);

num_plots = num_dt+2;
figrows = ceil(sqrt(num_plots));
figcols = ceil(sqrt(num_plots));

if exist('fignum') == 0
    fignum = 101
end
figure(fignum), subplot(figrows,figcols,1), imshow(im_orig),
title('Original')
figure(fignum), subplot(figrows,figcols,2), imshow(im_noise),
title('Noise')

i = 0;
for d = dvals
    i = i+1;
    dt = 10^(-d);

    tstart = tic;

    [im_final exp_dt exp_n(i) exp_ierr(i)] =
pm_explicit(im_noise,kval,dx,dy,dt,nsteps,tol,option);
    %[im_final exp_dt exp_n(i) exp_ierr(i)] =
pm_explicit_dt(im_noise,kval,dx,dy,dt,nsteps,option,tol,'v');

    % Collect the dt values in a single array
    dtvals(i) = exp_dt;

    % Downsample final image to return to original image size
    im_down = downsample(downsample(im_final',ceil(1/dx))',ceil(1/dy));

    % Calculate error
    exp_err(i) = norm(im2double(im_orig)-
im_down)/norm(im2double(im_orig));

    exptime(i) = toc(tstart);

    figure(fignum), subplot(figrows,figcols,i+2), imshow(im_down), ...

```

```

        title(sprintf('dt=%3.1d (%3d iter) \n Error =
%7.5d',exp_dt,exp_n(i),exp_err(i)));
    end

    format shorteng
    disp('errvals = [dt-values #Iterations FinalError IterError
Time(sec)]');
    errvals = [dtvals' exp_n' exp_err' exp_ierr' exptime']
    figure(fignum+1), loglog(dtvals,exp_err), title(myloglogtitle),...
        xlabel('Timestep values'),ylabel('norm(orig-final)/norm(orig)')

    diary off

```

**Figure 60: MATLAB Script - PM\_Explicit\_Run.m**

```

diary PM_Implicit_Run_1000.txt

disp('=====');
disp(' ');
disp('Running PM_Implicit_Converge.m Script to view decline in error
rates');

% Check that the im_noise image exists, if not, use std cameraman image
if exist('im_orig') == 0
    im_orig = imread('cameraman.tif');
    im_orig = im_orig(1:174,1:174);
    disp('No original image was identified. Using cameraman.tif');
else
    disp('An original image in memory was identified. Using im_orig');
end

if exist('im_noise') == 0
    im_noise = im_orig;
    disp('No noisy image was identified. Using im_orig directly');
    myloglogtitle = sprintf('PM Implicit: loglog(error) with NO NOISE -
original');
else
    disp('A noisy image in memory was identified. Using im_noise');
    myloglogtitle = sprintf('PM Implicit: loglog(error) with NOISE');
end

% Set parameters for Perona Malik Implicit Scheme
if exist('nsteps') == 0
    nsteps = 10;

```

```

end
if exist('tol') == 0
    tol = 1e-8;
end
kval    = 1/7;
option  = 1;

dx      = 1;
dy      = 1;

disp(sprintf('Parameters:\n - K = %4.2d\n - Max N Steps = %4d\n -
Option %ld (choice of g)\n - Stepsizes: dx = %d, dy = %d',...
    kval,nsteps,option,dx,dy));
disp(sprintf(' - Tolerance = %d (between successive iterations)',tol));
disp('-----');

dvals = 0:6;
num_dt = length(dvals);

num_plots = num_dt+2;
figrows = ceil(sqrt(num_plots));
figcols = ceil(sqrt(num_plots));

if exist('fignum') == 0
    fignum = 99
end
figure(fignum), subplot(figrows,figcols,1), imshow(im_orig),
title('Original')
tic
figure(fignum), subplot(figrows,figcols,2), imshow(im_noise),
title('Noise')
toc
i = 0;
for d = dvals
    i = i+1;
    dt = 10^(-d);

    tstart = tic;

    [im_final imp_dt imp_n(i) imp_ierr(i)] =
pm_implicit(im_noise,kval,dx,dy,dt,nsteps,tol,option);
    %[im_final imp_dt imp_n(i) imp_ierr(i)] =
pm_implicit_dt(im_noise,kval,dx,dy,dt,nsteps,option,tol,'v');

    % Collect the dt values in a single array
    dtvals(i) = imp_dt;

    % Downsample final image to return to original image size
    im_down = downsample(downsample(im_final',ceil(1/dx))',ceil(1/dy));

    % Calculate error
    imp_err(i) = norm(im2double(im_orig)-
im_down)/norm(im2double(im_orig));

```

```

    imptime(i) = toc(tstart);

    figure(fignum), subplot(figrows,figcols,i+2),imshow(im_down),...
        title(sprintf('dt=%3.1d (%3d iter) \n Error =
%7.5d',imp_dt,imp_n(i),imp_err(i)));
end

format shorteng
disp('errvals = [dt-values #Iterations FinalError IterError
Time(sec)]');
errvals = [dtvals' imp_n' imp_err' imp_ierr' imptime']
figure(fignum+1), loglog(dtvals,imp_err), title(myloglogtitle),...
    xlabel('Timestep values'),ylabel('norm(orig-final)/norm(orig)')

diary off

```

**Figure 61: MATLAB Script - PM\_Implicit\_Run.m**

## REFERENCES

- [1] P. & M. J. Perona, "Scale-Space and Edge Detection Using Anisotropic Diffusion," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 12, No. 7, pp. 629-639, 1990.
- [2] Y.-L. e. a. You, "Behavioral analysis of anisotropic diffusion in image processing," *IEEE Transactions on Image Processing*, vol. 5, p. 1539–1553, 1996.
- [3] M. R. Hajiaboli, "An Anisotropic Fourth-Order Partial Differential Equation for Noise Removal," Concordia University, Montreal, Canada, 2009.
- [4] R. Van Den Boomgaard, "Algorithms for Non-Linear Diffusion: Matlab in a Literate Programming Style.," Intelligent Sensory Information Systems, University of Amsterdam., The Netherlands.
- [5] USC GEOL557: Modeling Earth Systems, "Two-dimensional heat equation with FD," [Online]. Available: [http://geodynamics.usc.edu/~becker/teaching/557/problem\\_sets/problem\\_set\\_fd\\_2dheat.pdf](http://geodynamics.usc.edu/~becker/teaching/557/problem_sets/problem_set_fd_2dheat.pdf). [Accessed April 2013].
- [6] "Chapter 7: The Diffusion Equation," [Online]. Available: <http://pauli.uni-muenster.de/tp/fileadmin/lehre/NumMethoden/WS0910/ScriptPDE/Heat.pdf>. [Accessed April 2013].
- [7] "A METHOD FOR SAR SPECKLE REDUCTION BASED ON PARTIAL DIFFERENTIAL EQUATION," *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XXXVII, no. Part B7, 2008.

## **CURRICULUM VITAE**

Armelle S. Franklin graduated from Robinson High School, Fairfax, Virginia, in 2003. She received her Bachelor of Science in Mathematics from Temple University in 2007. In 2008, as an employee of Lockheed Martin Corporation, Armelle was selected to participate in the Engineering Leadership Development Program and is now a Senior Systems Engineer. Armelle will graduate with her Master of Science in Mathematics from George Mason University in Spring 2013.