Spring 5-2017

Efficient Denoising and Sharpening of Color Images through Numerical Solution of Nonlinear Diffusion Equations

Linh T. Duong
University of Southern Mississippi

The University of Southern Mississippi

Efficient Denoising and Sharpening of Color Images through Numerical Solution of Nonlinear Diffusion Equations

by

Linh Duong

A Thesis
Submitted to the Honors College of
The University of Southern Mississippi
in Partial Fulfillment
of the Requirements for the Degree of
Bachelor of Science
in the Department of Mathematics

May 2017

Approved by

James V. Lambers, Ph.D., Thesis Adviser
Associate Professor of Mathematics

Bernd Schroeder, Ph.D., Chair
Department of Mathematics

Ellen Weinauer, Ph.D., Dean
Honors College

Abstract

The purpose of this project is to enhance color images through denoising and sharpening, two important branches of image processing, by mathematically modeling the images. Modifications are made to two existing nonlinear diffusion image processing models to adapt them to color images. This is done by treating the red, green, and blue (RGB) channels of color images independently, contrary to the conventional idea that the channels should not be treated independently. A new numerical method is needed to solve our models for high resolution images since current methods are impractical. To produce an efficient method, the solution is represented as a linear combination of sines and cosines for easier numerical treatment and then computed by a combination of Krylov subspace spectral (KSS) methods and exponential propagation iterative (EPI) methods. Numerical experiments demonstrate that the proposed approach for image processing is effective for denoising and sharpening.

Key Words: nonlinear diffusion, denoising, sharpening, Krylov subspace spectral methods, exponential propagation iterative methods, Gaussian quadrature, Lanczos iteration

Acknowledgments

I would like to thank my friends and family for their support. I would also like to thank my advisor, Dr. James Lambers, for his constant encouragement and guidance throughout my time at the University of Southern Mississippi. I could not have done this without you all.

Table of Contents

List of Tables

List of Illustrations

Chapter 1

Introduction

The purpose of this project is to enhance color images through denoising and sharpening, two important branches of image processing, by mathematically modeling the images. Denoising is the reduction of noise, which can be defined as small oscillations in color intensity appearing as "fuzziness." Sudden changes in color intensity that clearly show boundaries are edges, and sharpening is the process to make edges more apparent. The enhancement of an image results in a more visually appealing image that is easier to understand. For example, depicted on the left of Figure 1.1 is a noisy image of Downtown Los Angeles, and depicted on the right is the same image after denoising.

Figure 1.1: Denoising of an image of Downtown Los Angeles [14].

Image processing has applications in various fields such as medical imaging, the production of images of the internal body through technology like X-rays, ultrasonography, and magnetic resonance imaging (MRI). Noise and blur are prominent in these images, so image processing is required to aid diagnosis and treatment. Image processing is also heavily used in law enforcement. Noisy, blurry image frames from surveillance videos can be enhanced

to gain information including facial recognition and license plate numbers in the effort to solve crime. With its wide range of applications, image processing is an important area in which advancements must be made.

This project will present improvements made to existing image processing models and their solution methods. It is hypothesized that treating the red, green, and blue (RGB) channels of color images independently, though untraditional, is a viable technique to denoise and sharpen. Furthermore, by applying new efficient computer-based methods to mathematical models designed in this project, image processing is hypothesized to be effective and scalable to higher resolution.

Chapter 2

Literature Review

Among the models for image processing, total variation and nonlinear diffusion are the most widely used. Total variation (TV) image processing [23] reduces the total variation within noisy images since they have a higher amount of variation due to the accumulation of small oscillations that occur with noise. Nonlinear diffusion image processing models an image's pixel values, ranging from 0 for no color to 255 for total color saturation, as a temperature profile undergoing diffusion. The heat equation, a partial differential equation (PDE) with varying thermal conductivity, is used to change the temperature over time. Diffusion forward in time results in a smoothing out effect and, thus, denoising. Diffusion backward in time has the opposite effect and yields sharpening. Both models can be used for grayscale and color images [6, 19], but this project will specifically work with color images. Moreover, this project will focus on nonlinear diffusion since it has the ability to be more effective and faster than TV and to utilize the vast existing knowledge on solving PDEs, equations that relate functions of two or more variables to their partial derivatives [10].

The Perona-Malik equation [22] was the first, and still is the most well-known, nonlinear diffusion image processing model. It introduced simultaneous forward and backward diffusion. However, the Perona-Malik equation has some shortcomings, as its approach for backward diffusion can lead to abrupt variations in color, which in turn produces excessive sharpening and a "cartoonish" effect. The cartoonish effect can be described as how pictures in a cartoon or coloring book can jump from one color to another without a smooth change. The Perona-Malik equation is also ill-posed, or theoretically unsound, meaning the behavior of its solutions (the enhanced images) is not well-understood due

to the lack of supporting theory. Another issue with being ill-posed is that solutions are more difficult to compute because they are highly sensitive to errors in data. The backward diffusion in the Perona-Malik equation can cause small changes, or errors, in data (the input images) to become amplified. Since its development, modifications have been made to the Perona-Malik equation to create improved models. Most of these are well-posed, but the efforts to control its backward diffusion have not been entirely successful. Attempts to regularize the Perona-Malik equation have resulted in unwanted blurring, which is especially adverse for color images [2, 5, 7, 9, 20].

Whereas the previously mentioned methods have changes in how to solve the Perona-Malik equation, Guidotti, Kim, and Lambers' methods have modifications to the actual equation. Their efforts to overcome the shortcomings of the Perona-Malik equation have led to well-posed models [14, 15] that prevent blurring through approaches to regularization that are easier to manage. To date, these models have only been applied to grayscale images, but the advancements in the models suggest that they can be easily adapted to color images by treating their red, green, and blue (RGB) channels as separate grayscale images. This approach is contrary to the conventional idea that the channels should not be treated separately [24]. However, putting the channels together results in more complicated models and much less efficient solution methods, making this type of image processing impossible in real time.

Numerical methods (computer-based methods) are used to solve PDE-based image processing models. A new method is needed since current methods have certain components that are out-of-date and inefficient, especially for high resolution images. As an alternative, this project will look at improved numerical methods that have been used for other equations. Exponential propagation iterative (EPI) methods [24] have shown to be efficient in solving nonlinear PDEs, which this project will work with. However, EPI methods are not scalable

and become slow at high resolution. Krylov subspace spectral (KSS) methods [21] are scalable, but they have mostly been used for linear PDEs. EPI and KSS methods will be further analyzed and adapted to enhance high resolution images.

It is hypothesized that combining EPI and KSS methods will produce a new numerical method that efficiently solves the image processing models developed in this project that treat the red, green, and blue channels of color images separately. This proposed approach to denoising and sharpening will be effective and scalable to higher resolution.

Chapter 3

Methodology

To design a new color image processing model, models created by Guidotti, Kim, and Lambers that modify the Perona-Malik equation will be further adapted. It is hypothesized that the red, green, and blue (RGB) channels of color images can be treated separately. To solve the new models, a numerical method will also be designed that represents the solution as a linear combination of sines and cosines and treats these different frequency components individually. It is hypothesized that this approach obtains the solution efficiently.

The starting point for a new model is the Perona-Malik equation [21] (introduced in Chapter 2)

$$\frac{\partial u}{\partial t} = \nabla \cdot \left(\frac{1}{1 + k^2 |\nabla u|^2} \nabla u \right), \tag{3.1}$$

which is a partial differential equation (PDE), where u is the input image as a function of (x, y) and t, where t is time. The spatial domain of u, which is composed of the (x, y) values, corresponds to the set of pixel locations, and the values of u are pixel values. All the pixel values make up the image. The equation contains the gradient of u, which is the rate of change of the image's colors with respect to the spatial directions, and k is a parameter for what is classified as small or large change. The solution of the PDE, a new function for u, is the output, or enhanced image. Here, u is a grayscale image.

The first existing modification that will be looked at is the equation developed by Guidotti and Lambers [14]

$$\frac{\partial u}{\partial t} = \nabla \cdot \left(\frac{1}{1 + k^2 |\nabla^{1-\varepsilon} u|^2} \nabla u \right), \tag{3.2}$$

where $0 < \varepsilon < \frac{1}{2}$ is a parameter. Perona-Malik is slightly changed to make u a periodic function by imposing periodic boundary conditions on the PDE. Periodic boundary conditions are used because taking derivatives of the basis functions (sines and cosines) still results in sines and cosines. Therefore, the new equation is easier to solve. However, using periodic boundary conditions has a drawback. Periodic functions wrap around, so what happens at one boundary of the image can affect the opposite boundary. To deal with this undesired effect, a buffer is added around the image by reflecting the content of the image across the boundary. As a result, only the content inside the buffer is affected by the periodic boundary conditions but not the rest of the image.

When $|\nabla u|$ is small, the model denoises because of forward diffusion, and when $|\nabla u|$ is large, the model sharpens because of backward diffusion. Thus, this equation is a forward-backward diffusion model like Perona-Malik. The application of $1-\varepsilon$ to the gradient, which consists of first partial derivatives, limits the gradient's growth by taking the partial derivatives of order $1-\varepsilon$. This lowers the order of the partial derivatives, and the lower the order of the partial derivative, the less amplification there is due to differentiation. Limiting the growth of the gradient, in turn, regularizes backward diffusion. It was discussed in Chapter 2 that regularization of Perona-Malik's backward diffusion is needed to combat its over-sharpening. With Perona-Malik, once the gradient is large enough, backward diffusion and, thus, sharpening occurs, which continues no matter how large the gradient gets.

In order to adapt (3.2) to color images, it is changed to

$$\frac{\partial u_i}{\partial t} = \nabla \cdot \left(\frac{1}{1+k^2|\nabla^{1-\varepsilon} u_i|^2} \nabla u_i \right), \tag{3.3}$$

where $\tilde{u}_{i,0}(x,y)$ is the noisy or blurry image inputted, which is option 1 for a new model and will be referred to as Model 1. Whereas u was a grayscale image in (3.1) and (3.2), (3.3) contains u_i, where $i = R,G,B$, to represent a function for each RGB channel of a color

image. Therefore, (3.3) needs to be solved three times, once for each channel. These three solutions will then be put together to get an enhanced color image.

The second equation that will be adapted, developed by Guidotti, Kim, and Lambers, is [15]

$$\frac{\partial u}{\partial t} = \nabla \cdot \left[\left(\frac{1}{1+k^2|\nabla u|^2} + \delta |\nabla u|^{p-2} \right) \nabla u \right], \tag{3.4}$$

where $\delta > 0$ and $1 \leq p < 2$ are parameters. This equation changes Perona-Malik by adding a new term to tone down the gradient and, thus, regularize Perona-Malik's backward diffusion. The extra term also ensures easier numerical treatment for the PDE. When $|\nabla u|$ is small, the model denoises; when $|\nabla u|$ is moderate, the model sharpens; and when $|\nabla u|$ is large, the model denoises to prevent discontinuities in the solution, which are harder to handle numerically. Unlike (3.1) and (3.2), this equation is a forward-backward-forward diffusion model. The model works best when $p \sim 1$. Moreover, if δ is chosen to be large, the model denoises. If δ is chosen to be small, the model sharpens [15]. Parameters are used in both modifications to determine how the model works – when to denoise and sharpen. Again, this equation is changed so that it can be applied to color images. The adapted equation is

$$\frac{\partial u_i}{\partial t} = \nabla \cdot \left[\left(\frac{1}{1+k^2|\nabla u_i|^2} + \delta |\nabla u_i|^{p-2} \right) \nabla u_i \right], \quad i = R, G, B, \tag{3.5}$$

which is option 2 for a new model and will be called Model 2.

When solving the PDE-based models in this project, numerical methods are needed because it is not possible to use analytical methods. Since the PDEs in this project are nonlinear, a numerical method that can work with the nonlinearity is called for. Nonlinear PDEs are more difficult to solve compared to linear PDEs, but solution methods for linear PDEs can be examined to determine solution methods for nonlinear PDEs. This is because nonlinear PDEs can be approximated by linear PDEs [3], which leads to methods for solving nonlinear PDEs with the same accuracy that is possible for linear PDEs.

The solutions of PDEs have to be represented in a way that can be computed numerically. For linear initial value problems, the general form for a linear, homogeneous PDE that is first-order in time can be written as

$$\frac{\partial u}{\partial t} = Lu, \tag{3.6}$$

where L is a differential operator. The initial condition is $u(x,y,0) = u_0(x,y)$, where $u_0(x,y)$ is the initial data. Therefore, $u(x,y,t) = e^{Lt}u_0(x,y)$ is the solution, but approximating e^{Lt} can be a challenge. It is a computationally expensive task when L, a differential operator, is acting on a function $u_0(x,y)$ that has high-frequency components. The definition of e^{Lt} can be considered as a Taylor series, $\sum_{j=0}^{\infty} \frac{(Lt)^j}{j!}$. Applying L^j to $u_0(x,y)$ leads to large terms in the Taylor expansion of the solution due to the amplification of high-frequency components by differentiation. These large terms cause the Taylor series to converge more slowly, increasing the computational expense. Therefore, it is impractical to evaluate e^{Lt} numerically.

Spatial discretization is used to change (3.6) to a system of ordinary differential equations (ODEs). They depend on one variable, making them easier to solve than PDEs [4]. Spatial discretization eliminates the spatial variables x, y so that the system of ODEs only depends on t. This is achieved by expressing the function u in (3.6) as a vector and expressing the differential operator L in (3.6) as a matrix A. Thus, (3.6) becomes

$$\vec{u}\,'(t) = A\vec{u}(t). \tag{3.7}$$

The initial condition is $\vec{u}(0) = \vec{u_0}$, which yields $\vec{u}(t) = e^{At}\vec{u}_0$ as the solution. If time-stepping is applied to compute the solution, then it gives $\vec{u}^{n+1}(t) \approx e^{A\Delta t}\vec{u}^n$, where $\Delta t = t_{n+1} - t_n$. Therefore, $t_n = n\Delta t$ for each n. Time-stepping starts solving a model at $t = 0$ and then takes small steps in time by Δt to get the solution [16]. For image processing, each time-step changes the image. Therefore, time-stepping is carried out until the image does not change

anymore because it has been enhanced as much as it can be. Though taking small time-steps improves accuracy, the trade-off is that it takes longer. Evaluating the solution this way is impractical because it would take too long to compute the exponential function $e^{A\Delta t}$ [18].

Another way that the solution can be represented is as a linear combination of sines and cosines, known as a Fourier series. This means that (3.7) can be represented as

$$\vec{u}(t) = \sum_{\omega_1,\omega_2=0}^{\infty} a_{\omega_1,\omega_2}(t)\cos(\omega_1\vec{x}+\omega_2\vec{y}) + b_{\omega_1,\omega_2}(t)\sin(\omega_1\vec{x}+\omega_2\vec{y}), \tag{3.8}$$

where a_{ω_1,ω_2} and b_{ω_1,ω_2} are unknown Fourier series coefficients. This form is particularly suitable for the solution of PDEs because it is easy to take the derivative of sine and cosine. This simplicity allows for more efficient numerical solution [16]. The goal is to solve for these coefficients by approximating $e^{A\Delta t}$ with a polynomial approximation. Existing time-stepping methods solve for the coefficients using the same polynomial to approximate $e^{A\Delta t}$ [18]. These numerical methods must use high-degree polynomials for high resolution functions, which takes more time.

An alternative option for time-stepping is Krylov subspace spectral (KSS) methods, which use different low-degree polynomials to solve for the coefficients, where the degree does not depend on resolution [21]. Therefore, KSS methods take less time, making them more efficient. In Chapter 4, it will be shown how to obtain the Fourier series coefficients for nonlinear diffusion image processing using a numerical method designed. In the new numerical method, KSS methods will be adapted to solve the nonlinear PDEs in this project, which will be done by combining KSS methods with exponential propagation iterative (EPI) methods [8]. Then, a computer program that will include this method will be written using MATLAB, a computer programming language. This program will solve the image processing models of this project by inputting a clean image, adding noise or blur to the image, and outputting an enhanced image.

Chapter 4

KSS Methods for Nonlinear Diffusion

A new numerical method is needed to efficiently solve our partial differential equation (PDE)-based image processing models. It was discussed in Chapter 3 that an appropriate way to represent the solution is as a linear combination of sines and cosines, or a Fourier series. While some numerical methods treat the different frequency components the same way, Krylov subspace spectral (KSS) methods treat them individually. This enables all the frequency components, low and high, to be computed as accurately as possible, which is necessary since they are needed to represent image details such as edges. In this chapter, KSS methods will be further analyzed and adapted to increase efficiency and to enhance high resolution images by solving for frequency-dependent Gaussian quadrature nodes. KSS methods will also be combined with exponential propagation iterative (EPI) methods to deal with the nonlinearity of our models.

To give some background on KSS methods (for linear PDEs, as they will be adapted to nonlinear PDEs later), we consider the parabolic PDE

$$\frac{\partial u}{\partial t} = \nabla \cdot (g(x,y)\nabla u) \tag{4.1}$$

on the domain $[0,2\pi]^2$ with periodic boundary conditions in both x and y. This is the form of the image processing models designed in Chapter 3, except for linearization. The initial condition is $u(x,y,0) = u_0(x,y)$, where $u_0(x,y)$ is the input data. The solution at time t_n is a Fourier series

$$u(x,y,t_n) = \frac{1}{2\pi} \sum_{\omega_1,\omega_2=-\infty}^{\infty} e^{i(\omega_1 x+\omega_2 y)} \hat{u}(\omega_1,\omega_2,t_n), \tag{4.2}$$

where the basis function $\{e^{i(\omega_1 x+\omega_2 y)}, \omega \in \mathbb{Z}^2\}$ is chosen because of the periodic boundary conditions of (4.1) and $\hat{u}(\omega_1,\omega_2,t_n)$ represents unknown Fourier coefficients. We let $Lu = (g(x,y)\nabla u)$ for the purpose of shorter notation and compute the Fourier coefficients of the solution $u(x,y,t_{n+1})$ as follows:

$$\hat{u}(\omega_1,\omega_2,t_{n+1}) = \left\langle \frac{1}{2\pi} e^{i(\omega_1 x+\omega_2 y)}, e^{L\Delta t} u(x,y,t_n) \right\rangle, \tag{4.3}$$

where $\left\langle \frac{1}{2\pi} e^{i(\omega_1 x+\omega_2 y)}, e^{L\Delta t} u(x,y,t_n) \right\rangle$ is the standard inner product on $[0,2\pi]^2$ and $e^{L\Delta t}$ is the solution operator of the PDE.

Spatial discretization is applied to (4.3) on an $N \times M$ grid to get the bilinear form

$$\vec{u}^H \varphi(A)\vec{v}, \tag{4.4}$$

where $\vec{u} = \frac{1}{2\pi} e^{i(\omega_1 x+\omega_2 y)}$ and $\vec{v} = u(x,y,t_n)$ are MN-vectors; $A = L_{MN}$, where L_{MN} is a spectral discretization of L; and $\varphi(\lambda) = e^{\lambda\Delta t}$.

Using the eigenvalues and eigenvectors of the matrix A [11], we can represent (4.3) as a Reimann-Stieltjes integral

$$\vec{u}^H \varphi(A)\vec{v} = \int_a^b \varphi(\lambda)\, d\alpha(\lambda), \tag{4.5}$$

where $\alpha(\lambda)$ is defined as in [8]. As a result, Gaussian quadrature rules can be used to obtain an approximation. The Lanczos algorithm is applied to A, a discretization of a differential operator, with initial vectors $\vec{u}$ and $\vec{v}$ to get the nodes and weights [11].

Computing (4.5) efficiently is difficult since $\varphi(A)$ is exponential. We can approximate (4.5) as

$$\int_a^b \varphi(\lambda)\, d\alpha(\lambda) \approx \int_a^b p(\lambda)\, d\alpha(\lambda), \tag{4.6}$$

where $p(\lambda)$ is an interpolating polynomial. We can then rewrite (4.6) as

$$\int_a^b p(\lambda)\, d\alpha(\lambda) = \vec{u}^H p(A)\vec{v}, \tag{4.7}$$

which is easier to compute since $p(A)$ is a polynomial function. The next step is to obtain interpolation points, which are the same as Gaussian quadrature nodes, for (4.6) for maximum accuracy, efficiency, and scalability.

When $\vec{u} = \vec{v}$, the weights are guaranteed to be positive, so we can use the Lanczos algorithm, which will be explained later, to get nodes. The inputs are A and $\vec{u}$. The output is a symmetric-tridiagonal matrix $\mathcal{T}$, and the nodes are eigenvalues of $\mathcal{T}$ [13]. If $\vec{u} \neq \vec{v}$, there could be a negative weight. Therefore, we choose a different approach since the quadrature rule would not apply for a negative weight [1]. Alternatively, we can find the approximation of the 2×2 matrix integral

$$\begin{bmatrix} \vec{u} & \vec{v} \end{bmatrix}^H \varphi(A) \begin{bmatrix} \vec{u} & \vec{v} \end{bmatrix} \tag{4.8}$$

by using the block Lanczos algorithm [12]. The inputs are A and $\begin{bmatrix} \vec{u} & \vec{v} \end{bmatrix}$, and the output is a symmetric block-tridiagonal matrix $\mathcal{T}$, where the nodes are the eigenvalues.

In the block KSS method, we define

$$R_0(\vec{\omega}) = \begin{bmatrix} \hat{e}_{\vec{\omega}} & \vec{u}^n \end{bmatrix} \tag{4.9}$$

with $\vec{\omega}(\omega_1, \omega_1)$, where $\hat{e}_{\vec{\omega}}$ is a discretization of $\frac{1}{2\pi} e^{i(\omega_1 x + \omega_2 y)}$ and $\vec{u}^n$ is the computed solution at time t_n. We get $R_0(\vec{\omega}) = X_1(\vec{\omega}) B_0(\vec{\omega})$ from the $Q\,R$ factorization of (4.9), which is needed to make the blocks orthogonal. Block Lanczos iteration is applied to the discretized operator A with $X_1(\vec{\omega})$ as the initial block. This gives a symmetric block-tridiagonal matrix

$$\mathcal{T}_K(\vec{\omega}) = \begin{bmatrix} M_1 & B_1^H & & \\ B_1 & M_2 & B_2^H & \\ & \ddots & \ddots & \ddots \\ & & B_{K-1} & M_K \end{bmatrix}, \tag{4.10}$$

where each entry is a 2×2 matrix that is a function of $\vec{\omega}$. We can then express each Fourier coefficient of the solution at time t_{n+1} as

$$\left[\hat{u}^{n+1}\right]_{\vec{\omega}} = \left[B_0^H E_{12}^H e^{-\mathcal{T}_K(\vec{\omega})\Delta t} E_{12} B_0 \right]_{12}, \quad E_{12} = \begin{bmatrix} \vec{e}_1 & \vec{e}_2 \end{bmatrix}. \tag{4.11}$$

The block KSS method is $O(\Delta t^{2K-1})$-order accurate [17].

Now we discuss block Lanczos iteration in more depth. We consider the block (4.9), where $\omega_1 = -N/2+1,...,N/2$ and $\omega_2 = -M/2+1,...,M/2$. Block Lanczos iteration is used to compute the symmetric block-tridiagonal matrix

$$\mathcal{T}_3(\vec{\omega}) = \begin{bmatrix} M_1 & B_1^H & 0 \\ B_1 & M_2 & B_2^H \\ 0 & B_2 & M_3 \end{bmatrix}, \tag{4.12}$$

where $K = 3$ as an example. We can the show the structure of (4.12) as the matrix

$$\mathcal{T}_3(\vec{\omega}) = \begin{bmatrix} \times & \times & \times & 0 & 0 & 0 \\ \times & \times & \times & \times & 0 & 0 \\ \times & \times & \times & \times & \times & 0 \\ 0 & \times & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times & \times \\ 0 & 0 & 0 & \times & \times & \times \end{bmatrix}, \tag{4.13}$$

where the $\times$'s indicate nonzero entries of matrices M_j and B_j. As $||\vec{\omega}|| \to \infty$, the matrix becomes

$$\mathcal{T}_3(\vec{\omega}) \approx \begin{bmatrix} \times & 0 & \times & 0 & 0 & 0 \\ 0 & \times & 0 & \times & 0 & 0 \\ \times & 0 & \times & 0 & \times & 0 \\ 0 & \times & 0 & \times & 0 & \times \\ 0 & 0 & \times & 0 & \times & 0 \\ 0 & 0 & 0 & \times & 0 & \times \end{bmatrix}, \tag{4.14}$$

where the $\times$'s are insensitive to frequency and the $\times$'s are heavily dependent on frequency as can be seen in [21]. It is observed that some of the $\times$'s from (4.13) become zero. This is because each entry is a Fourier coefficient of some function, and Fourier coefficients go to 0 as $||\vec{\omega}|| \to \infty$. Thus, each entry $(\mathcal{T}_3(\vec{\omega}))_{ij}$, where $i+j$ is odd, gets negligibly small.

If both the rows and columns are reordered so that odd numbered and even numbered

rows and columns are grouped together, we get the matrix

$$\mathcal{T}_3(\vec{\omega}) \approx \begin{bmatrix} \times & \times & 0 & 0 & 0 & 0 \\ \times & \times & \times & 0 & 0 & 0 \\ 0 & \times & \times & 0 & 0 & 0 \\ & & & & & \\ 0 & 0 & 0 & \times & \times & 0 \\ 0 & 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & 0 & \times & \times \end{bmatrix}. \tag{4.15}$$

We can see that all the $\times$'s are together in the upper left 3×3 block and all the $\times$'s are together in the lower right 3×3 block. This shows that the eigenvalue problem decouples. Essentially, one block Lanczos iteration becomes two "non-block" Lanczos, or simply Lanczos, iterations. The $\times$'s are frequency-independent nodes, which we can find by applying Lanczos iteration to the matrix A with initial vector $\vec{u}^n$. Since $\vec{u}^n$ does not depend on $\vec{\omega}$, the frequency-independent nodes only need to be computed once and are then shared by all frequencies, which saves time. The $\times$'s are frequency-dependent nodes. By applying Lanczos iteration analytically to the differential operator L with initial vector $\hat{e}_{\vec{\omega}}$, we obtain formulas for the nodes in terms of the coefficients of L.

The Lanczos algorithm is given as follows:

$r_0 = u, x_0 = 0$

for $k = 1, 2, ..., K$

$\beta_{k-1} = ||r_{k-1}||$

$x_k = r_{k-1}/\beta_{k-1}$

$v_k = Lx_k$

$\alpha_k = \langle x_k, v_k \rangle$

$r_k = v_k - \alpha_k x_k - \beta_{k-1} x_{k-1}$

end

The output is

$$\mathcal{T}_k(\vec{\omega}) = \begin{bmatrix} \alpha_1 & \beta_1 & & & \\ \beta 1 & \alpha_2 & \beta_2 & & \\ & \beta_2 & \ddots & \ddots & \\ & & \ddots & \ddots & \beta_{K-1} \\ & & & \beta_{K-1} & \alpha_K \end{bmatrix}. \tag{4.16}$$

We can apply the Lanczos algorithm to a case in which the initial vector is $u = e^{i(\omega_1 x+\omega_2 y)}$, $\vec{\omega} \in \mathbb{Z}^2$ and the differential operator is $Lu = \nabla \cdot (g\nabla u)$, where $g = \frac{1}{1+k^2||\nabla u||^2}$ is a function of (x,y). Two iterations will be carried out to get formulas for the frequency-dependent nodes α_1, β_1, and α_2.

Substituting $u = e^{i(\omega_1 x+\omega_2 y)}$ we have

$$\begin{aligned} r_0 &= u \\ &= e^{i(\omega_1 x+\omega_2 y)}, \end{aligned} \tag{4.17}$$

and

$$x_0 = 0 \tag{4.18}$$

is given.

Now we start the first iteration with $k = 1$. To obtain β_0, we substitute r_0 with $e^{i(\omega_1 x+\omega_2 y)}$ and evaluate its norm:

$$\begin{aligned} \beta_0 &= ||r_0|| \\ &= \left|\left| e^{i(\omega_1 x+\omega_2 y)} \right|\right| \\ &= \sqrt{\left\langle e^{i(\omega_1 x+\omega_2 y)}, e^{i(\omega_1 x+\omega_2 y)} \right\rangle} \\ &= \sqrt{\int_0^{2\pi}\int_0^{2\pi} \overline{e^{i(\omega_1 x+\omega_2 y)}} e^{i(\omega_1 x+\omega_2 y)} dx\,dy} \\ &= \sqrt{\int_0^{2\pi}\int_0^{2\pi} e^{-i(\omega_1 x+\omega_2 y)} e^{i(\omega_1 x+\omega_2 y)} dx\,dy} \\ &= \sqrt{2\pi^2} \\ &= 2\pi. \end{aligned} \tag{4.19}$$

Substitution of r_0 and β_0 gives us x_1:

$$\begin{aligned} x_1 &= r_0/\beta_0 \\ &= \frac{1}{2\pi}e^{i(\omega_1 x+\omega_2 y)}. \end{aligned} \tag{4.20}$$

v_1 is found by applying the differential operator L to $\frac{1}{2\pi}e^{i(\omega_1 x+\omega_2 y)}$ after substituting it for x_1:

$$\begin{aligned} v_1 &= Lx_1 \\ &= L\left(\frac{1}{2\pi}e^{i(\omega_1 x+\omega_2 y)}\right) \\ &= \nabla\cdot\left(g\nabla\frac{1}{2\pi}e^{i(\omega_1 x+\omega_2 y)}\right) \\ &= \frac{\partial}{\partial x}\left[g\frac{\partial}{\partial x}\left(\frac{1}{2\pi}e^{i(\omega_1 x+\omega_2 y)}\right)\right]+\frac{\partial}{\partial y}\left[g\frac{\partial}{\partial y}\left(\frac{1}{2\pi}e^{i(\omega_1 x+\omega_2 y)}\right)\right] \\ &= \frac{\partial}{\partial x}\left(g\frac{1}{2\pi}e^{i(\omega_1 x+\omega_2 y)}i\omega_1\right)+\frac{\partial}{\partial y}\left(g\frac{1}{2\pi}e^{i(\omega_1 x+\omega_2 y)}i\omega_2\right) \\ &= g\left(\frac{1}{2\pi}e^{i(\omega_1 x+\omega_2 y)}(i\omega_1)^2\right)+g_x\left(\frac{1}{2\pi}e^{i(\omega_1 x+\omega_2 y)}i\omega_1\right) \\ &\quad +g\left(\frac{1}{2\pi}e^{i(\omega_1 x+\omega_2 y)}(i\omega_2)^2\right)+g_y\left(\frac{1}{2\pi}e^{i(\omega_1 x+\omega_2 y)}i\omega_2\right). \end{aligned} \tag{4.21}$$

After grouping, we get

$$\begin{aligned} v_1 &= \frac{1}{2\pi}e^{i(\omega_1 x+\omega_2 y)}\left[g(i\omega_1)^2+g_x i\omega_1+g(i\omega_2)^2+g_y i\omega_2\right] \\ &= \frac{1}{2\pi}e^{i(\omega_1 x+\omega_2 y)}\left[-g\left(\omega_1^2+\omega_2^2\right)+i(\omega_1 g_x+\omega_2 g_y)\right]. \end{aligned}$$

α_1 is the inner product of x_1 and v_1. We plug in $x_1 = \frac{1}{2\pi}e^{i(\omega_1 x+\omega_2 y)}$ and $v_1 = \frac{1}{2\pi}e^{i(\omega_1 x+\omega_2 y)}\left[-g\left(\omega_1^2+\omega_2^2\right)+i(\omega_1 g_x+\omega_2 g_y)\right]$ and evaluate

$$\begin{aligned} \alpha_1 &= \langle x_1, v_1\rangle \\ &= \left\langle \frac{1}{2\pi}e^{i(\omega_1 x+\omega_2 y)}, \frac{1}{2\pi}e^{i(\omega_1 x+\omega_2 y)}\left[-g\left(\omega_1^2+\omega_2^2\right)+i(\omega_1 g_x+\omega_2 g_y)\right]\right\rangle \\ &= \int_0^{2\pi}\int_0^{2\pi}\overline{\frac{1}{2\pi}e^{i(\omega_1 x+\omega_2 y)}}\frac{1}{2\pi}e^{i(\omega_1 x+\omega_2 y)}\left[-g\left(\omega_1^2+\omega_2^2\right)+i(\omega_1 g_x+\omega_2 g_y)\right]dxdy \end{aligned} \tag{4.22}$$

$$\begin{aligned}
\alpha_1 &= \int_0^{2\pi}\int_0^{2\pi} \frac{1}{2\pi} e^{-i(\omega_1 x+\omega_2 y)} \frac{1}{2\pi} e^{i(\omega_1 x+\omega_2 y)} \left[-g\left(\omega_1^2+\omega_2^2\right)+i(\omega_1 g_x+\omega_2 g_y)\right] dxdy \\
&= \int_0^{2\pi}\int_0^{2\pi} \left(\frac{1}{2\pi}\right)^2 \left[-g\left(\omega_1^2+\omega_2^2\right)+i(\omega_1 g_x+\omega_2 g_y)\right] dxdy \\
&= -\left(\omega_1^2+\omega_2^2\right)\left(\frac{1}{2\pi}\right)^2 \int_0^{2\pi}\int_0^{2\pi} g\,dxdy + \left(\frac{1}{2\pi}\right)^2 i\omega_1 \int_0^{2\pi}\int_0^{2\pi} g_x\,dxdy \\
&\quad + \left(\frac{1}{2\pi}\right)^2 i\omega_2 \int_0^{2\pi}\int_0^{2\pi} g_y\,dxdy.
\end{aligned}$$

Let us denote the average of a function $f(x,y)$ on the interval $[a,b]$ by

$$\text{avg}_{x,y\in[0,2\pi]^2}(f(x,y)) = \frac{1}{(b-a)^2}\int_a^b\int_a^b f(x,y)\,dxdy = \overline{f(x,y)}.$$

We use this and the evaluation that $\left(\frac{1}{2\pi}\right)^2 i\omega_1 \int_0^{2\pi}\int_0^{2\pi} g_x\,dxdy = 0$

and $\left(\frac{1}{2\pi}\right)^2 i\omega_2 \int_0^{2\pi}\int_0^{2\pi} g_y\,dxdy = 0$ because of 2π periodicity to get

$$\begin{aligned}
\alpha_1 &= -\left(\omega_1^2+\omega_2^2\right)\left(\frac{1}{2\pi}\right)^2 2\pi\overline{g} \\
&= -\overline{g}\left(\omega_1^2+\omega_2^2\right).
\end{aligned}$$

Substitution for v_1, α_1, x_1, β_0, and x_0 gives us the following for r_1:

$$\begin{aligned}
r_1 &= v_1 - \alpha_1 x_1 - \beta_0 x_0 \qquad (4.23)\\
&= \frac{1}{2\pi} e^{i(\omega_1 x+\omega_2 y)} \left[-g\left(\omega_1^2+\omega_2^2\right)+i(\omega_1 g_x+\omega_2 g_y)\right] - \left[-\overline{g}\left(\omega_1^2+\omega_2^2\right)\right] \\
&\quad \times \left[\frac{1}{2\pi} e^{i(\omega_1 x+\omega_2 y)}\right] - 2\pi(0).
\end{aligned}$$

$\frac{1}{2\pi} e^{i(\omega_1 x+\omega_2 y)}$ is then factored out:

$$r_1 = \frac{1}{2\pi} e^{i(\omega_1 x+\omega_2 y)} \left[-g\left(\omega_1^2+\omega_2^2\right)+i(\omega_1 g_x+\omega_2 g_y)+\overline{g}\left(\omega_1^2+\omega_2^2\right)\right].$$

Using the notation

$$f-\overline{f} = \tilde{f},$$

we obtain

$$r_1 = \frac{1}{2\pi} e^{i(\omega_1 x+\omega_2 y)} \left[-\tilde{g}\left(\omega_1^2+\omega_2^2\right)+i(\omega_1 g_x+\omega_2 g_y)\right].$$

We continue with the second iteration with $k = 2$. Using the expression we just found for r_1, we evaluate β_1:

$$\begin{aligned} \beta_1 &= ||r_1|| \\ &= \left|\left|\frac{1}{2\pi}e^{i(\omega_1 x+\omega_2 y)}\left[-\tilde{g}\left(\omega_1^2+\omega_2^2\right)+i(\omega_1 g_x+\omega_2 g_y)\right]\right|\right|. \end{aligned} \tag{4.24}$$

Squaring both sides allows us to evaluate β_1 more conveniently:

$$\beta_1^2 = \left|\left|\frac{1}{2\pi}e^{i(\omega_1 x+\omega_2 y)}\left[-\tilde{g}\left(\omega_1^2+\omega_2^2\right)+i(\omega_1 g_x+\omega_2 g_y)\right]\right|\right|^2. \tag{4.25}$$

Since we know that the $e^{i(\omega_1 x+\omega_2 y)}$ terms will cancel out, this simplifies

$$\begin{aligned} \beta_1^2 &= \left|\left|\frac{1}{2\pi}\left[-\tilde{g}\left(\omega_1^2+\omega_2^2\right)+i(\omega_1 g_x+\omega_2 g_y)\right]\right|\right|^2 \\ &= \left|\left|\frac{1}{2\pi}\left[-\tilde{g}\left(\omega_1^2+\omega_2^2\right)\right]\right|\right|^2+\left|\left|\frac{1}{2\pi}i(\omega_1 g_x+\omega_2 g_y)\right|\right|^2. \end{aligned}$$

The i terms will cancel out, so we get

$$\begin{aligned} \beta_1^2 &= \left|\left|\frac{1}{2\pi}\left[-\tilde{g}\left(\omega_1^2+\omega_2^2\right)\right]\right|\right|^2+\left|\left|\frac{1}{2\pi}(\omega_1 g_x+\omega_2 g_y)\right|\right|^2 \\ &= \left(\frac{1}{2\pi}\right)^2\left(\omega_1^2+\omega_2^2\right)^2||-\tilde{g}||^2+\left(\frac{1}{2\pi}\right)^2||\omega_1 g_x+\omega_2 g_y||^2. \end{aligned}$$

We use $||-\tilde{g}||^2 = ||\tilde{g}||^2$ and rewrite $\omega_1 g_x+\omega_2 g_y$ as $\vec{\omega}\cdot\nabla g$, where $\vec{\omega} = (\omega_1,\omega_2)$, so that

$$\beta_1^2 = \left(\frac{1}{2\pi}\right)^2\left(\omega_1^2+\omega_2^2\right)^2||\tilde{g}||^2+\left(\frac{1}{2\pi}\right)^2||\vec{\omega}\cdot\nabla g||^2.$$

Therefore,

$$\beta_1 = \sqrt{\left(\frac{1}{2\pi}\right)^2\left(\omega_1^2+\omega_2^2\right)^2||\tilde{g}||^2+\left(\frac{1}{2\pi}\right)^2||\vec{\omega}\cdot\nabla g||^2}.$$

We can use substitution to find expressions for x_2 and v_2:

$$\begin{aligned} x_2 &= r_1/\beta_1 \\ &= \frac{1}{2\pi}e^{i(\omega_1 x+\omega_2 y)}\left[-\tilde{g}\left(\omega_1^2+\omega_2^2\right)+i(\omega_1 g_x+\omega_2 g_y)\right]\Big/ \\ &\quad \sqrt{\left(\frac{1}{2\pi}\right)^2\left(\omega_1^2+\omega_2^2\right)^2||\tilde{g}||^2+\left(\frac{1}{2\pi}\right)^2||\vec{\omega}\cdot\nabla g)||^2}, \end{aligned} \tag{4.26}$$

and

$$\begin{aligned} v_2 &= Lx_2 \\ &= L\Bigg(\frac{1}{2\pi} e^{i(\omega_1 x+\omega_2 y)} \left[-\tilde{g}\left(\omega_1^2+\omega_2^2\right) + i(\omega_1 g_x + \omega_2 g_y)\right] \Big/ \\ &\quad \sqrt{\left(\frac{1}{2\pi}\right)^2 \left(\omega_1^2+\omega_2^2\right)^2 ||\tilde{g}||^2 + \left(\frac{1}{2\pi}\right)^2 ||\vec{\omega}\cdot\nabla g)||^2]} \Bigg). \end{aligned} \tag{4.27}$$

After substitution, we obtain an inner product we can evaluate conveniently for α_2:

$$\begin{aligned} \alpha_2 &= \langle x_2, v_2 \rangle \\ &= \langle x_2, Lx_2 \rangle \\ &= \left\langle \frac{r_1}{\beta_1}, L\frac{r_1}{\beta_1} \right\rangle \\ &= \left\langle \frac{r_1, Lr_1}{\beta_1^2} \right\rangle. \end{aligned} \tag{4.28}$$

We can pull out $\frac{1}{\beta_1^2}$ and apply L to r_1:

$$\begin{aligned} \alpha_2 &= \frac{1}{\beta_1^2}\langle r_1, Lr_1 \rangle \\ &= \frac{1}{\beta_1^2}\langle r_1, \nabla\cdot(g\nabla r_1)\rangle \\ &= \frac{1}{\beta_1^2}\langle r_1, (g(r_1)_x)_x + (g(r_1)_y)_y\rangle. \end{aligned}$$

After integration by parts, this becomes

$$\begin{aligned} \alpha_2 &= \frac{1}{\beta_1^2}[\langle r_1, (g(r_1)_x)_x\rangle + \langle r_1, (g(r_1)_y)_y\rangle] \\ &= \frac{1}{\beta_1^2}[-\langle (r_1)_x, g(r_1)_x\rangle - \langle (r_1)_y, g(r_1)_y\rangle]. \end{aligned}$$

We use the substitution of r_1, neglecting $i(\omega_1 g_x + \omega_2 g_y)$ because of its lower order in $\vec{\omega}$, so

that we have

$$
\begin{aligned}
\alpha_2 \approx\ & \frac{1}{\beta_1^2}\Bigg[-\Bigg\langle\left(\frac{1}{2\pi}e^{i(\omega_1 x+\omega_2 y)}\left(-\tilde{g}\left(\omega_1^2+\omega_2^2\right)\right)\right)_x, \\
& g\left(\frac{1}{2\pi}e^{i(\omega_1 x+\omega_2 y)}\left(-\tilde{g}\left(\omega_1^2+\omega_2^2\right)\right)\right)_x\Bigg\rangle-\Bigg\langle\left(\frac{1}{2\pi}e^{i(\omega_1 x+\omega_2 y)}\left(-\tilde{g}\left(\omega_1^2+\omega_2^2\right)\right)\right)_y, \\
& g\left(\frac{1}{2\pi}e^{i(\omega_1 x+\omega_2 y)}\left(-\tilde{g}\left(\omega_1^2+\omega_2^2\right)\right)\right)_y\Bigg\rangle\Bigg] \\
\approx\ & \frac{1}{\beta_1^2}\Bigg[-\left(\frac{1}{2\pi}\left(\omega_1^2+\omega_2^2\right)\right)^2\left\langle\left(-\tilde{g}e^{i(\omega_1 x+\omega_2 y)}\right)_x, g\left(-\tilde{g}e^{i(\omega_1 x+\omega_2 y)}\right)_x\right\rangle \\
& -\left(\frac{1}{2\pi}\left(\omega_1^2+\omega_2^2\right)\right)^2\left\langle\left(-\tilde{g}e^{i(\omega_1 x+\omega_2 y)}\right)_y, g\left(-\tilde{g}e^{i(\omega_1 x+\omega_2 y)}\right)_y\right\rangle\Bigg] \\
\approx\ & \frac{1}{\beta_1^2}\Bigg[-\left(\frac{1}{2\pi}\left(\omega_1^2+\omega_2^2\right)\right)^2\Bigg[\left\langle\left(\tilde{g}e^{i(\omega_1 x+\omega_2 y)}\right)_x, g\left(\tilde{g}e^{i(\omega_1 x+\omega_2 y)}\right)_x\right\rangle \\
& +\left\langle\left(\tilde{g}e^{i(\omega_1 x+\omega_2 y)}\right)_y, g\left(\tilde{g}e^{i(\omega_1 x+\omega_2 y)}\right)_y\right\rangle\Bigg]\Bigg] \\
\approx\ & \frac{1}{\beta_1^2}\Bigg[-\left(\frac{1}{2\pi}\left(\omega_1^2+\omega_2^2\right)\right)^2\Bigg[\left\langle\tilde{g}e^{i(\omega_1 x+\omega_2 y)}i\omega_1+\tilde{g}_x e^{i(\omega_1 x+\omega_2 y)}, g\left(\tilde{g}e^{i(\omega_1 x+\omega_2 y)}i\omega_1\right.\right. \\
& \left.\left.+\tilde{g}_x e^{i(\omega_1 x+\omega_2 y)}\right)\right\rangle+\left\langle\tilde{g}e^{i(\omega_1 x+\omega_2 y)}i\omega_2+\tilde{g}_y e^{i(\omega_1 x+\omega_2 y)}, g\left(\tilde{g}e^{i(\omega_1 x+\omega_2 y)}i\omega_2\right.\right. \\
& \left.\left.+\tilde{g}_y e^{i(\omega_1 x+\omega_2 y)}\right)\right\rangle\Bigg]\Bigg] \\
\approx\ & \frac{1}{\beta_1^2}\Bigg[-\left(\frac{1}{2\pi}\left(\omega_1^2+\omega_2^2\right)\right)^2\Bigg[\left\langle\tilde{g}e^{i(\omega_1 x+\omega_2 y)}i\omega_1+\tilde{g}_x e^{i(\omega_1 x+\omega_2 y)}, g\tilde{g}e^{i(\omega_1 x+\omega_2 y)}i\omega_1\right. \\
& \left.+g\tilde{g}_x e^{i(\omega_1 x+\omega_2 y)}\right\rangle+\left\langle\tilde{g}e^{i(\omega_1 x+\omega_2 y)}i\omega_2+\tilde{g}_y e^{i(\omega_1 x+\omega_2 y)}, g\tilde{g}e^{i(\omega_1 x+\omega_2 y)}i\omega_2\right. \\
& \left.+g\tilde{g}_y e^{i(\omega_1 x+\omega_2 y)}\right\rangle\Bigg]\Bigg].
\end{aligned}
$$

Now we evaluate the inner product:

$$
\begin{aligned}
\alpha_2 &\approx \frac{1}{\beta_1^2}\left[-\left(\frac{1}{2\pi}\left(\omega_1^2+\omega_2^2\right)\right)^2\left(\int_0^{2\pi}\int_0^{2\pi}\overline{\left(\tilde{g}e^{i(\omega_1 x+\omega_2 y)}i\omega_1+\tilde{g}_x e^{i(\omega_1 x+\omega_2 y)}\right)}\right.\right.\\
&\quad\times\left(g\tilde{g}e^{i(\omega_1 x+\omega_2 y)}i\omega_1+g\tilde{g}_x e^{i(\omega_1 x+\omega_2 y)}\right)dxdy\\
&\quad+\int_0^{2\pi}\int_0^{2\pi}\overline{\left(\tilde{g}e^{i(\omega_1 x+\omega_2 y)}i\omega_2+\tilde{g}_y e^{i(\omega_1 x+\omega_2 y)}\right)}\\
&\quad\left.\left.\times\left(g\tilde{g}e^{i(\omega_1 x+\omega_2 y)}i\omega_2+g\tilde{g}_y e^{i(\omega_1 x+\omega_2 y)}\right)dxdy\right)\right]\\
&\approx \frac{1}{\beta_1^2}\left[-\left(\frac{1}{2\pi}\left(\omega_1^2+\omega_2^2\right)\right)^2\left(\int_0^{2\pi}\int_0^{2\pi}\left(\tilde{g}e^{-i(\omega_1 x+\omega_2 y)}(-i\omega_1)+\tilde{g}_x e^{-i(\omega_1 x+\omega_2 y)}\right)\right.\right.\\
&\quad\times\left(g\tilde{g}e^{i(\omega_1 x+\omega_2 y)}(i\omega_1)+g\tilde{g}_x e^{i(\omega_1 x+\omega_2 y)}\right)dxdy\\
&\quad+\int_0^{2\pi}\int_0^{2\pi}\left(\tilde{g}e^{-i(\omega_1 x+\omega_2 y)}(-i\omega_2)+\tilde{g}_y e^{-i(\omega_1 x+\omega_2 y)}\right)\\
&\quad\left.\left.\times\left(g\tilde{g}e^{i(\omega_1 x+\omega_2 y)}(i\omega_2)+g\tilde{g}_y e^{i(\omega_1 x+\omega_2 y)}\right)dxdy\right)\right]\\
&\approx \frac{1}{\beta_1^2}\left[-\left(\frac{1}{2\pi}\left(\omega_1^2+\omega_2^2\right)\right)^2\left(\int_0^{2\pi}\int_0^{2\pi}g\tilde{g}^2+g\tilde{g}\tilde{g}_x(-i\omega_1)+g\tilde{g}\tilde{g}_x i\omega_1+g\tilde{g}_x^2\,dxdy\right.\right.\\
&\quad\left.\left.+\int_0^{2\pi}\int_0^{2\pi}g\tilde{g}^2+g\tilde{g}\tilde{g}_y(-i\omega_2)+g\tilde{g}\tilde{g}_y i\omega_2+g\tilde{g}_y^2\,dxdy\right)\right]\\
&\approx \frac{1}{\beta_1^2}\left[-\left(\frac{1}{2\pi}\left(\omega_1^2+\omega_2^2\right)\right)^2\left(\int_0^{2\pi}\int_0^{2\pi}g\tilde{g}^2+g\tilde{g}_x^2\,dxdy\right.\right.\\
&\quad\left.\left.+\int_0^{2\pi}\int_0^{2\pi}g\tilde{g}^2+g\tilde{g}_y^2\,dxdy\right)\right]\\
&\approx \frac{1}{\beta_1^2}\left[-\left(\frac{1}{2\pi}\left(\omega_1^2+\omega_2^2\right)\right)^2\left(2\pi\overline{g}_{\tilde{g}^2}\omega_1^2+2\pi\overline{g}_{\tilde{g}_x^2}+2\pi\overline{g}_{\tilde{g}^2}\omega_2^2+2\pi\overline{g}_{\tilde{g}_y^2}\right)\right]\\
&\approx \frac{1}{\beta_1^2}\left[-\left(\omega_1^2+\omega_2^2\right)^2\left(\overline{g}_{\tilde{g}^2}\omega_1^2+\overline{g}_{\tilde{g}_x^2}+\overline{g}_{\tilde{g}^2}\omega_2^2+\overline{g}_{\tilde{g}_y^2}\right)\right]\\
&\approx \frac{1}{\beta_1^2}\left[-\left(\omega_1^2+\omega_2^2\right)^3\overline{g}_{\tilde{g}^2}+\left(\omega_1^2+\omega_2^2\right)^2\left(\overline{g}_{\tilde{g}_x^2}+\overline{g}_{\tilde{g}_y^2}\right)\right]\\
&\approx \frac{1}{\beta_1^2}\left[-\left(\omega_1^2+\omega_2^2\right)^3\overline{g}_{\tilde{g}^2}+\left(\omega_1^2+\omega_2^2\right)^2\overline{g}_{||\nabla g||^2}\right].
\end{aligned}
$$

We can now divide by β_1^2, neglecting $\left(\omega_1^2+\omega_2^2\right)^2\overline{g}_{||\nabla g||^2}$ and $\left(\frac{1}{2\pi}\right)^2||\vec{\omega}\cdot\nabla g||^2$:

$$\begin{aligned}\alpha_2 &\approx \frac{-\left(\omega_1^2+\omega_2^2\right)^3\overline{g}_{\tilde{g}^2}+\left(\omega_1^2+\omega_2^2\right)^2\overline{g}_{||\nabla g||^2}}{\left(\frac{1}{2\pi}\right)^2\left(\omega_1^2+\omega_2^2\right)^2||\tilde{g}||^2+\left(\frac{1}{2\pi}\right)^2||\vec{\omega}\cdot\nabla g||^2}\\ &\approx \frac{-\left(\omega_1^2+\omega_2^2\right)^3\overline{g}_{\tilde{g}^2}}{\left(\frac{1}{2\pi}\right)^2\left(\omega_1^2+\omega_2^2\right)^2||\tilde{g}||^2}.\end{aligned}$$

Using inner products, this can be rewritten as

$$\alpha_2 \approx \frac{-\left(\omega_1^2+\omega_2^2\right)^3(\frac{1}{2\pi})^2\langle\tilde{g},g\tilde{g}\rangle}{\left(\frac{1}{2\pi}\right)^2\left(\omega_1^2+\omega_2^2\right)^2\langle\tilde{g},\tilde{g}\rangle}.$$

Let us define the average of a function with respect to some weight function as

$$\overline{f}_h=\int_0^{2\pi}\int_0^{2\pi} f\left(\frac{h}{||h||}\right)^2=\frac{\langle h,fh\rangle}{\langle h,h\rangle}.$$

We use this to give us the final formula for α_2:

$$\begin{aligned}\alpha_2 &\approx -\left(\omega_1^2+\omega_2^2\right)\overline{g}_{\left(\frac{\tilde{g}}{||\tilde{g}||}\right)^2}\\ &\approx -\left(\omega_1^2+\omega_2^2\right)\overline{g}_{\tilde{g}}.\end{aligned}$$

After obtaining the formulas for the nodes, the output is

$$\mathcal{T}_2(\vec{\omega})\approx\begin{bmatrix}\overline{g}||\vec{\omega}||^2 & \left(\frac{1}{2\pi}\right)^2\sqrt{||\vec{\omega}||^4||\tilde{g}||^2+||\vec{\omega}\cdot\nabla g||^2}\\ \left(\frac{1}{2\pi}\right)^2\sqrt{||\vec{\omega}||^4||\tilde{g}||^2+||\vec{\omega}\cdot\nabla g||^2} & -||\vec{\omega}||^2\overline{g}_{\tilde{g}}\end{bmatrix}, \tag{4.29}$$

where $\omega_1^2+\omega_2^2=||\vec{\omega}||^2$. We can get the eigenvalues of this matrix by solving for the roots of

$$\det(\mathcal{T}_2(\vec{\omega})-\lambda I). \tag{4.30}$$

This results in

$$\lambda_{1,2}\approx-\frac{\mathrm{tr}(\mathcal{T}_2(\vec{\omega}))\pm\sqrt{\mathrm{tr}(\mathcal{T}_2(\vec{\omega}))^2-4\det(\mathcal{T}_2(\vec{\omega}))}}{2}. \tag{4.31}$$

This is the method we will use to obtain the frequency-dependent nodes that are used to approximate each Fourier coefficient of the solutions of our PDE-based image processing models.

Since KSS methods have been mostly used for linear PDEs, they need to be adapted to nonlinear PDEs so that we can use them for our image processing models. This will be done by combining KSS methods with exponential propagation iterative (EPI) methods, which are efficient in solving nonlinear PDEs. We will use a 3rd-order, 2-stage EPI method [24]

$$\begin{aligned} Y_1 &= \vec{y}_n + \frac{1}{3} h a_{11} \varphi_1\left(\frac{1}{3}hA\right) F(\vec{y}_n), \\ \vec{y}_{n+1} &= \vec{y}_n + h\varphi_1(hA)F(\vec{y}_n) + 3hb_1\varphi_2(hA)[F(Y_1) - F(\vec{y}_n) - A(Y_1 - \vec{y}_n)], \end{aligned} \tag{4.32}$$

where $a_{11} = 9/4$ and $b_1 = 32/81$, and

$$R(Y_1) = F(Y_1) - F(\vec{y}_n) - A(Y_1 - \vec{y}_n),$$

where $A_n = \frac{dF(\vec{y}(t_n))}{d\vec{y}}$ is is the Jacobian of $F(\vec{y}(t))$ and $R(\vec{y}(t))$ is the nonlinear remainder function. Furthermore,

$$\varphi_1(\lambda) = \frac{e^\lambda - 1}{\lambda}, \varphi_2(\lambda) = \frac{e^\lambda - \lambda - 1}{\lambda^2}, \varphi_3(\lambda) = \frac{e^\lambda(6-\lambda) - (6 + 5\lambda + 2\lambda^2)}{\lambda^3}. \tag{4.33}$$

To describe how EPI methods work, let us consider a nonlinear autonomous system of ordinary differential equations (ODEs)

$$\vec{y}' = F(\vec{y}), \quad \vec{y}(t_0) = \vec{y}_0. \tag{4.34}$$

The Taylor expansion of $F(\vec{y}(t))$ around $\vec{y}(t_n)$ yields the ODE

$$\frac{d\vec{y}}{dt} = F(\vec{y}(t_n)) + A_n(\vec{y}(t) - \vec{y}(t_n)) + R(\vec{y}(t)). \tag{4.35}$$

We can represent the solution of (5.2) as an integral by using an integrating factor $e^{-A_n t}$ and integrating (5.3) over the time interval $[t_n, t_{n+1}]$ to get

$$\vec{y}(t_{n+1}) = \vec{y}(t_n) + \left[e^{A_n \Delta t} - I\right] A_n^{-1} F(\vec{y}(t_n)) + \int_{t_n}^{t_{n+1}} e^{A_n(t_{n+1} - \tau)} R(\vec{y}(\tau))\, d\tau. \tag{4.36}$$

This can be approximated by numerical methods that use the approximation of products of matrix functions and vectors of the form $\varphi(A\tau)\vec{b}$, where φ is a function such as in (4.33), A

is a matrix such as A_n, τ is a scaling parameter based on the time step such as Δt, and $\vec{b}$ is a vector such as $F(\vec{y}_n)$.

EPI methods use Krylov projection to compute these products so that

$$\varphi(A\tau)\vec{b} \approx ||\vec{b}||_2 V_m \varphi(H_m \tau)\vec{e}_1, \tag{4.37}$$

where H_m is an upper Hessenberg matrix, which is given by $H_m = V_m^H A V_m$, and $V_m = \begin{bmatrix} v_1 & v_2 & \dots & v_m \end{bmatrix}$, with $\{v_1, v_2, ..., v_m\}$ an orthonormal basis of the Krylov subspace $K_m(A, \vec{b})$ that can be found using the Arnoldi algorithm [13]. The number of the Krylov vectors produced, the eigenvalues of A, the magnitude of τ, and φ determine how accurate the approximation of (4.37) is.

Instead of using Krylov projection to compute $\varphi(A\tau)\vec{b}$, we can use the KSS methods described earlier [8]. This is our combination of KSS and EPI methods. Time-stepping methods that use Krylov projection solve for the different frequency components of the solution with the same polynomial or rational function. The function must be high-degree for high resolution images, but this is computationally expensive. On the other hand, KSS methods treat the frequency components individually by utilizing a low-degree polynomial. An interpolating polynomial with frequency-dependent interpolation points such as $\lambda_{1,2}$ from (4.31) is used to approximate φ for each component. Therefore, components are solved for with different low-degree polynomials. This is possible because KSS methods are independent of resolution. The advantages of using KSS methods are a high order of accuracy and stability to higher resolution.

We can now implement the new numerical method by coding it in MATLAB. It will be used in the next chapter.

Chapter 5

Numerical Experiments

Now that we have our image processing models ready to solve with the numerical method designed, we can perform numerical experiments to enhance color images. Each image is processed with both models, tuning the parameters of each respective model for the best results. It was discussed in Chapter 3 that changing the parameters controls when the model denoises and sharpens.

The first two cases deal with denoising. Given a clean image u_0, Gaussian white noise with $\mu = 0$ and $\sigma^2 = 0.01$ is added so that we get a noisy image. Then we get noisy grayscale images for each red, green, and blue (RGB) channel $\tilde{u}_{i,0}$ for (3.3) and (3.5).

The enhancement results for the noisy images are now presented. Each model executes 50 time steps to get an enhanced image. In Figure 5.2, we see the results of processing an image of bell peppers (Figure 5.1) with Model 1 using $k = 1 \times 10^{-6}$, $\varepsilon = 0.1$, and $dt = 1 \times 10^{-7}$.

Figure 5.1: Clean, complete bell peppers image.

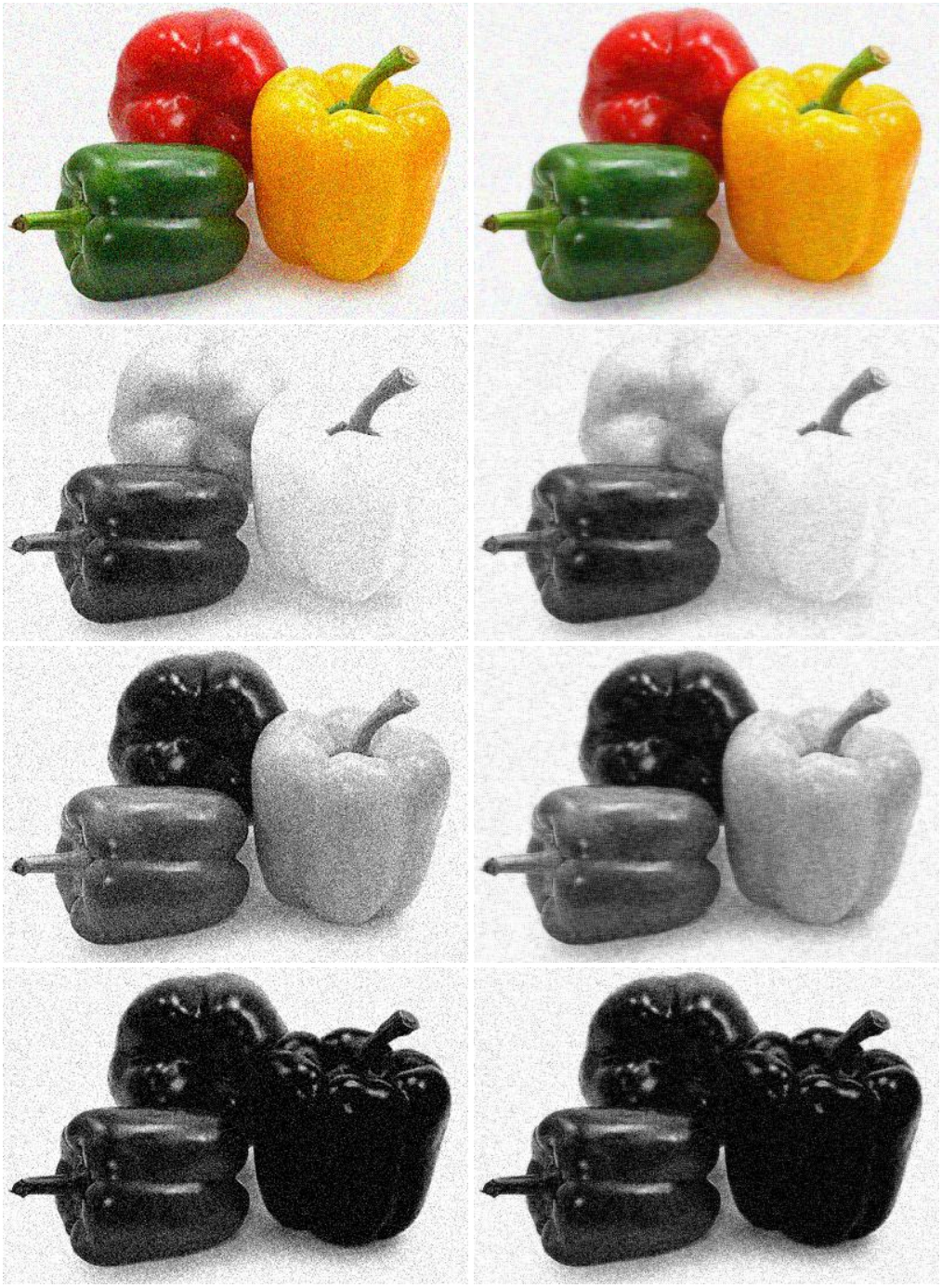

Figure 5.2: Enhancement of noisy bell peppers image with Model 1. **Left Column** - Noisy images, **Right Column** - Enhanced images corresponding to the images to the left. **First Row** - Complete image, **Second Row** - R channel, **Third Row** - G channel, **Fourth Row** - B channel.

We also illustrate our results with peak signal-to-noise ratio (PSNR) [14]

$$\text{PSNR}(f_0, f_1) = 20\log_{10}\left(\frac{255}{\sqrt{MSE(f_0, f_1)}}\right), \tag{5.1}$$

which is a measure of the deviation between two images. The PSNR value for a noisy image measures the deviation between the noisy image and the clean image, while the value for an enhanced image measures the deviation between the enhanced image and the clean image. A higher PSNR for an enhanced image indicates less deviation and, thus, more agreement with the clean image. Therefore, a high increase between the PSNR for a noisy image and the PSNR for an enhanced image is desired. Table 5.1 contains the PSNR values for denoisng the bell peppers image with Model 1. Since the Gaussian white noise added to a clean image is random, five trials are executed, and each PSNR for a noisy image is different.

	Trial	PSNR (noisy)	PSNR (enhanced)	PSNR increase
Channel 1	1	32.0219	32.1061	0.0842
	2	31.9992	32.0517	0.0525
	3	31.9565	32.0254	0.0689
	4	31.9100	31.9700	0.0600
	5	31.9983	32.0533	0.0550
Channel 2	1	33.6475	33.7650	0.1175
	2	33.5408	33.6406	0.0998
	3	33.4932	33.5731	0.0799
	4	33.4701	33.5687	0.0986
	5	33.4649	33.5743	0.1094
Channel 3	1	33.4168	33.4410	0.0242
	2	33.2953	33.3154	0.0201
	3	33.4224	33.4421	0.0197
	4	33.2413	33.2503	0.0090
	5	33.4933	33.5183	0.0250

Table 5.1: PSNR values for noisy images, PSNR values for enhanced images, and PSNR increases for five trials of denoising bell peppers image with Model 1.

The bell peppers image is processed with Model 2 using $k = 1 \times 10^{-15}$, $p = 1.0003$, and $\delta = 0.1$. Because δ changes over time [14], we let $\delta_{max} = 1$ so that δ increases linearly

to this value for $n_1 = 40$ time steps and $\delta_{min} = 0.1$ so that δ then decreases linearly to this value for $n_1 = 40$ time steps. The parameter $dt = 1 \times 10^{-7}$ is also used. The results are shown in Figure 5.3 and Table 5.2

	Trial	PSNR (noisy)	PSNR (enhanced)	PSNR increase
Channel 1	1	31.9367	32.0127	0.0760
	2	31.9782	32.0323	0.0541
	3	32.0265	32.0977	0.0712
	4	31.8109	31.8685	0.0576
	5	32.1349	32.0665	0.0684
Channel 2	1	33.4721	33.5663	0.0942
	2	33.4960	33.5748	0.0788
	3	33.4088	33.5046	0.0958
	4	33.6565	33.7389	0.0824
	5	33.5219	33.6109	0.0890
Channel 3	1	33.2477	33.2337	0.0140
	2	33.2998	33.3298	0.0300
	3	33.3118	33.3316	0.0198
	4	33.2491	33.2734	0.0243
	5	33.2473	33.2637	0.0164

Table 5.2: PSNR values for noisy images, PSNR values for enhanced images, and PSNR increases for five trials of denoising bell peppers image with Model 2.

The second image processed is an image of the Aubrey K. Lucas Administration Building on the University of Southern Mississippi campus (Figure 5.4). Model 1 uses the parameters $k = 1 \times 10^{-8}$ for the R and G channels and $k = 1 \times 10^{-5}$ for the B channel. Different values are chosen for k because 1×10^{-8} turns out to be too small for the B channel, so the best value for k is found to be 1×10^{-5} instead. The other parameters are $\varepsilon = 0.1$ and $dt = 1 \times 10^{-7}$. The results are shown in Figure 5.5 and Table 5.3. The parameters used for Model 2 are $k = 0.001$, $p = 1.0003$, $\delta = 0.1$, $\delta_{max} = 2$, $\delta_{min} = 0.1$, $n_1 = 40$, and $dt = 1 \times 10^{-7}$. The results are shown in Figure 5.6 and Table 5.4

In these experiments, effective denoising is observed. Model 1 and Model 2 have similar results for the bell peppers image. However, Model 2 performs much better than Model 1

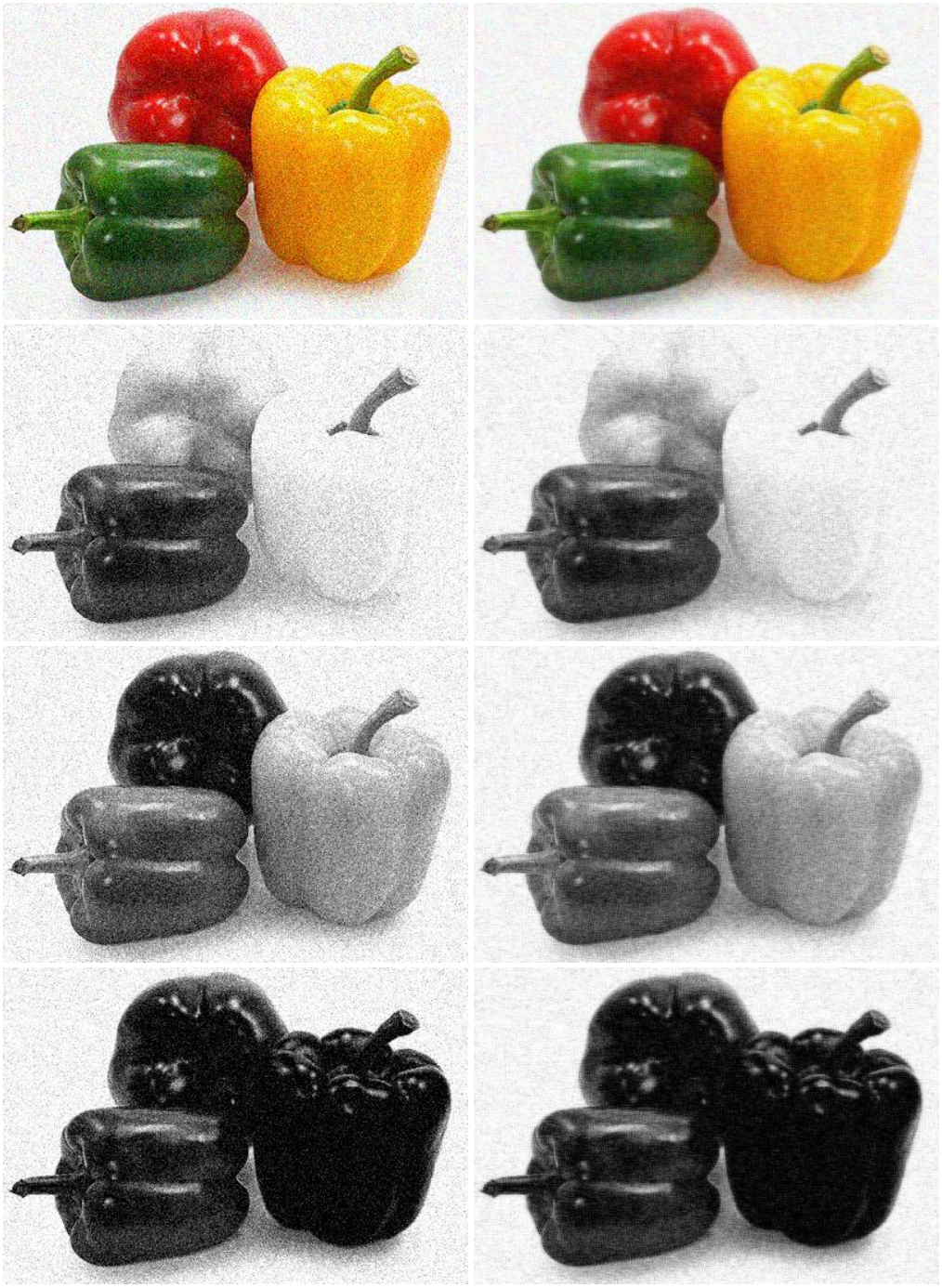

Figure 5.3: Enhancement of noisy bell peppers image with Model 2. **Left Column** - Noisy images, **Right Column** - Enhanced images corresponding to the images to the left. **First Row** - Complete image, **Second Row** - R channel, **Third Row** - G channel, **Fourth Row** - B channel.

Figure 5.4: Clean, complete Administration Building image.

	Trial	PSNR (noisy)	PSNR (enhanced)	PSNR increase
Channel 1	1	41.4047	42.1226	0.7179
	2	41.2990	42.9671	0.6681
	3	41.3980	41.9019	0.5039
	4	41.2949	41.9332	0.6383
	5	41.3818	42.5085	0.6767
Channel 2	1	41.3021	42.3049	1.0028
	2	41.2036	42.3132	1.1096
	3	41.3215	42.4603	1.1388
	4	42.2426	42.2911	1.0485
	5	41.3010	42.2130	0.9120
Channel 3	1	36.4976	36.5386	0.0410
	2	36.4180	36.4408	0.0228
	3	36.5060	36.5336	0.0276
	4	36.4253	36.4516	0.0263
	5	36.4707	36.5032	0.0325

Table 5.3: PSNR values for noisy images, PSNR values for enhanced images, and PSNR increases for five trials of denoising Administration Building image with Model 1.

for the Administration Building image. For both models, there is a smaller effect on the B channels than on the other channels.

The next case works with sharpening. Blur is added to a clean image with a convolution with a Gaussian kernel to produce a blurry image. We can then separate the image into blurry grayscale images for each RGB channel $\tilde{u}_{i,0}$ to use for (3.3) and (3.5). The image processed is a cyan flower (Figure 5.7). Because the cyan color only has pixel values from

Figure 5.5: Enhancement of noisy Administration Building image with Model 1. **Left Column** - Noisy images, **Right Column** - Enhanced images corresponding to the images to the left. **First Row** - Complete image, **Second Row** - R channel, **Third Row** - G channel, **Fourth Row** - B channel.

Figure 5.6: Enhancement of noisy Administration Building image with Model 2. **Left Column** - Noisy images, **Right Column** - Enhanced images corresponding to the images to the left. **First Row** - Complete image, **Second Row** - R channel, **Third Row** - G channel, **Fourth Row** - B channel.

	Trial	PSNR (noisy)	PSNR (enhanced)	PSNR increase
Channel 1	1	41.3559	42.5790	1.2231
	2	41.3381	42.5283	1.1902
	3	41.4198	42.2817	0.8619
	4	41.4715	42.4275	0.9560
	5	41.4261	42.2383	0.8122
Channel 2	1	41.2355	43.0414	1.8059
	2	41.2387	43.0139	1.7752
	3	41.2602	42.9655	1.7053
	4	41.1410	42.9904	1.8494
	5	41.2254	42.8645	1.6391
Channel 3	1	36.506	36.5452	0.0392
	2	36.4252	36.4819	0.0567
	3	36.4707	36.5212	0.0505
	4	36.45241	36.5245	0.0721
	5	36.5284	36.6021	0.0737

Table 5.4: PSNR values for noisy images, PSNR values for enhanced images, and PSNR increases for five trials of denoising Administration Building image with Model 2.

the G and B channels, the R channel is disregarded. As before, 50 time steps are taken in each model. The parameters with the best results for Model 1 are $k = 0.0001$, $\varepsilon = 0.1$, and $dt = 1 \times 10^{-7}$. PSNR usage for the enhancement of noisy images is adapted for blurry images. The results are in Figure 5.8 and Table 5.5.

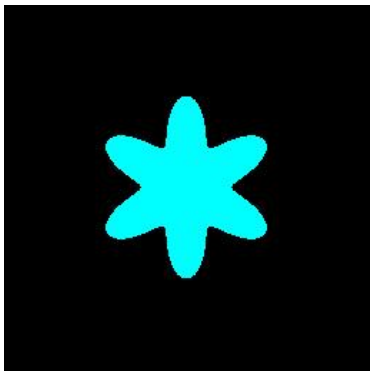

Figure 5.7: Clean, complete cyan flower image.

In order for Model 2 to sharpen, it needs to include how the image was blurred [14].

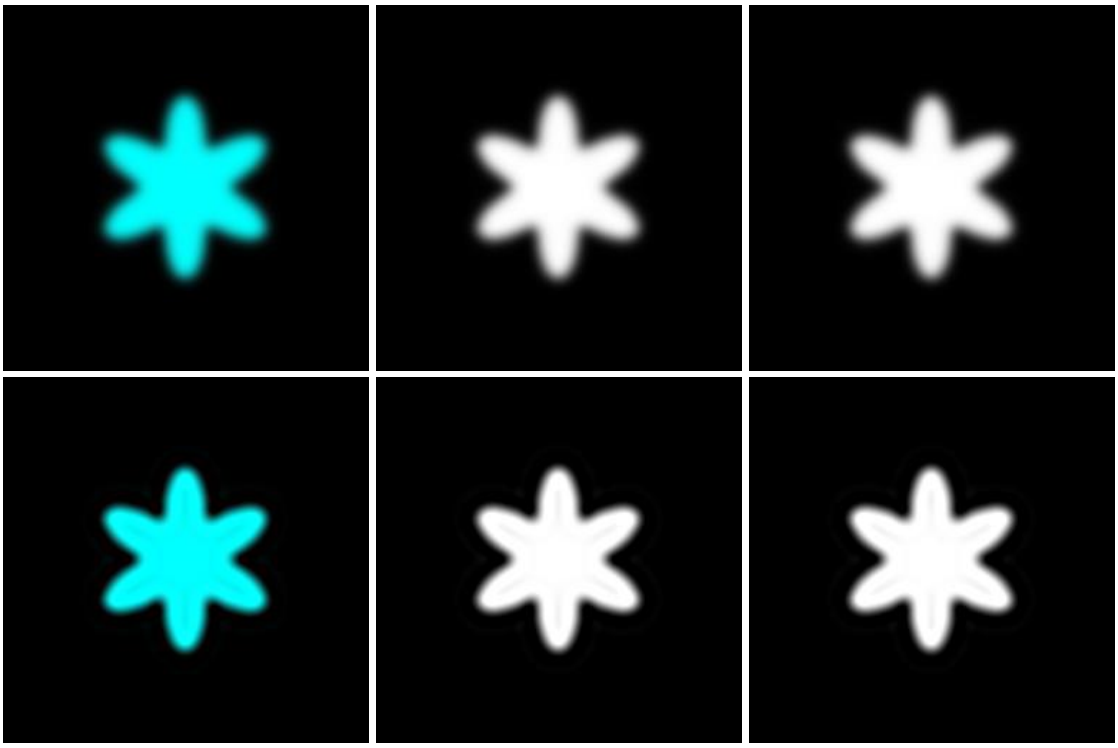

Figure 5.8: Enhancement of blurry cyan flower image with Model 1. **Top Row** - Blurry images, **Bottom Row** - Enhanced images corresponding to the above images. **First Column** - Complete image, **Second Column** - G channel, **Third Column** - B channel.

Channel	PSNR (blurry)	PSNR (enhanced)	PSNR increase
2	29.1259	33.1206	3.9947
3	29.2879	33.7377	4.4498

Table 5.5: PSNR values for blurry images, PSNR values for enhanced images, and PSNR increases for sharpening cyan flower image with Model 1.

Therefore, an extra term is added to (3.5) to get

$$\frac{\partial u_i}{\partial t} = \nabla \cdot \left[\left(\frac{1}{1+k^2|\nabla u_i|^2} + \delta |\nabla u_i|^{p-2} \right) \nabla u_i \right] + \lambda K'(\tilde{u}_{i,0} - K u_i), \quad i = R,G,B, \tag{5.2}$$

where λ is a scaling parameter, K is a blurring operator chosen to be a Gaussian kernel, and K' is the adjoint operator. Since Gaussian functions are radially symmetric, $K' = K$. The equation

$$\frac{\partial u_i}{\partial t} = \nabla \cdot \left[\left(\frac{1}{1+k^2|\nabla u_i|^2} + \delta |\nabla u_i|^{p-2} \right) \nabla u_i \right] + \lambda K' * (\tilde{u}_{i,0} - K * u_i) \quad i = R,G,B \tag{5.3}$$

is computed numerically with the blurred image $\tilde{u}_{i,0}$ as the input. Model 2 works best with the parameters $k = 0.001$, $p = 1.03$, $\delta = 1 \times 10^{-5}$ (with no need to change it over time), $n_1 = 40$ time steps, and $\lambda = 3 \times 10^6$. The results are shown in Figure 5.9 and Table 5.6.

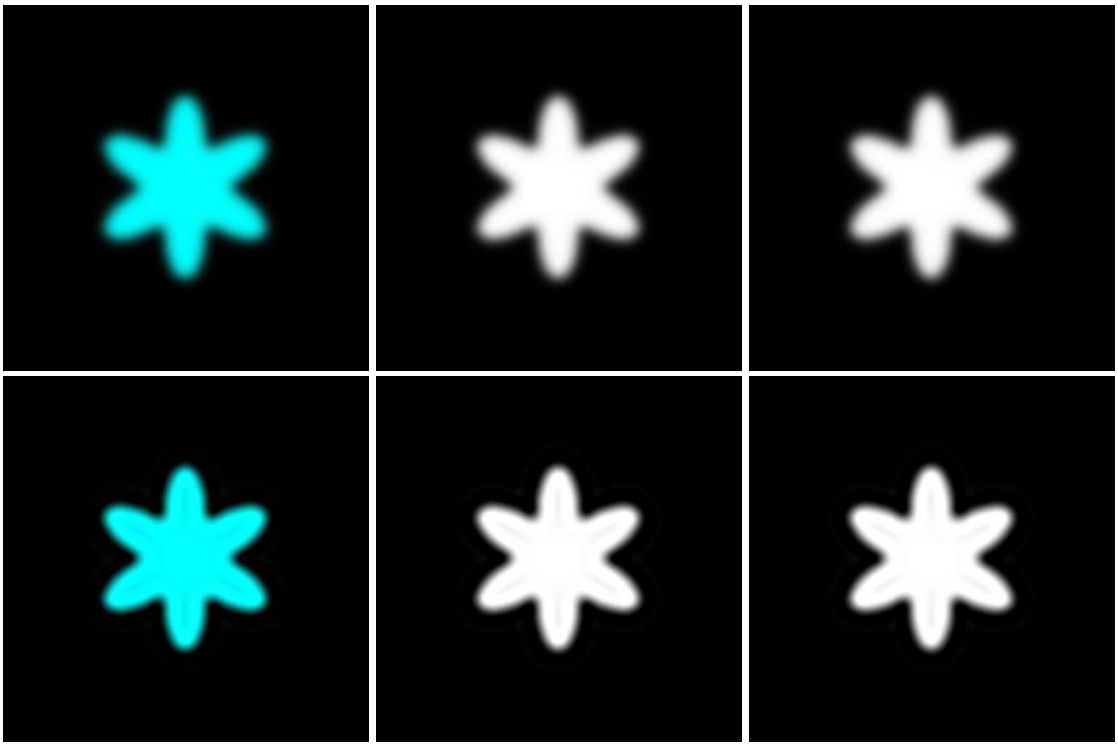

Figure 5.9: Enhancement of blurry cyan flower image with Model 2. **Top Row** - Blurry images, **Bottom Row** - Enhanced images corresponding to the above images. **First Column** - Complete image, **Second Column** - G channel, **Third Column** - B channel.

Channel	PSNR (blurry)	PSNR (enhanced)	PSNR increase
2	29.1259	33.1190	3.9931
3	29.2879	33.7352	4.4473

Table 5.6: PSNR values for blurry images, PSNR values for enhanced images, and PSNR increases for sharpening cyan flower image with Model 2.

The experiments show that both models effectively sharpen the blurry cyan flower image. Furthermore, all of the denoising and sharpening experiments confirm that separating the RGB channels leads to successful image enhancement.

Chapter 6

Conclusion

The two nonlinear diffusion image processing models designed in this project successfully enhanced color images, when treating their red, green, and blue (RGB) channels separately. This was possible because Krylov subspace spectral (KSS) methods were adapted to nonlinear diffusion to deal with high resolution and nonlinear partial differential equations (PDEs). By combining them with exponential propagation iterative (EPI) methods, a numerical method was developed to efficiently solve our models. Numerical experiments resulted in effective denoising and sharpening, demonstrating the usefulness of the proposed approach for image processing.

KSS and EPI methods were successfully used for the first time to solve nonlinear PDEs to high order accuracy, without having to use standard Krylov projection, unlike in [8]. This bodes well for using the methods to work with other nonlinear PDEs. Moreover, even higher order versions of the methods could be used for this project's PDEs to improve efficiency. Other improvements that could be made in the models include adaptive time stepping and automatic parameter selection. It may also be attempted to find faster ways to compute the frequency-dependent Gaussian quadrature nodes.

This project can be expanded by testing the models with different types of noise and blur in images. Executing more time steps could also be investigated to produce better results. Furthermore, the models could be compared to other existing models to see how well they perform and how efficient they are. Implementation in C++ would allow comparison to other models that use C++. These expansions would provide more insight into the performance of our models.

Bibliography

[1] Atkinson, K., *An Introduction to Numerical Analysis (2nd Ed.)*, Hoboken, NJ: John Wiley & Sons, Inc. (2013)

[2] Alvarez, L., Lions, P. L., Morel, J. M., "Image selective smoothing and edge detection by nonlinear diffusion. II", *SIAM Journal on Numerical Analysis* **29**(3) (1992), p. 845-866.

[3] Betounes, D., *Differential Equations: Theory and Applications (2nd Ed.)*, New York, NY: Springer (2009).

[4] Burden, R. L., Faires, J. D., Burden, A. M., *Numerical Analysis (10th Ed.)*, Boston, MA: Cengage Learning (2016).

[5] Catté, F., Lions, P. L., Morel, J. M., Coll, T., "Image selective smoothing and edge detection by nonlinear diffusion", *SIAM Journal on Numerical Analysis* **29**(1) (1991), p. 182-193.

[6] Chan, T. F., Kang, S. H., Shen, J., "Total variation denoising and enhancement of color images based on the CB and HSV color models", *Journal of Visual Communication and Image Representation* **12**(4) (2001), p. 422-435.

[7] Chen, Y., Bose, P., "On the incorporation of time-delay regularization into curvaturebased diffusion", *Journal of Mathematical Imaging and Vision* **14**(2) (2001), p. 149-164.

[8] Cibotarica, A., Lambers, J. V., Palchack, E. M., "Solution of nonlinear time-dependent PDE through componentwise approximation of matrix functions", *Journal of Computation Physics* **321** (2016), p. 1120-1143.

[9] Cottet, G. H., Ayyadi, M. E., "A Volterra type model for image processing", *IEEE Transactions on Image Processing* **7**(3) (1998), p. 292-303.

[10] Farlow, S. J., *Partial Differential Equations for Scientists and Engineers*, Mineola, NY: Dover Publications, Inc. (1993).

[11] Golub, G. H., Meurant, G., "Matrices, moments, and quadrature", *Numerical Analysis 1933: Proceedings of the 15th Dundee Conference, June–July 1993*, D. F. Griffiths, G. A. Watson Eds., Harlow, Essex, England: Longman Scientific & Technical (1994), p. 105-156.

[12] Golub, G. H., Underwood, R., "The block Lanczos method for computing eigenvalues", *Mathematical Software III*, J. R. Rice Ed., (1997), p. 361-377.

[13] Golub, G. H., Van Loan, C. F., *Matrix Computations (3rd Ed.)*, Baltimore, Maryland: The John Hopkins University Press, 1996.

[14] Guidotti, P., Kim, Y., Lambers, J. V., "Image restoration with a new class of forward-backward-forward diffusion equations of Perona-Malik type with applications to satellite image enhancement", *SIAM Journal on Imaging Sciences* **6**(3) (2013), p. 1416-1444.

[15] Guidotti, P., Lambers, J. V., "Two new nonlinear nonlocal diffusions for noise reduction", *Journal of Mathematical Imaging and Vision* **33**(1) (2009), p. 27-35.

[16] Gustafsson, B., Kreiss, H.-O., Oliger, J., *Time-Dependent Problems and Difference Methods (2nd Ed.)*, Hoboken, NJ: John Wiley & Sons, Inc. (2013).

[17] Lambers, J. V., "Enhancement of Krylov subspace spectral methods by block Lanczos iteration", *Electronic Transactions on Numerical Analysis* **31** (2008), p. 86-109.

[18] Moler, C., Loan, C. V., "Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later", *SIAM Review* **45**(1) (2003), p. 3-49.

[19] Moreno, J. C., Prasath, V. B. S., Neves, J. C., "Color image processing by vectorial total variation with gradient channels coupling", *Inverse Problems and Imaging* (2015). In press.

[20] Nitzberg, M. Shiota, T., "Nonlinear image filtering with edge and corner enhancement", *IEEE Transactions on Pattern Analysis and Machine Intelligence* **14**(8) (1992), p. 826-833.

[21] Palchak, E. M., Cibotarica, A., Lambers, J. V., "Solution of time-dependent PDE through rapid estimation of block Gaussian quadrature nodes", *Linear Algebra and its Applications* **468** (2015), p. 233-259.

[22] Perona, P., Malik, J., "Scale-space and edge detection using anistotropic diffusion", *IEEE Transactions on Pattern Analysis and Machine Intelligence* **12**(7) (1990), p. 161-192.

[23] Rudin, L. I., Osher, S., Fatemi, E., "Nonlinear total variation based noise removal algorithms", *Physica D* **60** (1992), p. 259-268.

[24] Tokman, M., "Efficient integration of large stiff systems of ODEs with exponential propagation iterative (EPI) methods", *Journal of Computational Physics* **213** (2006), p. 748-776.