2006

VARIATIONAL METHODS FOR IMAGE DEBLURRING AND DISCRETIZED PICARD'S METHOD

James H. Money
University of Kentucky, jmoney@ms.uky.edu

ABSTRACT OF DISSERTATION

James H. Money

The Graduate School

University of Kentucky

2006

VARIATIONAL METHODS FOR IMAGE DEBLURRING AND DISCRETIZED PICARD'S METHOD

ABSTRACT OF DISSERTATION

A dissertation submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy in the
College of Arts and Sciences
at the University of Kentucky

By

James H. Money

Lexington, Kentucky

Director: Dr. Sung Ha Kang, Department of Mathematics

Lexington, Kentucky

2006

ABSTRACT OF DISSERTATION

VARIATIONAL METHODS FOR IMAGE DEBLURRING AND DISCRETIZED PICARD'S METHOD

In this digital age, it is more important than ever to have good methods for processing images. We focus on the removal of blur from a captured image, which is called the image deblurring problem. In particular, we make no assumptions about the blur itself, which is called a blind deconvolution. We approach the problem by miniming an energy functional that utilizes total variation norm and a fidelity constraint.

In particular, we extend the work of Chan and Wong to use a reference image in the computation. Using the shock filter as a reference image, we produce a superior result compared to existing methods. We are able to produce good results on non-black background images and images where the blurring function is not centro-symmetric. We consider using a general L^p norm for the fidelity term and compare different values for p. Using an analysis similar to Strong and Chan, we derive an adaptive scale method for the recovery of the blurring function.

We also consider two numerical methods in this disseration. The first method is an extension of Picard's method for PDEs in the discrete case. We compare the results to the analytical Picard method, showing the only difference is the use of the approximation versus exact derivatives. We relate the method to existing finite difference schemes, including the Lax-Wendroff method. We derive the stability constraints for several linear problems and illustrate the stability region is increasing. We conclude by showing several examples of the method and how the computational savings is substantial.

The second method we consider is a black-box implementation of a method for solving the generalized eigenvalue problem. By utilizing the work of Golub and Ye, we implement

a routine which is robust against existing methods. We compare this routine against JDQZ and LOBPCG and show this method performs well in numerical testing.

KEYWORDS: Total Variation Image Deblurring, Semi-Blind Image Deconvolution, L^p-norm Fidelity term, Picard's Method, symmetric generalized eigenvalue problem

James H. Money

3 May, 2006

VARIATIONAL METHODS FOR IMAGE DEBLURRING AND DISCRETIZED PICARD'S METHOD

By

James H. Money

Dr. Sung Ha Kang
Director of Dissertation

Dr. Serge Ochanine
Director of Graduate Studies

3 May, 2006

RULES FOR THE USE OF DISSERTATIONS

Unpublished dissertations submitted for the Doctor's degree and deposited in the University of Kentucky Library are as a rule open for inspection, but are to be used only with due regard to the rights of the authors. Bibliographical references may be noted, but quotations or summaries of parts may be published only with the permission of the author, and with the usual scholarly acknowledgments.

Extensive copying or publication of the dissertation in whole or in part requires also the consent of the Dean of the Graduate School of the University of Kentucky.

DISSERTATION

James H. Money

The Graduate School

University of Kentucky

2006

VARIATIONAL METHODS FOR IMAGE DEBLURRING AND DISCRETIZED PICARD'S METHOD

DISSERTATION

A dissertation submitted in partial fulfillment of the
requirements of the degree of Doctor of Philosophy
at the University of Kentucky

By

James H. Money

Lexington, Kentucky

Director: Dr. Sung Ha Kang, Department of Mathematics

Lexington, Kentucky

2006

To my loving wife, Patti

ACKNOWLEDGMENTS

I would like to thank the members of my comittee for their patience with me. They include Russell Brown, Qiang Ye, and Grzegorz Wasilkowski. Most importantly, I would like to thank my advisor Sung Ha Kang for her time and support throughout the years.

I would also like to thank all my professors at University of Kentucky, especially Ed Enochs, Ronald Gariepy, Tom Hayden, Peter Hislop, and Ren-cang Li. Their classes shaped my time here and I will carry these experiences with me the rest of my life.

I would like to thank all my friends and family, who are too many to mention. Their encouragement through the years has helped me reach this goal.

Finally, I would like to thank my wife, Patti. Her patience over the years and her encouragement allowed me to finish the task I started. Without her help, I could not have done this.

I would like to thank the ACM for reprint permission for portions of Chapter 1 and 5, which originally appears in the *ACM Transactions on Mathematical Software* [51].

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

1 Preliminaries

The area of applied mathematics has grown tremendously over the past century. With the availability of low cost computing equipment in recent years, applied mathematics has expanded into many related fields. Among these are scientific computations, computational biology and physics, materials science, and computational geometry. In addition, other related subject areas, including robotics, engineering, and finance have influenced the recent trends in applied mathematics.

In the area of numerical analysis, work continues to grow, driven by problems being considered outside the discipline of mathematics. Numerical analysis includes many interesting topics. Problems include curve fitting techniques, numerical integration and differentiation, and solutions to differential equations. There are also routines for solving linear systems and eigenvalue problem, as well as optimization methods and combinatorial algorithms.

One of the related areas that has seen a recent surge of interest is the area of image processing. With the rise of digital cameras, scanners, and digital imagery, it is more important than ever to have good methods for processing this imagery. The digital images could be distorted or have noise which needs to be removed. Some researchers are interested in recovering important features contained in the image. For example, they may want to recover the location of buildings in satellite imagery or identify objects that are blocking a path for a mobile robot. Given that many images require massive amounts of storage space, we need to consider methods by which to compress the imagery in order to save on storage costs.

A good example of the problems encountered with digital imagery is given by NASA's Deep Impact mission spacecraft. This satellite was sent to study the innards of a comet, but one of the instrument cameras was not properly calibrated before it was sent into space. The result was all the imagery returned to earth was blurry and all the images would have to be corrected after the data was returned.

In this dissertation, we consider aspects of image processing and numerical methods. With respect to image processing, there are many approaches to solving the problem, including wavelet, stochastic, and partial differential equations(PDE) based approaches. We consider using the PDE based approach in this research. The PDE based methods have several advantages over some other methods. These methods regard images as continuous functions and use gradients, diffusion, and curvature in a natural way to model the images.

This approach also has some advantages when recovering sharp edges in images. For numerical schemes, we consider two methods. The first is a discrete version of Picard's Method, which we have adapted for PDE based problems. This method uses a simple recurrence relation to solve the PDE to any degree of accuracy. This recurrence relation automatically generates the power series expansion of the solution. The second method we consider is an eigenvalue solver that is implemented in a black-box routine. This method uses an inverse free procedure to avoid inverting the matrix in the calculation. With this inverse free formulation, we can solve some ill-conditioned problems to high accuracy.

1.1 Image Processing and Image Model

The field of image processing is broad and contains many interesting applications. Some of the common image processing areas are image restoration, compression, and segmentation. Many times, the size of the raw data for the images can require gigabytes of data storage. Researchers have developed routines to compress an image into a reversible form to save storage space [26, 27, 11]. In this area, there are methods for the compression via wavelets, using general compression schemes that are applicable to any type of file, and methods which allow some loss of data.

The area of segmentation distinguishes objects from the background in an image [53, 49, 67, 22]. This is particular useful for satellite imagery from an intelligence standpoint. It is also useful for identification purpose by using facial imagery in a database. Segmentation is used in robotics, where it is important to locate the correct objects to move or manipulate.

Another area of image processing is image restoration. In image restoration, a distorted image is restored to its' original form. This distortion is typically caused by noise in transmission, lens calibration, motion of the camera, or age of the original source of the image. We focus on image restoration in this dissertation.

Within image restoration, there are many tasks that researchers consider. There has been significant work on denoising, where noise is removed from the image [57, 28, 11, 17, 25, 42]. This noise could be from transmission problems or due to some atmospheric problem at the time the image was captured. There is image inpainting, which recovers missing areas from an image [8, 18, 16, 15, 20]. These missing regions may occur because of age of the original object that was photographed, or physical defects in the object. Another area in restoration is image deblurring [48, 32, 23, 75, 12, 42]. In this area, the objective is to recover the true image given a blurry image. We will focus on image deblurring in this dissertation.

There are many models for images. For example, there are wavelet based approaches [29,

11, 12, 25]. There are also stochastic based methods for processing images [64, 48, 36, 32]. A more detailed discussion of these and other areas can be found in [19, 21]. We focus on a PDE based image model, which is

$$u_0 = k \star u + \eta \tag{1.1}$$

where u_0 is the observed image, u is the true image, k is the blurring function or kernel, and η is the additive noise term. The domain of the image is called Ω and typically is a square or rectangle. The images we consider are greyscale images and thus the range of the functions are the reals. In equation (1.1), $\star$ indicates the convolution operator which is defined to be

$$(k \star u)(x, y) = \int_\Omega k(x - s, y - t)u(s, t)\, ds\, dt$$

We assume that $u \geq 0$, $k \geq 0$ and $\int_\Omega k(x, y)\, dx\, dy = 1$.

Using this PDE based model, we consider minimizing an energy functional. In particular, we minimize the number of oscillations in the image u by using the total variation norm. We enforce several constraints in order to obtain a solution close to the observed image, while recovering sharp edges in the true image. We also enforce several conditions so that our computed image results in a physically based solution.

1.1.1 Total Variation Minimizing Schemes

Prior to 1990, there were many attempts to model image processing using energy functionals with the L^2 norm on the gradient[34, 59, 74, 41]. In all these cases, the resulting linear system is easy to solve, but does not give the quality results one would desire. In addition, Rudin, et.al, in [57], noted that if one compared minimizing using $|\nabla u|$ versus $|\nabla u|^2$, with the same set of constraints, the $|\nabla u|$ based method generated superior results. Morel, et.al, in [10] also used the Euler-Lagrange form of total variation of the gradient and derived a motion by mean curvature PDE for denoising.

Thus, we consider using the total variation(TV) norm, which is defined to be

$$\|u\|_{TV(\Omega)} = \int_\Omega |\nabla u(x, y)|\, dx\, dy. \tag{1.2}$$

Then, we minimize (1.2), but subject to some constraints particular to the problem in order that the output image is not the constant valued image. In [57], Rudin, Osher and Fatemi

considered the denoising case, where the constraints are maintaining the mean value of the image or the noise having mean value zero, i.e.,

$$\int_\Omega u(x,y)\,dx\,dy = \int_\Omega u_0(x,y)\,dxdy \tag{1.3}$$

and the standard deviation of the noise is σ or

$$\frac{1}{2}\int_\Omega (u-u_0)^2\,dx\,dy = \sigma^2, \sigma > 0. \tag{1.4}$$

Hence, the energy functional they wished to minimize for the denoising case was

$$\min_u \|u\|_{TV(\Omega)} + \frac{\lambda_1}{2}\int_\Omega (u(x,y)-u_0(x,y))^2\,dx\,dy + \lambda_2\int_\Omega (u(x,y)-u_0(x,y))\,dxdy \tag{1.5}$$

where λ_1 and λ_2 are Lagrange multipliers to be chosen. If we apply the Euler-Lagrange equations to (1.5), we obtain

$$\nabla\cdot\left(\frac{\nabla u}{|\nabla u|}\right) - \lambda_1(u-u_0) - \lambda_2 = 0 \tag{1.6}$$

We note here that we eliminate the constraint λ_2 since we manually enforce the mean value of the output image u. Hence, we solve the PDE

$$\nabla\cdot\left(\frac{\nabla u}{|\nabla u|}\right) - \lambda(u-u_0) = 0 \tag{1.7}$$

Thus, we reduce (1.7) to solving the PDE

$$\begin{cases} u_t & = \nabla\cdot\left(\frac{\nabla u}{|\nabla u|}\right) - \lambda_1(u-u_0) \\ u(x,y,0) & = u_0(x,y) \\ \frac{\partial u}{\partial \nu} & = 0 \text{ on the boundary of } \Omega = \partial\Omega \end{cases} \tag{1.8}$$

by utilizing Time Marching on (1.6) until the PDE reaches steady state. We note that since the denominator of $\nabla\cdot\left(\frac{\nabla u}{|\nabla u|}\right)$ can be zero, we implement this term with a parameter $\beta \approx 0$, added to eliminate the singularity. That is, we implement this term as

$$\nabla\cdot\left(\frac{\nabla u}{\sqrt{u_x^2+u_y^2+\beta^2}}\right).$$

To summarize, when considering a TV minimizing scheme, we consider the constrained TV minimizing functional and apply the Euler-Lagrange equations and solve the corresponding Euler-Lagrange form for the minimizing functional.

1.1.2 Image Deblurring

One problem we consider is recovering the true image from a blurry and noisy observed image. It is also known as the image deblurring problem and has been studied for known and unknown blur kernels. We focus on the *blind deconvolution* dealing with unknown kernels.

The blind deconvolution problem has been studied by various researchers using different approaches. There are approaches using functional settings with different types of alternating minimization schemes. You and Kaveh [80] considered using an alternating scheme involving the H^1 norm for the kernel. Building on You and Kaveh's work, Chan and Wong [23] extended their idea to the TV norm for both the image and the kernel, noting that in many cases the kernel function has sharp edges (such as motion blur and out of focus blur). The authors used the Alternating-Minimization [23] method for image and kernel recovery. In [47], Lin, et. al., extended the TV functional [23] to include additional constraints on the kernel in the problem and used Bregman iteration to improve the result. More recent works deal with spatially variant blurs [54, 78] and non-local functional as in [42].

Another approach for deblurring is to apply various filtering techniques. In [32], Fish et. al. considered using the Richardson-Lucy algorithm to implement an alternating minimization scheme using Bayes's Theorem, and got improved results from Weiner filter blind deconvolution. Using partial differential equations is proposed by Osher and Rudin [56] via shock filter. This method reconstructs the edges by creating shock at inflection points and finds accurate edge locations. Alvarez and Mazorra [1] considered a similar approach but pre-conditioning the image with diffusion in order that it can handle denoising and deblurring simultaneously. Gilboa, et. al [38, 39] extended this idea by using a complex diffusion process to be robust against noise.

There is considerable work on combining several functionals in various image processing tasks. Chan, et. al, in [24] considered using blind deblurring with inpainting, and calculated the solution as a single method. Bar, et. al., in [5], considered coupling with an edge detection for Gaussian type kernels. In [6], the authors used the L^1 fidelity term to remove salt and pepper noise in a deblurring problem. In [4], the authors considered deblurring and impulse noise removal via a combination of the Mumford-Shah model and total variation models in

a multichannel setting, and in [7], the authors combined semi-blind image restoration with segmentation for parametric blur kernels.

Given the quantity of research in the area, Chan and Shen [19, 21] present an overview of image deblurring methods developed over the past two decades, which includes stochastic methods, Tikhonov regularizations including TV regularization, and wavelet based algorithms.

We focus on the model of You and Kaveh [80] and Chan and Wong [23] which is to consider minimizing the functional

$$\min_{k,u} \|k \star u - u_0\|^2_{L^2(\Omega)} + \lambda_1 \|k\| + \lambda_2 \|u\| \tag{1.9}$$

where the norms $\|k\|$ and $\|u\|$ in (1.9) are either the Sobolev norm for You and Kaveh or the TV norm for Chan and Wong. Here, we note that the kernel may have sharp edges as the image does, such as the case with motion or out of focus blurs. We focus on Chan and Wong's work in [23] by considering the minimizing functional

$$\min_{k,u} \|k \star u - u_0\|^2_{L^2(\Omega)} + \lambda_1 \|k\|_{TV(\Omega)} + \lambda_2 \|u\|_{TV(\Omega)} \tag{1.10}$$

Chan and Wong implement the method by considering the solution for k and u separately via the Euler-Lagrange forms

$$u(-x,-y) \star (u(x,y) \star k - u_0) - \lambda_1 \nabla \cdot \left(\frac{\nabla k}{|\nabla k|} \right) = 0 \tag{1.11}$$

for kernel k and for image u, they solve

$$k(-x,-y) \star (k(x,y) \star u - u_0) - \lambda_2 \nabla \cdot \left(\frac{\nabla u}{|\nabla u|} \right). = 0 \tag{1.12}$$

The solution is computed by alternately solving for k and then u, for several iterations until the solution converges, typically after three iterations. Initial values for the computation is u_0 for u and $\delta(x,y)$, which is the Dirac-delta distribution, for the initial kernel approximation.

1.2 Numerical Methods

Within numerical analysis, there are many different types of methods. For example, there are methods for locating roots, interpolating data points, solving differential equations, and

finding eigenvalues. We focus on two of these methods in this dissertation. The first method we consider is used for numerical solutions of PDEs. Within this area, there are a number of methods for solving PDEs. There are classical finite difference schemes [52] where each derivative is approximated using nearby points derived via Taylor's Theorem. There are also finite element methods [81] which handles more complex domains and uses the basis functions to approximate the solution. In this dissertation, we apply Picard's method [70] to the PDE to construct the power series approximation.

There are many methods for solving eigenvalue problems. In general, these can be divided into two types of methods. The first are direct solvers, such as the Power Method or the QR algorithm(see [37]). These methods work by directly computing the eigenvalues and eigenvectors for the matrix. These methods are designed for matrices that are dense and have relatively small dimension. The other class of methods are iterative schemes. These are generally used for large sparse matrices and only return a few of the eigenvalues. We focus on this class of methods in section 1.2.2. In particular, we focus on Krylov subspace methods, where we form a basis for the subspace and choose the best approximate to solve the larger problem.

1.2.1 Picard's Method

One way to find the solution of an ordinary differential equation is to apply Picard's Method. Picard's Method is a method that has been widely studied since its' introduction by Emile Picard in [60]. The method was designed to prove existence of solutions of ordinary differential equations(ODEs) of the form

$$\begin{cases} y'(t) &= f(t,y) \\ y(t_0) &= y_0 \end{cases}$$

by defining the recurrence relation based on the fact

$$y(t) = y_0 + \int_{t_0}^{t} f(s, y(s))\, ds.$$

The only assumptions that are made are f and $\frac{\partial f}{\partial y}$ are continuous in some rectangle surrounding the point (t_0, y_0). In particular, the recurrence relation is given by

$$\begin{cases} \phi^{(0)}(t) &= y_0 \\ \phi^{(n)}(t) &= y_0 + \int_{t_0}^{t} f(s, \phi^{(n-1)}(s))\, ds, \quad n = 1, 2, \ldots \end{cases}. \tag{1.13}$$

While the recurrence relation results in a straight-forward algorithm to implement on the computer, the iterates become hard to compute after a few steps. For example, consider the ODE

$$\begin{cases} y'(t) &= 1/y(t) \\ y(1) &= 1 \end{cases}$$

which has the solution $y(t) = \sqrt{2t-1}$. However, the Picard iterates are

$$\begin{aligned} \phi^{(0)}(t) &= 1 \\ \phi^{(1)}(t) &= 1 + \int_1^t 1\, ds = 1 + (t-1) = t \\ \phi^{(2)}(t) &= 1 + \int_1^t 1/s\, ds = 1 + \ln t \\ \phi^{(3)}(t) &= 1 + \int_1^t 1/(1+\ln s)\, ds \end{aligned}$$

and we note the last integral is difficult to calculate. Continuing beyond he fourth iterate only results in the increasing problems with calculating the integral. As a result, Picard's Method is generally not used in this form.

Parker and Sochacki, in [69], considered the same problem, but restricted the problem to an autonomous ODE with $t_0 = 0$ and f restricted to polynomial form. In this setting, the iterates result in integration consisting of polynomials. They also showed that the n-th Picard iterate is the MacLaurin polynomial of degree n for $y(t)$ if $\phi^{(n)}(t)$ is truncated to degree n at each step. This form of Picard's method is called the Modified Picard Method(MPM).

In [69], Parker and Sochacki showed that a large class of ODEs could be converted to polynomial form using substitutions and using a system of equations. While this class of ODEs is dense in the analytic functions, it does not include all analytic functions. They also showed one can approximate the solution by a polynomial system and the resulting error bound when using these approximations. Parker and Sochacki also showed that if $t_0 \neq 0$, one computes the iterates as if $t_0 = 0$ and then the approximated solution to the ODE is $\phi^{(n)}(t + t_0)$.

In [70], Parker and Sochacki showed that the ODE based method can be applied to partial differential equations(PDEs) when the PDE is converted to an initial value problem form for PDEs. The resulting solution from MPM is the truncated power series solution from the Cauchy-Kovelsky theorem[31].

Both the ODE and PDE versions of MPM are now used to solve a number of problems including some stiff ODEs. Rudmin[65] describes how to use the MPM to solve the N-Body problem for the solar system accurately. Pruett, et. al [61], analyzed how to adaptively

choose the timestep size and the proper number of iterates for a smaller N-Body simulation and when a singularity was present.

Carothers, et. al, in [9], have proved some remarkable properties of these polynomial systems. They constructed a method by which an ODE could be analytic but could not be converted to polynomial form. They provide a method to convert any polynomial system to a quadratic polynomial system and how to decouple an system of ODEs into a single ODE. Extending the work of Rudmin, they derive an algebraic method to compute the coefficients of the MacLaurin expansion using Cauchy products.

Warne, et. al. [77], computed an error bound when using the MPM that does not involve using the n-derivative of the function. This explicit *a-priori* bound was then used to adaptively choose the timestep size for several problems. They showed a way to generate the Pade approximation using the MacLaurin expansion from MPM.

Researchers extended the MPM to use parallel computations and adaptively choose the timesteps as the algorithm executes. In [50], the method is modified to include a generic form for ODEs and PDEs and allowed the computation in parallel for any system of equations using a generic text based input file. This method was later modified using the error bound result in [77] to choose adaptive timesteps while performing the parallel computations.

To illustrate the MPM, consider solving the ODE

$$\begin{cases} y'(t) &= y^2(t) \\ y(0) &= 1 \end{cases} \tag{1.14}$$

where the solution to (1.14) is given by $y(t) = -1/(t-1)$. The Modified Picard iterates are

$$\phi^{(0)}(t) = 1$$

$$\phi^{(1)}(t) = 1 + \int_0^t 1^2 \, ds = 1 + t$$

$$\phi^{(2)}(t) = 1 + \int_0^t (1+s)^2 \, ds = 1 + t + t^2 + t^3/3$$

We truncate to degree two since this is the second iterate and proceed as before, calculating

$$\phi^{(3)}(t) = 1 + \int_0^t (1 + s + s^2)^2 \, ds = 1 + t + t^2 + t^3 + \ldots$$

$$\phi^{(4)}(t) = 1 + \int_0^t (1 + s + s^2)^2 \, ds = 1 + t + t^2 + t^3 + t^4 + \dots$$

To compare this to the MacLaurin expansion, we get, in fact, that

$$y(t) = -1/(t-1) = 1 + t + t^2 + t^3 + t^4 + \dots$$

which matches precisely at each degree to the modified Picard iterates.

To highlight the implementation of MPM for PDEs, consider the Sine-Gordon equation

$$\begin{cases} u_{tt} = u_{xx} + \sin u \\ u(x,0) = \cos x \\ u_t(x,0) = 0 \end{cases}$$

The right hand side of this PDE is not in polynomial form. In particular, $\sin u$ is not polynomial. Let $z = u_t$, $v = \cos u$, and $w = \sin u$. Then, the corresponding system after substituting is

$$\begin{cases} u_t = z & u(x,0) = \cos x \\ z_t = u_{xx} + w & z(x,0) = 0 \\ v_t = -wz & v(x,0) = \cos(\cos x) \\ w_t = vz & w(x,0) = \sin(\cos x) \end{cases}$$

Since the right hand side is polynomial and equivalent to the Sine-Gordon equation, we call the Sine-Gordon equation **projectively polynomial**. The MPM is applied on the polynomial system.

1.2.2 Generalized Eigenvalue Problems and EIGIFP

We consider computing a few algebraically smallest or largest eigenvalues and their corresponding eigenvectors of the generalized eigenvalue problem

$$Ax = \lambda Bx \tag{1.15}$$

where A and B are large (and typically sparse) symmetric matrices and B is positive definite. This eigenvalue problem, sometimes referred to as a pencil eigenvalue problem for (A, B), arises in a large variety of scientific and engineering applications, see [37, 58, 66] for general discussions. The efficient solution of large scale eigenvalue problems is of great importance.

These large scale eigenvalue problems appear in structural analysis where it is important to know resonance frequencies. They are also important to analyzing linear circuits in electrical engineering. A prior version of this work appears in [51].[1]

Over the years, many numerical methods and software have been developed to solve large scale eigenvalue problems. These include publicly distributed programs; we refer to Bai et al. [3] for a comprehensive list of references and codes. Links to most of the publicly distributed programs can also be found in Bai et al.[3]. A large majority of the programs are based on the Lanczos algorithm, including ARPACK (implemented in the MATLAB built-in function `eigs`) [46] and `irbleigs` [2]. Methods of this type require inverting B and, when eigenvalues are badly separated, they typically need to use a shift-and-invert transformation which is not always feasible or efficient. Other programs such as JDQZ [33, 68], JDCG [55], and LOPBCG [45] do not require inverting B or a shift-and-invert transformation, but they appear to require more user inputs, such as an initial approximate eigenvalue or preconditioners.

The method we implement is called `eigifp` . The underlying algorithm of `eigifp` is an inverse free preconditioned Krylov subspace projection method developed by Golub and Ye [40]. In this method, we iteratively improve an approximate eigenvector x_k through the Rayleigh-Ritz projection on the Krylov subspace of dimension m generated by $A - \rho_k B$ and x_k, where $\rho_k = x_k^T A x_k / x_k^T B x_k$. The projection is carried out by constructing a basis for the Krylov subspace through an inner iteration, where the matrices A and B are only used to form matrix-vector products and $\mathcal{O}(m)$ vector memory is required. The method is proved to converge at least linearly and a congruence transformation can be implicitly applied to precondition the original eigenvalue problem so as to accelerate convergence. `eigifp` implements this preconditioned algorithm and has also incorporated several algorithmic enhancements and implementation details to arrive at an efficient black-box implementation.

Comparing with existing programs, `eigifp` possesses some important features that allow it to solve some difficult problems without any input from users. First, for the generalized eigenvalue problem (with $B \neq I$), `eigifp` does not require the inversion of B as most other methods do. Second, it includes the congruent transformation based preconditioning and this is done with no required user inputs such as a user supplied preconditioner or a shift (as in a shift-and-invert transformation). The program uses an incomplete LDL^T factorization of a shifted matrix $A - \sigma B$ to generate a preconditioner, where the shift σ is an approximate eigenvalue determined also by the program. With the use of the incomplete factorization,

[1] © 2005 ACM, Inc. Included here by permission.

the computational and memory cost of preconditioning can be controlled and the preconditioning is implemented in the code in a black-box fashion. Thus, `eigifp` will be most useful for problems where preconditioning by the standard shift-and-invert transformation is not feasible; but it can be competitive in other circumstances. Finally, `eigifp` is relatively simple in implementation with only one performance tuning parameter (i.e. the dimension of the Krylov subspace). This parameter can be either set by users or adaptively chosen by the program.

1.3 Organization

The dissertation is organized as follows. In Chapter 2, we derive the Semi-Blind method by separating the Euler-Lagrange equations into two functionals and using a reference image to approximate the true image. We utilize the shock filter for a reference image and analyze the error in computing the kernel when using the reference image. We briefly discuss the numerical implementation, and show how to refine the kernel output to improve the computed image. We show several examples of Semi-Blind method including images with non-black backgrounds and kernels that are non-centrosymmetric. We also compare the Semi-Blind method to the Chan and Wong model, showing the Semi-Blind method generates superior results.

Chapter 3 considers utilizing different fidelity terms. In particular, we use a general L^p fidelity term in the functional. We derive a one dimensional analysis for the relation of λ_1 with the size of the kernel and object in the image. From this analysis, we formulate an adaptive scale method for kernel recovery. We discuss the scale dependence when using the L^1 fidelity term for image recovery and compare it to L^2 image recovery. A detailed comparison of using the various L^p norms for the kernel and image recovery in discussed for the blind and Semi-Blind method. We present several examples when using the L^1 norm for the fidelity term, including an example with non-centrosymmetric kernel.

We develop the Discretized Picard's Method in Chapter 4 by using the discrete data directly in computations. This method is implemented by using linear operators to approximate the derivatives. We discuss how the Discretized Picard's Method approximates the derivatives of each term for the Modified Picard Method. We show that the first and second iterates are the standard forward time difference and Lax-Wendroff schemes, respectively. We provide a method to generate the stability constraints for several examples in one and two dimensions. We illustrate those examples and show that the method is accurate and computationally competitive.

Chapter 5 covers the black-box implementation of `eigifp` . We review the basic method of Golub and Ye and show how to implement the preconditioning scheme for the method. We discuss the LOBPCG type enhancements to the subspace and how to deflate for finding interior eigenvalues. The implementation specific issues for the black-box routine are covered and some numerical details are provided. A detailed numerical comparison with LOBPCG and JDQZ is completed and shows the method is competitive against those routines.

2 Semi-Blind Total Variation Deconvolution

2.1 TV Minimizing Blind Deconvolution

In this chapter, we extend the Chan and Wong[23] minimizing functional with an reference image to improve image results and the computational cost for the blind deconvolution. In [23], Chan and Wong consider the minimizing functional

$$\min_{k,u} \|k \star u - u_0\|^2_{L^2(\Omega)} + \lambda_1 \|k\|_{TV(\Omega)} + \lambda_2 \|u\|_{TV(\Omega)} \tag{2.1}$$

and solve the corresponding Euler-Lagrange forms for (2.1). For the kernel k, they solve

$$u(-x,-y) \star (u(x,y) \star k - u_0) - \lambda_1 \nabla \cdot \left(\frac{\nabla k}{|\nabla k|} \right) = 0 \tag{2.2}$$

and for the image u, they solve

$$k(-x,-y) \star (k(x,y) \star u - u_0) - \lambda_2 \nabla \cdot \left(\frac{\nabla u}{|\nabla u|} \right) = 0. \tag{2.3}$$

In order to solve this minimization problem, Chan and Wong iteratively solve for equation (2.2) and (2.3) by starting with the initial conditions $u = u_0$ and $k = \delta(x,y)$. Their method is summarized below in Algorithm 2.1 and is called the Alternating Minimization(AM) method.

Algorithm 2.1 Alternating Minimization Method

Require: u_0, the input image
  Set $u^{(0)} = u_0$, $k^{(0)} = \delta(x,y)$.
  for n from 1 to n_{max} **do**
    Solve $u^{(n-1)}(-x,-y) \star (u^{(n-1)}(x,y) \star k^{(n)} - u_0) - \lambda_1 \nabla \cdot \left(\frac{\nabla k^{(n)}}{|\nabla k^{(n)}|} \right) = 0$ for $k^{(n)}$.
    Solve $k^{(n)}(-x,-y) \star (k^{(n)}(x,y) \star u^{(n)} - u_0) - \lambda_2 \nabla \cdot \left(\frac{\nabla u^{(n)}}{|\nabla u^{(n)}|} \right) = 0$ for $u^{(n)}$.
  end for

We consider the functional (2.1) and separate it into two functionals that are equivalent when solved via the Euler-Lagrange forms (2.2) and (2.3). The first functional corresponding to equation (2.2) is

$$\min_k \frac{1}{2} \|k \star u - u_0\|^2_{L^2(\Omega)} + \lambda_1 \int_\Omega |\nabla k| \, dx \, dy \tag{2.4}$$

and the corresponding functional for (2.3) is

$$\min_u \frac{1}{2}\|k \star u - u_0\|^2_{L^2(\Omega)} + \lambda_2 \int_\Omega |\nabla u|\, dx\, dy. \tag{2.5}$$

We note that in Figure 2.1, in image (a) is the output of the AM method for one iteration using u_0 for the input image. In image (b), we see the same output using the true image u, which results in a substantially better result and close to the true answer. In light of this fact, we consider using a reference image between u and u_0 that approximates the true image but is not computationally difficult to compute. Instead of functional (2.4), we consider using a

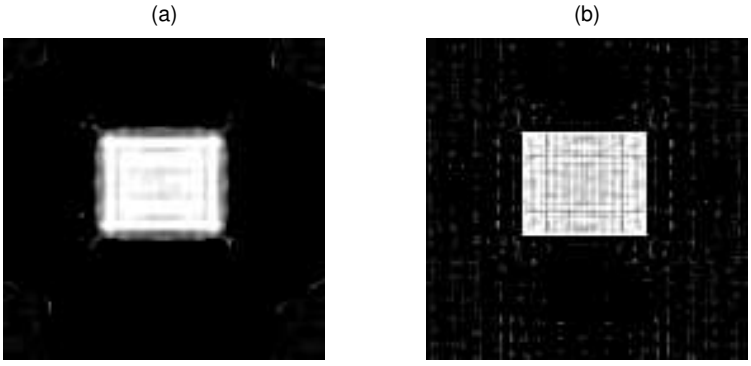

Figure 2.1: Comparison of initial value for u in Algorithm 2.1. In (a) we used $u = u_0$, the blurry image. In (b), u was the true image. Note that the improved results, as expected, when using the true image.

reference image u_r instead of u and get the functional

$$\min_k \frac{1}{2}\|k \star u_r - u_0\|^2_{L^2(\Omega)} + \lambda_1 \int_\Omega |\nabla k|\, dx\, dy \tag{2.6}$$

and then solve the Euler-Lagrange form

$$u_r(-x,-y) \star (u_r(x,y) \star k - u_0) - \lambda_1 \nabla \cdot \left(\frac{\nabla k}{|\nabla k|}\right) = 0. \tag{2.7}$$

Then, we solve equation (2.3) for u and no iteration is required. Since we utilize a reference image for kernel reconstruction in (2.6), we refer to this method as the *Semi-Blind decon-*

volution. However, we are solving a blind deblurring problem where we do not assume any particular properties about the kernel nor the image except for the fact we want them to be BV functions. The Semi-Blind method is summarized in Algorithm 2.2.

Algorithm 2.2 Semi-Blind Method

Require: u_0, the input image

Compute reference image u_r.

Solve $u_r(-x,-y) \star (u_r(x,y) \star k - u_0) - \lambda_1 \nabla \cdot \left(\frac{\nabla k}{|\nabla k|} \right) = 0$ for k.

Solve $k(-x,-y) \star (k(x,y) \star u - u_0) - \lambda_2 \nabla \cdot \left(\frac{\nabla u}{|\nabla u|} \right) = 0$ for u.

Output image u.

The method is illustrated in Figure 2.2. The reference image u_r in (c) need not be a perfect deblurred result. However, combined with the k and u minimization functionals (2.6) and (2.5), the result of the Semi-Blind method in in (d) recovers sharp edges. Note that reference image does provide good information on the location of edges in the image. With this correct information in u_r, the minimization scheme results in a good reconstruction of the kernel k, which results in turn with a good reconstructed image.

2.2 Reference Image and Shock Filter

Many different types of reference images may be used for u_r in (2.7). In this dissertation, we consider using Rudin and Osher's shock filter [56]. The idea behind the shock filter in one dimension is to reverse the effects of applying the Gaussian operator to a curve with a jump. Figure 2.3 illustrates the idea. In (a), we see the effect of smoothing the sharp edges curve. In (b), we see the opposite effect, which is result of the shock filter.

In one dimension, the shock filter takes the PDE form

$$u_t = -|u_x| F(u_{xx})$$

where $F(x)$ satisfies

$$\begin{cases} F(x) > 0 & \text{if } x > 0 \\ F(x) = 0 & \text{if } x = 0 \\ F(x) < 0 & \text{if } x < 0 \end{cases} \tag{2.8}$$

Clearly, $sign(x)$ satisfies (2.8) and is chosen to be the operator in [56]. We note here that when u_{xx} is zero, we have reached the point of inflection of the Figure 2.3(a) or the edge in

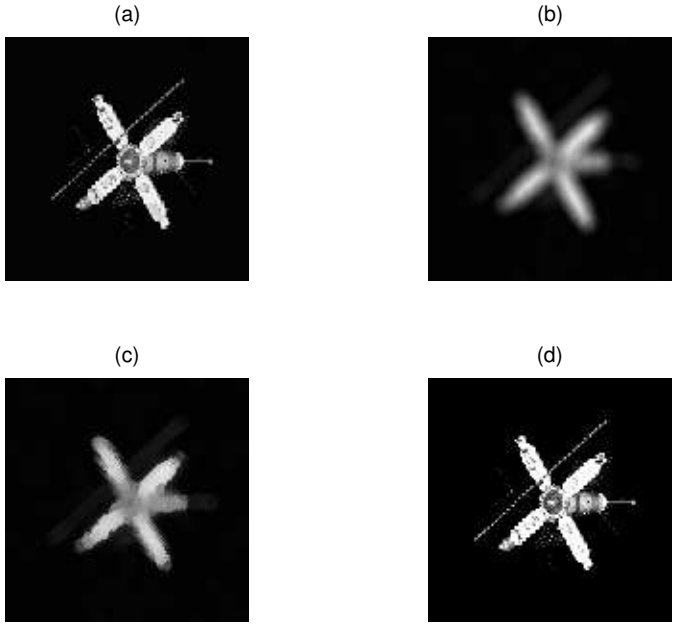

Figure 2.2: Illustration of the Semi-Blind method in Algorithm 2.2. (a) True image (b) Blurry given original image u_0 (c) Reference image u_r using shock filter (d) Semi-Blind method output

Figure 2.3(b). The extension to two dimensions is to consider the PDE

$$u_t = -|\nabla u| F(\mathcal{L}(u)) \tag{2.9}$$

where F is chosen the same as before, but $\mathcal{L}(u)$ is chosen to be a second order differential operator that detects the edges in the image. An easy choice for $\mathcal{L}$ is

$$\mathcal{L}(u) = \Delta u,$$

but this does not detect edges along the nontangental directions. A better choice for $\mathcal{L}$ is

$$\mathcal{L}(u) = \nabla u \cdot \begin{bmatrix} u_{xx} & u_{xy} \\ u_{yx} & u_{yy} \end{bmatrix} \nabla u = u_x^2 u_{xx} + 2u_x u_y u_{xy} + u_y u_{yy} \tag{2.10}$$

which is a second order term in the direction of the gradient and the $\mathcal{L}$ that Osher and Rudin employ in [56].

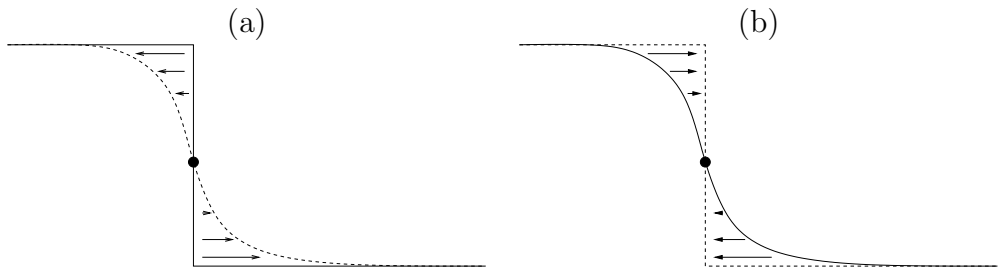

Figure 2.3: Illustration of the Shock Filter.Image (a) is the process of smoothing a sharp edge. Image (b) is the process of reversing the smooth via the shock filter.

Alvarez and Mazorra [1] considered a similar approach but pre-conditioned the image with diffusion in order that it can handle denoising and deblurring simultaneously. Gilboa, et. al.[39, 38] extended this idea by using a complex diffusion process to be robust against noise. Figure 2.4 illustrates the differences when using the diffusion process with a blurry and noisy image. The blurry and noisy image is in (a). In image (b) is the shock filter result with no diffusion applied. We see that the shock filter enhances the noise. In image (c), we see the same result, but with diffusion applied to the image before the shock filter was applied. The noise is not present in image (c) and the edges remain well defined despite the smoothing applied to the image. As a result, we use diffusion as needed in our calculations when either noise is present or the kernels are non-Gaussian.

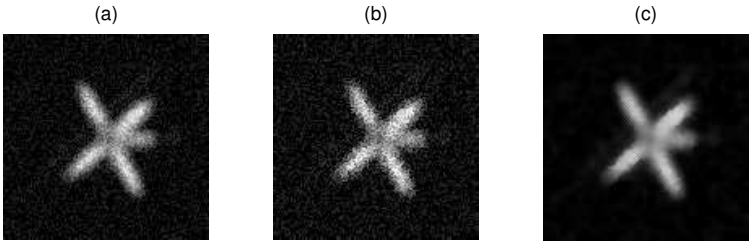

Figure 2.4: Pre-applying diffusion before using the shock filter. (a) The blurry and noisy image. (b) Shock filter result with no diffusion. (c) Shock filter result with diffusion pre-applied to the blurry and noisy image. Note the edges are well defined and noise has been removed in (c) but not in (b).

The shock filtered image u_r gives good clues for the edge information and the position of objects in the image as well as the shape of the kernel k. The convolution $k \star u$ is only affected when the value of u changes such as at the edges. Therefore, it is important that the

edge information is correct. The location of the edges in u_r determines the relative position of the kernel k as well.

We utilized the Lagged Diffusivity Fixed Point Method (see [75, 76]) for solving Euler-Lagrange equations (2.7) and (2.3). This is a scheme where the denominator for the TV term is lagged by one step and iterated until convergence. We denote the lagged TV term by L such that

$$Lk^n = -\nabla \cdot \left(\frac{\nabla k^n}{|\nabla k^{n-1}|} \right).$$

The following theorem further illustrates the importance of having a good reference image u_r.

Theorem 2.2.1 *Assume the reference image is a perturbation of the true image, i.e., $u_r = u + \delta u$, then the error in the computed kernel using one step of the Lagged Diffusivity Fixed Point method is*

$$E = \delta U^T U + \delta U^T \delta U$$

where U and δU are the convolution matrix operators for u and $\delta u = u_r - u$, respectively. Hence the error bound is

$$\frac{\|\delta k\|_2}{\|k\|_2} \leq \| \left(U^T U - \lambda_1 L \right)^{-1} \|_2 \left(\|\delta U\|_2^2 + \|U\|_2 \|\delta U\|_2 \right) \tag{2.11}$$

where $\| \cdot \|_2$ denotes the vector two norm.

Proof. The Lagged Diffusivity Fixed Point Method solves

$$U^T(Uk - u_0) - \lambda_1 Lk = 0$$

which is

$$\left(U^T U - \lambda_1 L \right) k = U^T u_0.$$

If $A = U^T U - \lambda_1 L$ and $b = U^T u_0$, we get $Ak = b$. In fact, we solve the perturbed system

$$\left((U + \delta U)^T (U + \delta U) - \lambda_1 L \right) k = (U + \delta U)^T u_0$$

or

$$\left(U^T U - \lambda_1 L + \delta U^T (U + \delta U) \right) k = U^T u_0 + \delta U^T u_0$$

Thus, $\delta A = \delta U^T (U + \delta U)$ and $\delta b = \delta U^T u_0$, where $(A + \delta A)k = b + \delta b$. By rearranging the

terms, we get $A(k+\delta k) = b + \delta b$ where $A\delta k = \delta A k$, hence $\delta k = A^{-1}\delta A k$. Therefore,

$$\begin{aligned}\|\delta k\|_2 &\leq \|A^{-1}\|_2 \|\delta A\|_2 \|k\|_2 \\ &= \| \left(U^T U - \lambda_1 L\right)^{-1} \|_2 \|\delta U^T (U + \delta U) \|_2 \|k\|_2\end{aligned}$$

Thus, the relative error is bounded by equation (2.11). □ ∎

Note that the error of the kernel is proportional to the error in approximating the true image u by u_r. If the approximation error for u_r is large, the kernel might contain large amounts of error. In addition, the condition number of $\| \left(U^T U - \lambda_1 L\right)^{-1} \|_2$ can also affect the error; therefore, λ_1 should be chosen properly. A badly chosen λ_1 will ill-condition the matrix $U^T U - \lambda_1 L$ resulting in poor results for the kernel. Since u_r is calculated via the shock filter, the edge information and position is close to the true image, resulting in a small δU term throughout the image. As a result, the Semi-Blind method results in a small error in the computation of the kernel function.

2.3 Numerical Implementation

There are a wide range of models and algorithms for numerical computation for image deblurring problems [14, 75, 76]. In [75], various methods for recovery of a blurry image are discussed, including a chapter dedicated to the TV based recovery of the image using fixed point methods. Vogel and Oman [76] present a more detailed analysis for the denoising case for the methods comparing the various fixed point schemes. In [14], the authors present a comparison of the fixed point, time marching, and primal-dual based methods.

In computation of the reference image u_r, we first use heat equation to smooth the image (implemented by centered differences). We then applied the shock filter, as implemented in [56],

$$\begin{aligned}u_{ij}^{n+1} = \; & u_{ij}^n - \frac{\Delta t}{h}\sqrt{((\Delta_+^x u_{ij}^n)^+)^2 + ((\Delta_-^x u_{ij}^n)^-)^2 + ((\Delta_+^y u_{ij}^n)^+)^2 + ((\Delta_-^y u_{ij}^n)^-)^2}\, sign^+(L(u^n)) \\ & -\frac{\Delta t}{h}\sqrt{((\Delta_+^x u_{ij}^n)^-)^2 + ((\Delta_-^x u_{ij}^n)^+)^2 + ((\Delta_+^y u_{ij}^n)^-)^2 + ((\Delta_-^y u_{ij}^n)^+)^2}\, sign^-(L(u^n))\end{aligned}$$

where $\Delta_+^x u_{ij} = u_{i+1,j} - u_{ij}$ and $\Delta_-^x u_{ij} = u_{ij} - u_{i-1,j}$ and similarly for $\Delta_\pm^y$. The superscripts means $F^+ = max(0, F)$ and $F^- = min(0, F)$. For $L(u^n) = u_{xx}u_x^2 + 2u_{xy}u_x u_y + u_{yy}u_y^2$, the centered difference schemes are used for the second order derivatives and the minmod scheme for the first order derivatives, $minmod(a, b) = \dfrac{sign(a) + sign(b)}{2} min(|a|, |b|)$.

For the computation of equations (2.7) and (2.3), we utilized the Lagged Diffusivity Fixed Point method [75, 76], which is to iterate on n for

$$u_r(-x,-y) \star \left(u_r \star k^{n+1} - u_0\right) - \lambda_1 \nabla \cdot \left(\frac{\nabla k^{n+1}}{|\nabla k^n|}\right) = 0$$

for (2.7) and similarly for (2.3). We utilized the FFT to compute the convolutions and used the conjugate gradient method to solve the corresponding linear systems at each step. We compute 10 iterates using the fixed point scheme, and stop if the desired residual is attained. For further details on implementing the fixed point scheme, see [23].

2.3.1 Kernel Refinements

When calculating the blur kernel k from (2.7), there is often noise in the solution as well as some Gaussian decay at the edge of the kernel support region. Figure 2.5(a) illustrates the typical output when using an out of focus blur. We wish to first eliminate the Gaussian characteristics of the kernel by enforcing the sharp edges in the kernel k. In order to enforce the piecewise nature of the blur kernel, we first apply the shock filter via equation (2.9) using the kernel k for the input function. The result is shown in Figure 2.5(b). If we do not have a Gaussian kernel, we may omit this step.

However, in Figure 2.5(b) we see that there is some noise at the base of the kernel where the blur kernel k should be zero. In order to eliminate this effect from the shock filter, we can adaptively denoise the output from the shock filter for k. In order to do this without adding an extra parameters to the algorithm, we use the adaptive TV denoising method of Strong and Chan in [73]. In order to estimate the scale of the kernel, we threshold the kernel for the size calculations only, and use the cutoff value to be

$$\text{cutoff} = \frac{min(k) + max(k)}{2}$$

and use this to estimate D, which is the size of the support of the kernel. Then to estimate the boundary of D, ∂D, we find the locations of large gradient changes and add up the length. Then, the scale of the kernel is given by

$$\text{scale} = \frac{|D|}{|\partial D|}$$

Using this information, we can apply the adaptive scale method for denoising, which involves

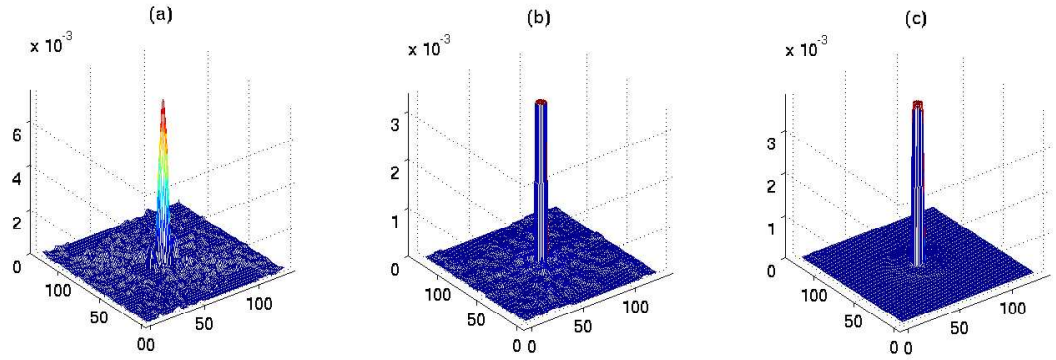

Figure 2.5: Comparison of unrefined and refined kernels. (a) is the output from (2.7). Note the smoothness and noise at the base. (b) is the output after applying the shock filter. (b) is the output after adaptive denoising. Note that this is similar to the true kernel.

running TV denoising via equation (1.6) using a small λ. Then, we can form the shifts in the output kernel by writing

$$\delta_{ij} = k_{ij} - TV(k)_{ij}$$

where TV is the output from the TV based denoising routine in section 1.1.1. Next, we compute λ via

$$\lambda = \delta_{ij} \frac{|D|}{|\partial D|}.$$

Lastly, we run the denoising algorithm with a pixel based λ on k. We note here that one can also use the D to be $\Omega - supp(k)$, and get similar results. The final output for the kernel after denoising is shown in Figure 2.5(c). We see the kernel takes on the shape of the out of focus blur as we wanted and the noise has been removed from the kernel that existed outside the support of k.

This adaption of the Semi-Blind Method is call Semi-Blind with Kernel Refinements and is detailed in Algorithm 2.3.

In Figure 2.6, we present a comparison between reconstructed images using Semi-Blind method with or without kernel refinements. Image (c) is the result without kernel refinement,

Algorithm 2.3 Semi-Blind Method with Kernel Refinements

Require: u_0, the input image

Compute reference image u_r via $\begin{cases} u_t = -|\nabla u| sign(\mathcal{L}(u)) \\ u(x,y,0) = u_0 \end{cases}$.

Solve $u_r(-x,-y) \star (u_r(x,y) \star k^{(1)} - u_0) - \lambda_1 \nabla \cdot \left(\frac{\nabla k^{(1)}}{|\nabla k^{(1)}|} \right) = 0$ for k.

Apply the shock filter on $k^{(1)}$ to get $k^{(2)}$.

Compute $|D|$ and $|\partial D|$.

Denoise $k^{(2)}$ with small λ to get $k^{(3)}$.

Compute $\delta_{ij} = k^{(2)} - k^{(3)}$.

Set $\lambda_{ij} = \delta_{ij} * \frac{|D|}{|\partial D|}$.

Denoise with λ_{ij} to get $k^{(4)}$.

Solve $k^{(4)}(-x,-y) \star (k^{(4)}(x,y) \star u - u_0) - \lambda_2 \nabla \cdot \left(\frac{\nabla u}{|\nabla u|} \right) = 0$ for u.

Output image u.

and image (d) shows the reconstruction with kernel refinement. These results are compared to Chan and Wong's TV blind deconvolution using AM method, which is shown in image (b). It shows a compatible results between [23] and Semi-Blind method without refinement on k in image (c), while there is an improvement in Semi-Blind method, since more of the antenna is visible in image (c). Since the two functionals (2.4) and (2.5) are separated, image (c) can be further refined to image (d). Image (d) shows dramatic improvement in the result, and it is very close to true image.

2.3.2 Examples

For all of the following experiments, we used out of focus blurs for the kernel unless otherwise noted. All the images are grey scale, and sized 127 x 127. The computations were performed on a 2.0 Ghz Pentium 4 machine using Matlab.

Our first example is presented in Figure 2.7. It shows the deblurring result with a black background and exhibits good results with the kernel improvements when using the L^2 norm for both u and k.

Figure 2.8 shows an experiment with non-black background. Since the background is not black, when computing the kernel, errors in the approximation compounds instead of disappearing. This is shown in Figure 2.8(c) where many wrinkles are presented in the result from using [23]. Image (d) shows Semi-Blind results with refinements, and it suffers much less from this effect. The recovered image (d) is close to the true image, and even recovers the reflection on the cup. The light shadows of the cup in the true image are emphasized in

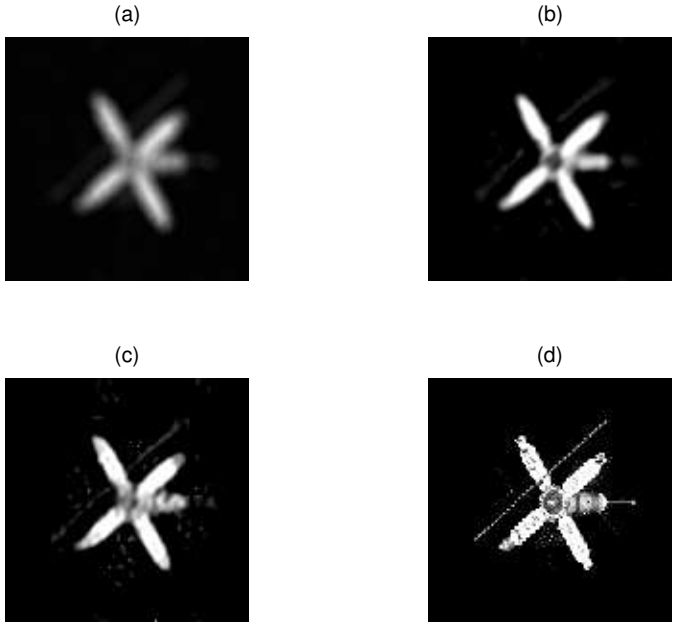

Figure 2.6: Comparison between AM method [23], Semi-Blind methods with or without kernel refinement. (a) The given blurry original image u_0. (b) Chan and Wong's AM method [23] (c) Semi-Blind result without kernel refinements. (d) Semi-Blind result with kernel refinements. Image (d) shows dramatic improvement in the reconstructed image u.

the result (d), and there is a contrast shift. Nevertheless, it recovers the sharp edges with details from given original image (b).

This Semi-Blind model can be applied to blurry image with noise. Figure 2.9 shows an example when the noise is added after blurring the image. In the result image (d), some details are lost due to denoising effect of choosing large λ_2 in (2.5). However, recovers reasonable image from image (b).

Since we added one step of computing the reference image u_r, we compare the computational time of the Semi-Blind method with AM method in [23]. In Tables 2.1 and 2.2, it shows the savings in computational time by only performing one iterations of kernel and image functional separately. The computational cost of the shock filter is negligible compared to TV functional iterations, and kernel refinements take only a fraction of the computational time. The overall cost of computing the Semi-Blind method with the kernel refinements is roughly less than one-third that of AM method [23] with three loop iterations.

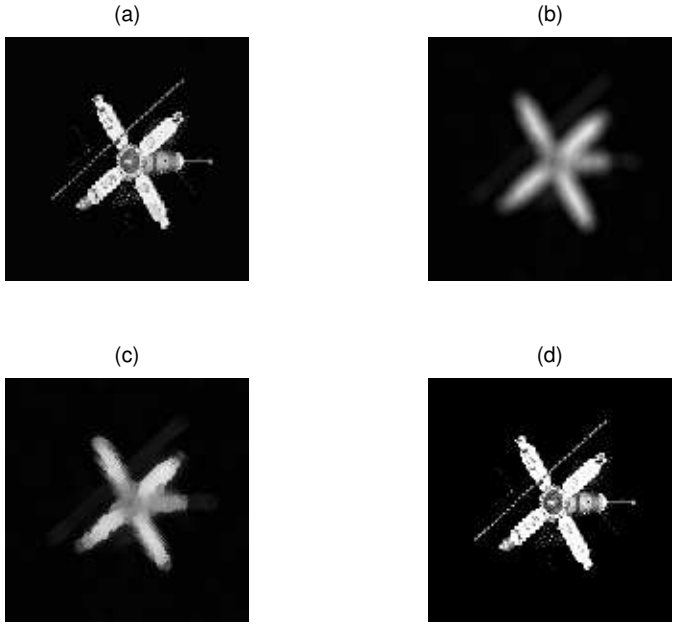

Figure 2.7: Non-black background image with L^2 fitting term. (a) The true image. (b) The given blurry original image. (c) Reference Image u_4 (d) Semi-Blind result (with kernel refinement). Image (d) recovers the sharp edges with details.

The next example shows the use of a non-centrosymmetric kernel in Figure 2.10. We can see with the addition of the non-symmetric kernel, the result is not as impressive as with the symmetric example, but there result is superior to existing blind methods. However, edges and fine details are still recovered in the image.

For our final example, let us consider a natural image. Figure 2.11 illustrates the use of the Semi-Blind Method on the natural image with an out of focus blur. We see that even in a more natural setting, the method computes an accurate result. We note here that even the shadow and fine details are recovered in the computed image.

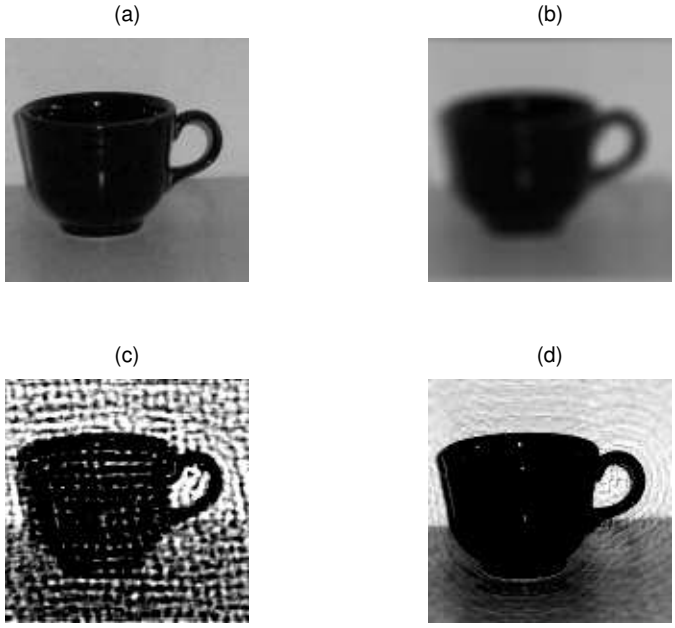

Figure 2.8: Non-black background image with L^2 fitting term. (a) The true image. (b) The given blurry original image. (c) Chan and Wong [23] result. (d) Semi-Blind result (with kernel refinement). Image (d) recovers the sharp edges with details such as the reflections on the cup.

Blind Deconvolution	**Step**	**Time(secs)**
Loop 1	Find kernel	299.71
	Find image	306.72
Loop 2	Find kernel	301.88
	Find image	303.63
Loop 3	Find kernel	312.71
	Find image	306.14
	Total:	1830.79

Table 2.1: Computational time for Chan and Wong [23] method.

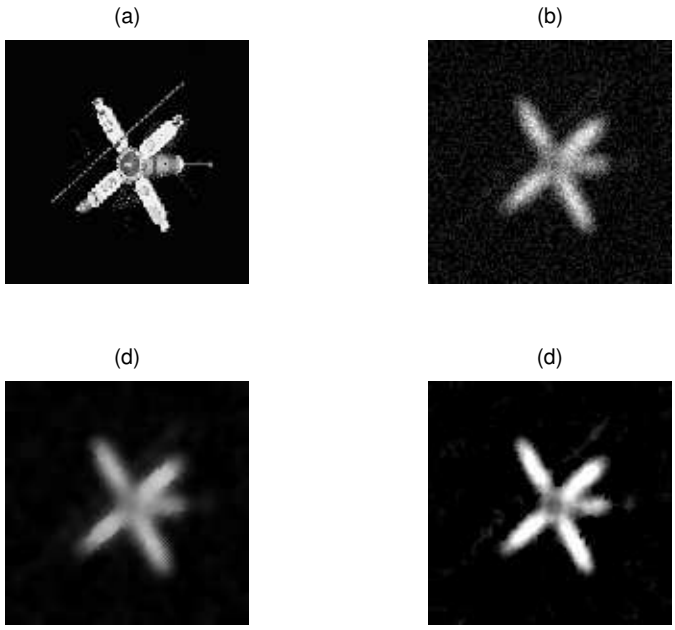

Figure 2.9: (a) The true image. (b) The given blurry and noisy image u_0. (c) Shock filtered image u_r. (d) Semi-Blind result shows reconstruction u.

Semi-Blind Method	**Time(secs.)**
Initial Shock filter	0.9
Find kernel	300.19
shock kernel	2.19
adaptively denoise kernel	10.57
Find image	181.76
Total:	495.61

Table 2.2: Computational time for Semi-Blind method with refinements.

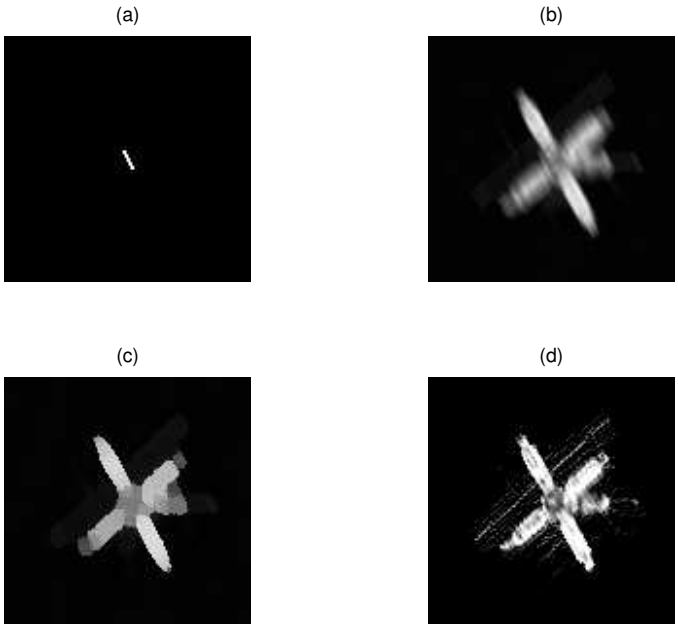

Figure 2.10: Figure (a) is the non-centrosymmetric kernel. Image (b) is the corresponding blurry image. Image (c) is the reference image. Image (d) is the computed image. Note that the output image is accurate despite the nonsymmetry in the kernel.

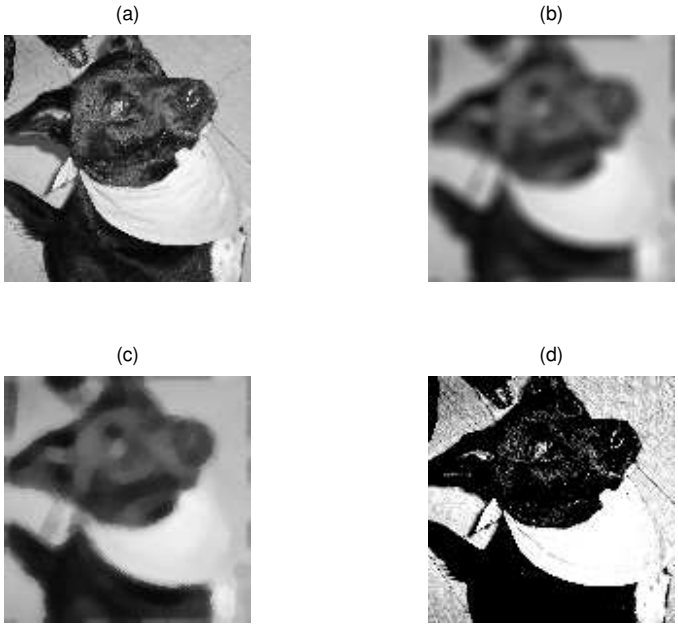

Figure 2.11: Figure (a) is the true image. Image (b) is the corresponding blurry image. Image (c) is the reference image. Image (d) is the computed image.

3 L^p Norm Fidelity in Semi-Blind Total Variation Deconvolution

3.1 L^p Fidelity Term Functionals

In this chapter, we extend the functional (2.1) from using the L^2 norm for the fidelity term to the L^p norm for $p \geq 1$. We consider the minimizing functional,

$$\min_{k,u} \frac{1}{p}\|k \star u - u_0\|^p_{L^p(\Omega)} + \lambda_1 \|k\|_{TV(\Omega)} + \lambda_2 \|u\|_{TV(\Omega)}. \tag{3.1}$$

When applying the Euler-Lagrange equations to (3.1), this is equivalent, as in the L^2 case, to considering the functionals

$$\min_{k} \frac{1}{p}\int_\Omega |k \star u - u_0|^p \, dx \, dy + \lambda_1 \int_\Omega |\nabla k| \, dx \, dy \tag{3.2}$$

and

$$\min_{u} \frac{1}{p}\int_\Omega |k \star u - u_0|^p \, dx \, dy + \lambda_2 \int_\Omega |\nabla u| \, dx \, dy. \tag{3.3}$$

As before in functional (2.6) we consider computing a reference image u_r instead of u in (3.2) and obtain

$$\min_{k} \frac{1}{p}\int_\Omega |k \star u_r - u_0|^p \, dx \, dy + \lambda_1 \int_\Omega |\nabla k| \, dx \, dy. \tag{3.4}$$

Now, applying the Euler-Lagrange equations to (3.4) and (3.3) we obtain

$$u_r(-x,-y) \star \left(|u_r(x,y) \star k - u_0|^{p-1} \frac{u_r(x,y) \star k - u_0}{|u_r(x,y) \star k - u_0|} \right) - \lambda_1 \nabla \cdot \left(\frac{\nabla k}{|\nabla k|} \right) = 0$$

and

$$k(-x,-y) \star \left(|k(x,y) \star u - u_0|^{p-1} \frac{k(x,y) \star u - u_0}{|k(x,y) \star u - u_0|} \right) - \lambda_2 \nabla \cdot \left(\frac{\nabla u}{|\nabla u|} \right) = 0.$$

In particular, we wish to consider $p = 1$ in this chapter, which is equivalent to solving the equations

$$u_r(-x,-y) \star \left(\frac{u_r(x,y) \star k - u_0}{|u_r(x,y) \star k - u_0|} \right) - \lambda_1 \nabla \cdot \left(\frac{\nabla k}{|\nabla k|} \right) = 0 \tag{3.5}$$

and

$$k(-x,-y) \star \left(\frac{k(x,y) \star u - u_0}{|k(x,y) \star u - u_0|} \right) - \lambda_2 \nabla \cdot \left(\frac{\nabla u}{|\nabla u|} \right) = 0. \tag{3.6}$$

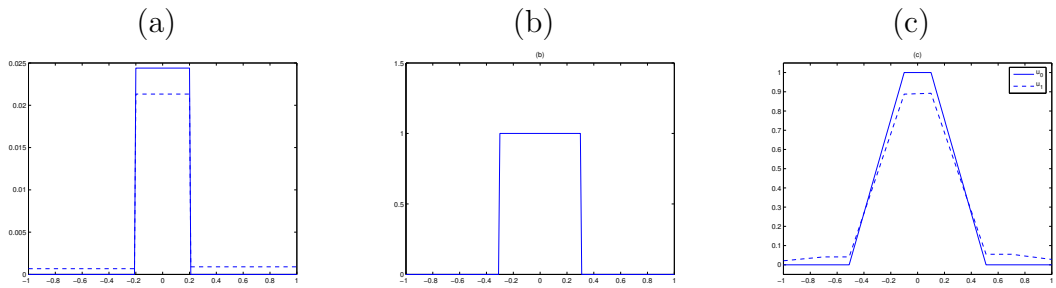

Figure 3.1: (a) The input kernel k (solid) and the output kernel $\tilde{k}$ (dotted) after applying equation (2.4) with a sufficiently large λ_1. (b) The true image u. (c) The given blurred image $u_0 = k \star u$ (solid) and the reconstructed image $u_1 = \tilde{k} \star u$ (dotted) with $\tilde{k}$. We compute the error between u_0 and u_1

3.2 Adaptive Scale Selection for L^2

In this section, we consider computing the optimal value for λ_1 for the kernel functional (2.4) given information on the scale of the kernel and object in the image. Strong and Chan completed a similar analysis for the denoising case in [73, 72].

We consider the one dimension case where a symmetric kernel k centered at the origin has radius r over the interval $(0, 1)$ with norm one. We also assume there is one object in the image u which has radius $\tilde{x} \in (0, 1)$. We can define u and normalized k by

$$u(x) = \begin{cases} 1 & -\tilde{x} \leq x \leq \tilde{x} \\ 0 & x < -\tilde{x} \text{ or } x > \tilde{x} \end{cases} \text{ and } k(x) = \begin{cases} 0 & x > r \text{ or } x < -r \\ \frac{1}{2r} & -r \leq x \leq r \end{cases}$$

We assume natural boundary conditions, i.e. the boundary value γ is extended as far as needed for the convolution operator. Between the kernel radius r and the image radius $\tilde{x}$, we assume that $\tilde{x} \geq r$ and $\tilde{x} - 1 \leq -\tilde{x} - r$. If these conditions are not satisfied, we extend the size of the image and rescale to $[-1, 1]$. Figure 3.1 illustrates the details of u, k, and $u_0 = k \star u$.

Assuming there is no noise, the given blurry image u_0 is defined as $u_0(x) = k * u(x) = \int_{-1}^{1} k(y)u(x - y)\, dy$. By minimizing the TV functional (2.4) for the kernel, the height of the kernel reduces while raising the base of the kernel to preserve the normality of the kernel (see [73, 72] for details on denoising case). In Figure 3.1 (a), we denote the shifts in the height of the kernel function as δ_1, δ_2, and δ_3, from left to right. We define the kernel $\tilde{k}$ to

be the result of applying (2.4) as

$$\tilde{k}(x) = \begin{cases} \delta_1, & x < -r \\ \frac{1}{2r} - \delta_2, & -r \leq x \leq r \\ \delta_3, & x > r \end{cases} . \tag{3.7}$$

We assume that these δ_is are chosen to normalize $\tilde{k}$. We denote the original image $u_0 = k \star u$ and define $u_1 = \tilde{k} \star u$ and need to compute the L^2 error in approximating k by $\tilde{k}$.

In order to derive results similar to [73], we first need to verify the graph in Figure 3.1 does indeed represent piecewise linear equations and compute each lines' equation in preparation for computing the L^2 error estimate.

If we examine Figure 3.1(c) we see that the convolved function has constant slope and in fact this slope depends on the constant pieces of the kernel. We can see this by considering the derivative of the convolution operation of a general kernel K in the following lemma.

Lemma 3.2.1 *For any kernel K we have that*

$$\frac{d}{dx}(K \star u)(x) = \begin{cases} K(x + \tilde{x}) & x < \tilde{x} - 1 \\ K(x + \tilde{x}) - K(x - \tilde{x}) & \tilde{x} - 1 \leq x \leq 1 - \tilde{x} \\ -K(x - \tilde{x}) & x > 1 - \tilde{x} \end{cases}$$

Proof. We start by writing

$$\begin{aligned} \tfrac{d}{dx}\left(K \star u(x)\right) &= \tfrac{d}{dx}\left(\textstyle\int_{-1}^{1} K(y)u(x-y)\,dy\right) \\ &= \textstyle\int_{-1}^{1} K(y)\tfrac{d}{dx}\left(u(x-y)\right)\,dy \end{aligned}$$

We note that we will in general have two jumps in u, at $-\tilde{x}$ and $\tilde{x}$. But, if $x < -1 + \tilde{x}$, we only see the left jump, since the radius of the kernel region is 1. Similarly, we only see the

right jump if $x > 1 - \tilde{x}$. This results in the derivative:

$$= \int_{-1}^{1} K(y) \begin{cases} \delta(x-y+\tilde{x}) & x < \tilde{x} - 1 \\ \delta(x-y+\tilde{x}) - \delta(x-y-\tilde{x}) & \tilde{x} - 1 \leq x \leq 1 - \tilde{x} \\ -\delta(x-y-\tilde{x}) & x > 1 - \tilde{x} \end{cases} dy$$

$$= \begin{cases} K(x+\tilde{x}) & x < \tilde{x} - 1 \\ K(x+\tilde{x}) - K(x-\tilde{x}) & \tilde{x} - 1 \leq x \leq 1 - \tilde{x} \\ -K(x-\tilde{x}) & x > 1 - \tilde{x} \end{cases}$$

∎

From this lemma, we can easily generalize to $K = k$ or $K = \tilde{k}$ and simplify the convolutions into the form we need for piecewise constant slopes in the following corollary.

Corollary 3.2.1 *We have that*

$$\frac{d}{dx} u_0(x) = \begin{cases} 0 & x < -\tilde{x} - r \\ \frac{1}{2r} & -\tilde{x} - r \leq x \leq -\tilde{x} + r \\ 0 & -\tilde{x} + r < x < \tilde{x} - r \\ -\frac{1}{2r} & \tilde{x} - r \leq x \leq \tilde{x} + r \\ 0 & x > \tilde{x} + r \end{cases}$$

and

$$\frac{d}{dx} u_1(x) = \begin{cases} \delta_1 & x < -1 + \tilde{x} \\ 0 & -1 + \tilde{x} \leq x \leq -\tilde{x} - r \\ \frac{1}{2r} - \delta_1 - \delta_2 & -\tilde{x} - r \leq x \leq -\tilde{x} + r \\ \delta_3 - \delta_1 & -\tilde{x} + r < x < \tilde{x} - r \\ -\frac{1}{2r} + \delta_2 + \delta_3 & \tilde{x} - r \leq x \leq \tilde{x} + r \\ 0 & \tilde{x} + r < x < 1 - \tilde{x} \\ -\delta_3 & x \geq 1 - \tilde{x} \end{cases}$$

Proof. We see that from the above lemma we get that

$$\frac{d}{dx}u_0(x) \quad = \begin{cases} k(x+\tilde{x}) & x < \tilde{x} - 1 \\ k(x+\tilde{x}) - k(x-\tilde{x}) & \tilde{x} - 1 \leq x \leq 1 - \tilde{x} \\ -k(x-\tilde{x}) & x > 1 - \tilde{x} \end{cases}$$

Now, we need to consider the boundary cases first. On the left hand side, we consider

$$k(x+\tilde{x}) \quad = \begin{cases} 0 & x < -\tilde{x} - r \\ \frac{1}{2r} & -\tilde{x} - r \leq x \leq -\tilde{x} + r \\ 0 & x >= -\tilde{x} + r \end{cases}$$

Noting that $\tilde{x} - 1 \leq -\tilde{x} - r$, we see that this is always 0. For the right end term we have

$$k(x-\tilde{x}) \quad = \begin{cases} 0 & x < \tilde{x} - r \\ \frac{1}{2r} & \tilde{x} - r \leq x \leq \tilde{x} + r \\ 0 & x >= \tilde{x} + r \end{cases}$$

Again we note that $\tilde{x} - 1 \leq -\tilde{x} - r$ or $1 - \tilde{x} \geq \tilde{x} + r$, so this term is always zero as well. Hence we have

$$\begin{aligned} \frac{d}{dx}u_0(x) \quad &= \begin{cases} 0 & x + \tilde{x} < -r \\ \frac{1}{2r} & -r \leq x + \tilde{x} \leq r \\ 0 & x + \tilde{x} > r \end{cases} - \begin{cases} 0 & x - \tilde{x} < -r \\ \frac{1}{2r} & -r \leq x - \tilde{x} \leq r \\ 0 & x - \tilde{x} > r \end{cases} \\ &= \begin{cases} 0 & x < -\tilde{x} - r \\ \frac{1}{2r} & -\tilde{x} - r \leq x \leq -\tilde{x} + r \\ 0 & -\tilde{x} + r < x < \tilde{x} - r \\ -\frac{1}{2r} & \tilde{x} - r \leq x \leq \tilde{x} + r \\ 0 & x > \tilde{x} + r \end{cases} \end{aligned}$$

Similarly we have

$$\frac{d}{dx}u_1(x) \quad = \begin{cases} \tilde{k}(x+\tilde{x}) & x < \tilde{x} - 1 \\ \tilde{k}(x+\tilde{x}) - \tilde{k}(x-\tilde{x}) & \tilde{x} - 1 \leq x \leq 1 - \tilde{x} \\ -\tilde{k}(x-\tilde{x}) & x > 1 - \tilde{x} \end{cases}$$

For the left hand and right hand sides we get that

$$\tilde{k}(x+\tilde{x}) = \delta_1 \tag{3.8}$$

$$\tilde{k}(x-\tilde{x}) = -\delta_3 \tag{3.9}$$

since $\tilde{x} - 1 \leq -\tilde{x} - r$ again. Now we must compute

$$\begin{aligned} \tilde{k}(x+\tilde{x}) - \tilde{k}(x-\tilde{x}) \quad &= \begin{cases} \delta_1 & x + \tilde{x} < -r \\ \frac{1}{2r} - \delta_2 & -r \leq x + \tilde{x} \leq r \\ \delta_3 & x + \tilde{x} > r \end{cases} - \begin{cases} \delta_1 & x - \tilde{x} < -r \\ \frac{1}{2r} - & -r \leq x - \tilde{x} \leq r \\ \delta_3 & x - \tilde{x} > r \end{cases} \\ &= \begin{cases} \delta_1 & x < -\tilde{x} - r \\ \frac{1}{2r} - \delta_2 & -\tilde{x} - r \leq \tilde{x} \leq -\tilde{x} + r \\ \delta_3 & x > -\tilde{x} + r \end{cases} - \begin{cases} \delta_1 & x < \tilde{x} - r \\ \frac{1}{2r} - \delta_2 & \tilde{x} - r \leq x \leq \tilde{x} + r \\ \delta_3 & x > \tilde{x} + r \end{cases} \\ &= \begin{cases} 0 & x < -\tilde{x} - r \\ \frac{1}{2r} - \delta_2 - \delta_1 & -\tilde{x} - r \leq x \leq -\tilde{x} + r \\ \delta_3 - \delta_1 & -\tilde{x} + r < x < \tilde{x} - r \\ \delta_3 + \delta_2 - \frac{1}{2r} & \tilde{x} - r \leq x \leq \tilde{x} + r \\ 0 & x > \tilde{x} + r \end{cases} \end{aligned}$$

Combining with (3.8) and (3.9) we get the desired result. ∎

We would now like to estimate $\|\tilde{k} \star u - u_0\|^2_{L^2((-1,1))}$, the L^2 error term in the function (2.4).In order to do this, we need to determine the equations for each of the parts of $\tilde{k} \star u = u_1$ and u_0.

Lemma 3.2.2 *We have that*

$$u_1(x) = \begin{cases} \delta_1(x+\tilde{x}+1) & x < -1+\tilde{x} \\ 2\delta_1\tilde{x} & -1+\tilde{x} \le x \le -\tilde{x}-r \\ \left(\frac{1}{2r}-\delta_1-\delta_2\right)(x+\tilde{x}+r)+2\delta_1\tilde{x} & -\tilde{x}-r < x < -\tilde{x}+r \\ 2\delta_2\tilde{x}+1-2\delta_1 r-2\delta_2 r+(\delta_3-\delta_1)(x+\tilde{x}-r) & -\tilde{x}+r \le x \le \tilde{x}-r \\ 1-2\delta_2 r+2\delta_3\tilde{x}-2\delta_3 r+\left(\delta 2+\delta 3-\frac{1}{2*r}\right)(x-\tilde{x}+r) & \tilde{x}-r < x < \tilde{x}+r \\ 2\delta_3\tilde{x} & \tilde{x}+r \le x \le 1-\tilde{x} \\ -\delta_3(x-1+\tilde{x})+2\delta_3\tilde{x} & x > 1-\tilde{x} \end{cases}$$

and

$$u_0(x) = \begin{cases} 0 & x \le -\tilde{x}-r \\ \frac{1}{2r}(x+\tilde{x}+r) & -\tilde{x}-r < x < -\tilde{x}+r \\ 1 & -\tilde{x}+r \le x \le \tilde{x}-r \\ -\frac{1}{2r}(x-\tilde{x}-r) & \tilde{x}-r < x < \tilde{x}+r \\ 0 & x \ge \tilde{x}+r \end{cases}$$

Proof. We have the slopes from above, we just need to compute the points where the slope changes and simplify the results. Also, we denote the first equation $y_1(x)$, the second $y_2(x)$, and so on. First, we need $u_1(-1)$ to start the computations,

$$\begin{aligned} u_1(-1) &= \int_{-1}^{1} \tilde{k}(y)u(-1-y)\,dy \\ &= \int_{-1}^{1} \begin{cases} \delta_1 & y < -r \\ \frac{1}{2r}-\delta_2 & -r \le y \le r \\ \delta_3 & y > r \end{cases} u(1+y)\,dy \end{aligned}$$

utilizing that u is an even function. Expanding out u we have

$$\begin{aligned}
&= \int_{-1}^{1} \left[\begin{cases} \delta_1 & y < -r \\ \frac{1}{2r} - \delta_2 & -r \le y \le r \\ \delta_3 & y > r \end{cases} \right] \left[\begin{cases} 0 & y < -1 - \tilde{x} \\ 1 & -1 - \tilde{x} \le y \le -1 + \tilde{x} \\ 0 & y > -1 + \tilde{x} \end{cases} \right] dy \\
&= \int_{-1}^{-1+\tilde{x}} \left[\begin{cases} \delta_1 & y < -r \\ \frac{1}{2r} - \delta_2 & -r \le y \le r \\ \delta_3 & y > r \end{cases} \right] dy \\
&= \delta_1(-1 + \tilde{x} + 1) = \delta_1 \tilde{x}
\end{aligned}$$

since we have $-1 + \tilde{x} \le -\tilde{x} - r < -r$. Now, we have from above

$$\begin{aligned}
y_1(x) &= u_1(-1) + \delta_1(x - (-\tilde{x} - 1) \\
&= \delta_1 \tilde{x} + \tilde{x}(x + 1) \\
&= \delta_1(x + \tilde{x} + 1)
\end{aligned}$$

We need a point for $y_2(x)$, hence, we plug in $y_1(-1 + \tilde{x})$:

$$\begin{aligned}
y_1(-1 + \tilde{x}) &= \delta_1(-1 + \tilde{x} + \tilde{x} + 1) \\
&= 2\delta_1 \tilde{x}
\end{aligned}$$

Thus we have

$$\begin{aligned}
y_2(x) &= y_1(-1 + \tilde{x}) + 0 \cdot (x - (-1 + \tilde{x})) \\
&= 2\delta_1 \tilde{x}
\end{aligned}$$

Since this constant, we already have the next starting point and get

$$\begin{aligned}
y_3(x) &= y_2(-\tilde{x} - r) + \left(\tfrac{1}{2r} - \delta_1 - \delta_2\right)(x - (-\tilde{x} - r)) \\
&= 2\delta_1 \tilde{x} + \left(\tfrac{1}{2r} - \delta_1 - \delta_2\right)(x + \tilde{x} + r))
\end{aligned}$$

Next, we compute

$$\begin{aligned}
y_3(-\tilde{x} + r) &= 2\delta_1 \tilde{x} + \left(\tfrac{1}{2r} - \delta_1 - \delta_2\right)(-\tilde{x} + r + \tilde{x} + r)) \\
&= 2\delta_1 \tilde{x} + 2r\left(\tfrac{1}{2r} - \delta_1 - \delta_2\right) \\
&= 2\delta_1 \tilde{x} + 1 - 2\delta_1 r - 2\delta_2 r
\end{aligned}$$

So now we get

$$\begin{aligned} y_4(x) &= y_3(-\tilde{x}+r) + (\delta_3 - \delta_1)\,(x - (-\tilde{x}+r)) \\ &= 2\delta_1\tilde{x} + 1 - 2\delta_1 r - 2\delta_2 r + (\delta_3 - \delta_1)\,(x + \tilde{x} - r) \end{aligned}$$

Now we compute for our next point,

$$\begin{aligned} y_4(\tilde{x}-r) &= 2\delta_1\tilde{x} + 1 - 2\delta_1 r - 2\delta_2 r + (\delta_3 - \delta_1)\,(\tilde{x} - r + \tilde{x} - r) \\ &= 2\delta_1\tilde{x} + 1 - 2\delta_1 r - 2\delta_2 r + 2\,(\delta_3 - \delta_1)\,(\tilde{x} - r) \\ &= 1 - 2\delta_2 r + 2\delta_3\tilde{x} - 2\delta_3 r \end{aligned}$$

Hence we have

$$\begin{aligned} y_5(x) &= y_4(\tilde{x}-r) + \left(\delta_2 + \delta_3 - \tfrac{1}{2r}\right)(x - (\tilde{x}-r)) \\ &= 1 - 2\delta_2 r + 2\delta_3\tilde{x} - 2\delta_3 r + \left(\delta_2 + \delta_3 - \tfrac{1}{2r}\right)(x - \tilde{x} + r) \end{aligned}$$

Next we compute

$$\begin{aligned} y_5(\tilde{x}+r) &= 1 - 2\delta_2 r + 2\delta_3\tilde{x} - 2\delta_3 r + \left(\delta_2 + \delta_3 - \tfrac{1}{2r}\right)(\tilde{x} + r - \tilde{x} + r) \\ &= 1 - 2\delta_2 r + 2\delta_3\tilde{x} - 2\delta_3 r + 2\left(\delta_2 + \delta_3 - \tfrac{1}{2r}\right) r \\ &= 2\delta_3\tilde{x} \end{aligned}$$

Which gives us

$$\begin{aligned} y_6(x) &= y_5(\tilde{x}+r) + 0 \cdot (x - (\tilde{x}+r) \\ &= 2\delta_3\tilde{x} \end{aligned}$$

And finally we compute

$$y_6(1-\tilde{x}) = 2\delta_3\tilde{x}$$

to get

$$\begin{aligned} y_7(x) &= y_6(1-\tilde{x}) + -\delta_3(x - (1-\tilde{x})) \\ &= 2\delta_3\tilde{x} - \delta_3(x + \tilde{x} - 1) \end{aligned}$$

Putting these into piecewise notation appropriately completes the proof. A similar analysis results in u_0. ∎

Now we can compute the term $\tilde{k} \star u - u_0$ for use in our final calculation for the L^2 error term.

Lemma 3.2.3 *We have that*

$$\tilde{k} \star u - u_0 = \begin{cases} \delta_1(x+\tilde{x}+1) & x < -1+\tilde{x} \\ 2\delta_1\tilde{x} & -1+\tilde{x} \le x \le -\tilde{x}-r \\ -\delta_2(x+\tilde{x}+r) - \delta_1(x+r) + \delta_1\tilde{x} & -\tilde{x}-r < x < -\tilde{x}+r \\ 2\delta_1\tilde{x} - 2\delta_1 r - 2\delta_2 r + (\delta_3-\delta_1)(x+\tilde{x}-r) & -\tilde{x}+r \le x \le \tilde{x}-r \\ (\delta_2+\delta_3)(x-r) + (\delta_3-\delta_2)\tilde{x} & \tilde{x}-r < x < \tilde{x}+r \\ 2\delta_3\tilde{x} & \tilde{x}+r \le x \le 1-\tilde{x} \\ -\delta_3(x-1-\tilde{x}) & x > 1-\tilde{x} \end{cases}$$

Proof. We note that

$$\begin{aligned} \tilde{k} \star u - u_0 &= u_1(x) - u_0(x) \\ &= \begin{cases} \delta_1(x+\tilde{x}+1) & x < -1+\tilde{x} \\ 2\delta_1\tilde{x} & -1+\tilde{x} \le x \le -\tilde{x}-r \\ \left(\frac{1}{2r} - \delta_1 - \delta_2\right)(x+\tilde{x}+r) + 2\delta_1\tilde{x} & -\tilde{x}-r < x < -\tilde{x}+r \\ 2\delta_2\tilde{x} + 1 - 2\delta_1 r - 2\delta_2 r & -\tilde{x}+r \le x \le \tilde{x}-r \\ \quad +(\delta_3-\delta_1)(x+\tilde{x}-r) & \\ 1 - 2\delta_2 r + 2\delta_3\tilde{x} - 2\delta_3 r & \tilde{x}-r < x < \tilde{x}+r \\ \quad +\left(\delta 2 + \delta 3 - \frac{1}{2*r}\right)(x-\tilde{x}+r) & \\ 2\delta_3\tilde{x} & \tilde{x}+r \le x \le 1-\tilde{x} \\ -\delta_3(x-1+\tilde{x}) + 2\delta_3\tilde{x} & x > 1-\tilde{x} \end{cases} \\ &\quad - \begin{cases} 0 & x \le -\tilde{x}-r \\ \frac{1}{2r}(x+\tilde{x}+r) & -\tilde{x}-r < x < -\tilde{x}+r \\ 1 & -\tilde{x}+r \le x \le \tilde{x}-r \\ -\frac{1}{2r}(x-\tilde{x}-r) & \tilde{x}-r < x < \tilde{x}+r \\ 0 & x \ge \tilde{x}+r \end{cases} \end{aligned}$$

$$= \begin{cases} \delta_1(x+\tilde{x}+1) & x < -1+\tilde{x} \\ 2\delta_1\tilde{x} & -1+\tilde{x} \le x \le -\tilde{x}-r \\ -(\delta_1+\delta_2)(x+\tilde{x}+r)+2\delta_1\tilde{x} & -\tilde{x}-r < x < -\tilde{x}+r \\ 2\delta_1\tilde{x}-2\delta_1 r-2\delta_2 r+(\delta_3-\delta_1)(x+\tilde{x}-r) & -\tilde{x}+r \le x \le \tilde{x}-r \\ 1-2\delta_2 r+2\delta_3\tilde{x}-2\delta_3 r & \tilde{x}-r < x < \tilde{x}+r \\ \quad +\left(\delta 2+\delta 3-\frac{1}{2*r}\right)(x-\tilde{x}+r)+\frac{1}{2r}(x-\tilde{x}-r) & \\ 2\delta_3\tilde{x} & \tilde{x}+r \le x \le 1-\tilde{x} \\ -\delta_3(x-1+\tilde{x})+2\delta_3\tilde{x} & x > 1-\tilde{x} \end{cases}$$

$$= \begin{cases} \delta_1(x+\tilde{x}+1) & x < -1+\tilde{x} \\ 2\delta_1\tilde{x} & -1+\tilde{x} \le x \le -\tilde{x}-r \\ -\delta_2(x+\tilde{x}+r)-\delta_1(x+r)+\delta_1\tilde{x} & -\tilde{x}-r < x < -\tilde{x}+r \\ 2\delta_1\tilde{x}-2\delta_1 r-2\delta_2 r+(\delta_3-\delta_1)(x+\tilde{x}-r) & -\tilde{x}+r \le x \le \tilde{x}-r \\ (\delta_2+\delta_3)(x-r)+(\delta_3-\delta_2)\tilde{x} & \tilde{x}-r < x < \tilde{x}+r \\ 2\delta_3\tilde{x} & \tilde{x}+r \le x \le 1-\tilde{x} \\ -\delta_3(x-1-\tilde{x}) & x > 1-\tilde{x} \end{cases} \qquad (*)$$

To see the fifth component in (*), we compute

$$\begin{aligned} 1-2\delta_2 r+2\delta_3 \quad & \tilde{x}-2\delta_3 r+\left(\delta 2+\delta 3-\tfrac{1}{2*r}\right)(x-\tilde{x}+r)+\tfrac{1}{2r}(x-\tilde{x}-r) \\ & = -2\delta+2r+2\delta_3\tilde{x}-2\delta_3 r+(\delta_2+\delta_3)(x-\tilde{x}+r) \\ & = (\delta_2+\delta_3)x+\delta_3\tilde{x}-\delta_2\tilde{x}-\delta_3 r-\delta_2 r \\ & = (\delta_2+\delta_3)x+(\delta_3-\delta_2)\tilde{x}-(\delta_2+\delta_3)r \\ & = (\delta_2+\delta_3)(x-r)+(\delta_3-\delta_2)\tilde{x} \end{aligned}$$

∎

We are finally at a point we can compute the exact answer for the fitting term, which is given below.

Theorem 3.2.1 *We have that*

$$\begin{aligned}\tfrac{1}{2}\|\tilde{k}\star u-u_0\|^2_{L^2(-1,1)} = \;& \tfrac{7}{6}\left(\delta_1^2+\delta_3^2\right)\tilde{x}^3+2(\delta_1^2+\delta_3^2)\tilde{x}^2(1-2\tilde{x}-r)\\ &+\tfrac{4r}{3}\left[(\delta_2+\delta_3)^2r^2+3\delta_3^2\tilde{x}^2-3\delta_3(\delta_2+\delta_3)\tilde{x}r\right]\\ &+\tfrac{4r}{3}\left[3\delta_1^2\tilde{x}^2-3\delta_1^2\tilde{x}r+\delta_1^2r^2+\delta_2^2r^2-3\delta_1\delta_2\tilde{x}r+2\delta_1\delta_2r^2\right]\\ &+\tfrac{2(\tilde{x}-r)}{3}\left[2(\delta_1^2+\delta_3^2)(\tilde{x}-r)^2+6\delta_2^2r^2-6\delta_2(\delta_1+\delta_3)r(\tilde{x}-r)\right.\\ &\left.\quad-\delta_1\delta_3(\tilde{x}-r)^2-6\delta_1\delta_3\tilde{x}r+3\delta_1\delta_3(\tilde{x}^2+r^2)\right]\end{aligned}$$

Proof. Let's separate the integrals into their respective regions:

$$\begin{aligned}\tfrac{1}{2}\textstyle\int_{-1}^{1}(\tilde{k}\star u-u_0)^2\,dx = \;& \tfrac{1}{2}\textstyle\int_{-1+\tilde{x}}^{-\tilde{x}-r}(\tilde{k}\star u-u_0)^2\,dx+\tfrac{1}{2}\textstyle\int_{-\tilde{x}-r}^{-\tilde{x}+r}(\tilde{k}\star u-u_0)^2\,dx\\ &+\tfrac{1}{2}\textstyle\int_{-\tilde{x}+r}^{\tilde{x}-r}(\tilde{k}\star u-u_0)^2\,dx+\tfrac{1}{2}\textstyle\int_{\tilde{x}-r}^{\tilde{x}+r}(\tilde{k}\star u-u_0)^2\,dx\\ &+\tfrac{1}{2}\textstyle\int_{\tilde{x}+r}^{1-\tilde{x}}(\tilde{k}\star u-u_0)^2\,dx+\tfrac{1}{2}\textstyle\int_{1-\tilde{x}}^{1}(\tilde{k}\star u-u_0)^2\,dx\\ = \;& A+B+C+D+E+F+G\end{aligned}$$

Then, we can evaluate each part and simplify the result. Starting with A we have

$$\begin{aligned}A &= \tfrac{1}{2}\textstyle\int_{-1}^{-1+\tilde{x}}(\tilde{k}\star u-u_0)^2\,dx\\ &= \tfrac{1}{2}\textstyle\int_{-1}^{-1+\tilde{x}}\left(\delta_1(x+\tilde{x}+1)\right)^2\,dx\\ &= \tfrac{\delta_1^2}{2}\textstyle\int_{-1}^{-1+\tilde{x}}\left(x+\tilde{x}+1\right)^2\,dx\\ &= \tfrac{\delta_1^2}{6}(x+\tilde{x}+1)^2\Big|_{x=-1}^{-1+\tilde{x}}\\ &= \tfrac{\delta_1^2}{6}\left[8\tilde{x}^2-\tilde{x}^2\right]\\ &= \tfrac{7\delta_1^2\tilde{x}^3}{6}\end{aligned}$$

For B we get

$$\begin{aligned}B &= \tfrac{1}{2}\textstyle\int_{-1+\tilde{x}}^{-\tilde{x}-r}(\tilde{k}\star u-u_0)^2\,dx\\ &= \tfrac{1}{2}\textstyle\int_{-1+\tilde{x}}^{-\tilde{x}-r}\left[2\delta_2\tilde{x}\right]^2\,dx\\ &= 2\delta_1^2\tilde{x}^2\;x|_{\tilde{x}-1}^{-\tilde{x}-r}\\ &= 2\delta_1^2\tilde{x}^2\left(\tilde{x}-1+\tilde{x}+r\right)\\ &= -4\delta_1^2\tilde{x}^3-2r\delta_1^2\tilde{x}^2+2\delta_1^2\tilde{x}^2\end{aligned}$$

For C we have

$$\begin{aligned}
C &= \tfrac{1}{2}\int_{-\tilde{x}-r}^{-\tilde{x}+r}\left[-\delta_2(x+\tilde{x}+r)-\delta_1(x-\tilde{x}+r)\right]\,dx \\
&= \int_{-\tilde{x}-r}^{-\tilde{x}+r}\tfrac{\delta_2^2}{2}(x+\tilde{x}+r)^2+\tfrac{\delta_1^2}{2}(x-\tilde{x}+r)^2+\delta_1\delta_2(x+\tilde{x}+r)(x-\tilde{x}+r)\,dx \\
&= \int_{-\tilde{x}-r}^{-\tilde{x}+r}\tfrac{\delta_2^2}{2}(x+\tilde{x}+r)^2+\tfrac{\delta_1^2}{2}(x-\tilde{x}+r)^2+\delta_1\delta_2\left[x^2+2rx+(-\tilde{x}^2+r^2)\right]\,dx \\
&= \tfrac{\delta_2^2}{6}(x+\tilde{x}+r)^3|_{-\tilde{x}-r}^{-\tilde{x}+r}+\tfrac{\delta_1^2}{6}(x-\tilde{x}+r)^3|_{-\tilde{x}-r}^{-\tilde{x}+r}+\delta_1\delta_2\left(\tfrac{x^3}{3}+rx^2+(-\tilde{x}^2+r^2)x\right)|_{-\tilde{x}-r}^{-\tilde{x}+r} \\
&= \tfrac{4\delta_2^2r^3}{3}+\tfrac{4\delta_1^2}{3}\left[3\tilde{x}^3r-3\tilde{x}r^2+r^3\right] \\
&\quad +\delta_1\delta_2\left[\tfrac{1}{3}\left((-\tilde{x}+r)^3-(-\tilde{x}-r)^3\right)\right. \\
&\quad\quad \left.+r\left((-\tilde{x}+r)^2-(-\tilde{x}-r)^2\right)+(-\tilde{x}^2+r^2)\left(-\tilde{x}+r+\tilde{x}+r\right)\right] \\
&= \tfrac{4}{3}\delta_2^2r^3+\tfrac{4}{3}\delta_1^2\left[3\tilde{x}^2r-3\tilde{x}r^2+r^3\right]+\delta_1\delta_2\left[2\tilde{x}^2r+\tfrac{2}{3}r^3-4\tilde{x}r^2-2\tilde{x}^2r+2r^3\right] \\
&= \tfrac{4}{3}\delta_2^2r^3+\tfrac{4}{3}\delta_1^2\left[3\tilde{x}^2r-3\tilde{x}r^2+r^3\right]+\delta_1\delta_2\left[-4\tilde{x}r^2+\tfrac{8}{3}r^3\right] \\
&= \tfrac{4r}{3}\left[3\delta_1^2\tilde{x}^2-3\delta_1^2\tilde{x}r+\delta_1^2r^2+\delta_2^2r^2-3\delta_1\delta_2\tilde{x}r+2\delta_1\delta_2r^2\right]
\end{aligned}$$

For D, let's first simplify the integrand. Let's denote the integrand by $\frac{1}{2}D'^2$. Then we have

$$\begin{aligned}
D' &= 2\delta_1\tilde{x}-2\delta_1r-2\delta_2r+(\delta_3-\delta_1)(x+\tilde{x}-r) \\
&= 2\delta_1\tilde{x}-2\delta_1r-2\delta_2r+\delta_3(x+\tilde{x}-r)-\delta_1x-\delta_1\tilde{x}-\delta_1r \\
&= \delta_1\tilde{x}-\delta_1r-2\delta_2r+\delta_3(x+\tilde{x}-r)-\delta_1x \\
&= \delta_1(x-\tilde{x}+r)-2\delta_2r+\delta_3(x+\tilde{x}-r)
\end{aligned}$$

Thus we have that

$$\begin{aligned}
D'^2 &= \delta_1^2(x-\tilde{x}+r)^2+\left[-2\delta_2r+\delta_3(x+\tilde{x}-r)\right]^2-2\delta_1(x-\tilde{x}+r)\left[-2\delta_2r+\delta_3(x+\tilde{x}-r)\right] \\
&= \delta_1^2(x-\tilde{x}+r)^2+4\delta_2^2r^2+\delta_3^2(x+\tilde{x}-r)^2-4\delta_2\delta_3r(x+\tilde{x}-r) \\
&\quad -2\delta_1(x-\tilde{x}+r)\left[-2\delta_2r+\delta_3(x+\tilde{x}-r)\right] \\
&= \delta_1^2(x-\tilde{x}+r)^2+4\delta_2^2r^2+\delta_3^2(x+\tilde{x}-r)^2-4\delta_2\delta_3r(x+\tilde{x}-r)+4\delta_1\delta_2r(x-\tilde{x}+r) \\
&\quad -2\delta_1\delta_3(x-\tilde{x}+r)(x+\tilde{x}-r)
\end{aligned}$$

Hence we have that

$$\begin{aligned}
\tfrac{1}{2}D'^2 &= \tfrac{\delta_1^2}{2}(x-\tilde{x}+r)^2+2\delta_2^2r^2+\tfrac{\delta_3^2}{2}(x+\tilde{x}-r)^2-2\delta_2\delta_3r(x+\tilde{x}-r)+2\delta_1\delta_2r(x-\tilde{x}+r) \\
&\quad -\delta_1\delta_3(x-\tilde{x}+r)(x+\tilde{x}-r) \\
&= \tfrac{\delta_1^2}{2}(x-\tilde{x}+r)^2+2\delta_2^2r^2+\tfrac{\delta_3^2}{2}(x+\tilde{x}-r)^2-2\delta_2\delta_3r(x+\tilde{x}-r)+2\delta_1\delta_2r(x-\tilde{x}+r) \\
&\quad -\delta_1\delta_3(x^2+2r\tilde{x}-(\tilde{x}^2+r^2))
\end{aligned}$$

Thus, we have for the integral D,

$$\begin{aligned}
D &= \int_{-\tilde{x}+r}^{\tilde{x}-r} \left[\tfrac{\delta_1^2}{2}(x-\tilde{x}+r)^2 + 2\delta_2^2 r^2 + \tfrac{\delta_3^2}{2}(x+\tilde{x}-r)^2 - 2\delta_2\delta_3 r(x+\tilde{x}-r) \right. \\
&\qquad \left. +2\delta_1\delta_2 r(x-\tilde{x}+r) - \delta_1\delta_3(x^2+2r\tilde{x}-(\tilde{x}^2+r^2)) \right] dx \\
&= \left\{ \tfrac{\delta_1^2}{6}(x-\tilde{x}+r)^3 + 2\delta_2^2 r^2 x + \tfrac{\delta_3^2}{6}(x+\tilde{x}-r)^3 - \delta_2\delta_3 r(x+\tilde{x}-r)^2 + \delta_1\delta_2 r(x-\tilde{x}+r)^2 \right. \\
&\qquad \left. -\delta_1\delta_3 \left[\tfrac{x^3}{3} + 2r\tilde{x}x - (\tilde{x}^2+r^2)x \right] \right\}_{x=-\tilde{x}+r}^{\tilde{x}-r} \\
&= \tfrac{-4\delta_1^2}{3}(r-\tilde{x})^3 + 3\delta_2^2 r^2(\tilde{x}-r) + \tfrac{4\delta_3^2}{3}(\tilde{x}-r)^2 - 4\delta_2\delta_3 r(\tilde{x}-r)^2 - 4\delta_1\delta_2 r(r-\tilde{x})^2 \\
&\qquad -\delta_1\delta_3 \left\{ \tfrac{(\tilde{x}-r)^3-(r-\tilde{x})^2}{3} + 4\tilde{x}r(\tilde{x}-r) - 2(\tilde{x}^2+r^2)(\tilde{x}-r) \right\} \\
&= \tfrac{4}{3}\left(\delta_1^2+\delta_3^2\right)(\tilde{x}-r)^3 + 4\delta_2^2 r^2(\tilde{x}-r) - 4\delta_2 r(\delta_1+\delta_3)(\tilde{x}-r)^2 - \delta_1\delta_3 \left\{ \tfrac{2(\tilde{x}-r)^3}{3} \right. \\
&\qquad \left. +4\tilde{x}r(\tilde{x}-r) - 2(\tilde{x}-r)^2(\tilde{x}+r) \right\} \\
&= \tfrac{4}{3}\left(\delta_1^2+\delta_3^2\right)(\tilde{x}-r)^3 + 4\delta_2^2 r^2(\tilde{x}-r) - 4\delta_2 r(\delta_1+\delta_3)(\tilde{x}-r)^2 - \tfrac{2}{3}\delta_1\delta_3(\tilde{x}-r)^3 \\
&\qquad -4\delta_1\delta_3\tilde{x}r(\tilde{x}-r) - 2\delta_1\delta_3(\tilde{x}-r)^2(\tilde{x}+r) \\
&= \tfrac{2(\tilde{x}-r)}{3}\left[2(\delta_1^2+\delta_3^2)(\tilde{x}-r)^2 + 6\delta_2^2 r^2 - 6\delta_2(\delta_1+\delta_3)r(\tilde{x}-r) \right. \\
&\qquad \left. -\delta_1\delta_3(\tilde{x}-r)^2 - 6\delta_1\delta_3\tilde{x}r + 3\delta_1\delta_3(\tilde{x}^2+r^2) \right]
\end{aligned}$$

For E we have that

$$\begin{aligned}
E &= \int_{\tilde{x}-r}^{\tilde{x}+r} \tfrac{1}{2}\left[(\delta_1+\delta_3)(x-\tilde{x}-r) + 2\delta_3\tilde{x} \right]^2 dx \\
&= \int_{\tilde{x}-r}^{\tilde{x}+r} \left[\tfrac{(\delta_2+\delta_3)^2}{2}(x-\tilde{x}-r)^2 + 2\delta_3^2\tilde{x}^2 + 2\delta_3\tilde{x}(\delta_2+\delta_3)(x-\tilde{x}-r) \right] dx \\
&= \left\{ \tfrac{(\delta_2+\delta_3)^2}{6}(x-\tilde{x}-r)^3 + 2\delta_3^2\tilde{x}^2 x + \delta_3\tilde{x}(\delta_2+\delta_3)(x-\tilde{x}-r)^2 \right\}_{x=\tilde{x}-r}^{\tilde{x}+r} \\
&= \tfrac{4(\delta_2+\delta_3)^2}{3}r^3 + 4\delta_3^2\tilde{x}^2 r - 4\delta_3\tilde{x}(\delta_2+\delta_3)r^2 \\
&= \tfrac{4r}{3}\left[(\delta_2+\delta_3)^2 r^2 + 3\delta_3^2\tilde{x}^2 - 3\delta_3(\delta_2+\delta_3)\tilde{x}r \right]
\end{aligned}$$

Then for F we have

$$\begin{aligned}
F &= \tfrac{1}{2}\int_{\tilde{x}+r}^{1-\tilde{x}} \left[2\delta_3\tilde{x} \right]^2 dx \\
&= 2\delta_3^2\tilde{x}^2 \left\{ x \right\}_{x=\tilde{x}+r}^{1-\tilde{x}} \\
&= 2\delta_3^2\tilde{x}^2(1-2\tilde{x}-r)
\end{aligned}$$

And finally, for the last part of the integral we have

$$\begin{aligned}
G &= \tfrac{1}{2}\int_{1-\tilde{x}}^{1} \left[-\delta_3(x-1-\tilde{x}) \right]^2 dx \\
&= \tfrac{\delta_3^2}{6}\left\{ (x-1-\tilde{x})^3 \right\}_{x=1-\tilde{x}}^{1} \\
&= \tfrac{7}{6}\delta_3^2\tilde{x}^3
\end{aligned}$$

To complete the proof, combine $A - G$ and reduce. ∎

We need the derivatives with respect to the δ_is for computing the minimum in the functional (2.4). The following corollary summarizes this result.

Corollary 3.2.2 *Define*

$$\begin{aligned} f(\delta_1,\delta_2,\delta_3) = \; & \tfrac{7}{6}\left(\delta_1^2+\delta_3^2\right)\tilde{x}^3 + 2(\delta_1^2+\delta_3^2)\tilde{x}^2(1-2\tilde{x}-r) \\ & + \tfrac{4r}{3}\left[(\delta_2+\delta_3)^2r^2 + 3\delta_3^2\tilde{x}^2 - 3\delta_3(\delta_2+\delta_3)\tilde{x}r\right] \\ & + \tfrac{4r}{3}\left[3\delta_1^2\tilde{x}^2 - 3\delta_1^2\tilde{x}r + \delta_1^2r^2 + \delta_2^2r^2 - 3\delta_1\delta_2\tilde{x}r + 2\delta_1\delta_2r^2\right] \\ & + \tfrac{2(\tilde{x}-r)}{3}\left[2(\delta_1^2+\delta_3^2)(\tilde{x}-r)^2 + 6\delta_2^2r^2\right. \\ & \qquad \left. -6\delta_2(\delta_1+\delta_3)r(\tilde{x}-r) - \delta_1\delta_3(\tilde{x}-r)^2 - 6\delta_1\delta_3\tilde{x}r + 3\delta_1\delta_3(\tilde{x}^2+r^2)\right] \end{aligned}$$

Then we have that

$$\begin{aligned} \tfrac{df}{d\delta_1} &= \delta_1\left[-3\tilde{x}^3 - 4\tilde{x}^2r + 4\tilde{x}^2\right] + \delta_2\left[-4\tilde{x}^2r + 4\tilde{x}r^2 - \tfrac{4}{3}r^3\right] + \delta_3\left[\tfrac{4}{3}\tilde{x}^3 - 4\tilde{x}^2r + 4\tilde{x}r^2 - \tfrac{4}{3}r^3\right] \\ \tfrac{df}{d\delta_2} &= \delta_1\left[-4\tilde{x}^2r + 4\tilde{x}r^2 - \tfrac{4}{3}r^3\right] + \delta_2\left[-\tfrac{8}{3}r^3 + 8\tilde{x}r^2\right] + \delta_3\left[-4\tilde{x}^2r + 4\tilde{x}r^2 - \tfrac{4}{3}r^3\right] \\ \tfrac{df}{d\delta_3} &= \delta_1\left[\tfrac{4}{3}\tilde{x}^3 - 4\tilde{x}^2r + 4\tilde{x}r^2 - \tfrac{4}{3}r^3\right] + \delta_2\left[-4\tilde{x}^2r + 4\tilde{x}r^2 - \tfrac{4}{3}r^3\right] + \delta_3\left[-3\tilde{x}^3 - 4\tilde{x}^2r + 4\tilde{x}^2\right] \end{aligned}$$

At this point, we can state the main result for the case $p = 2$ in the following theorem.

Theorem 3.2.2 *The (one-dimensional) solution $\tilde{k}$ to*

$$\min_k\{\frac{1}{2}\|k\star u - u_0\|^2_{L^2(-1,1)} + \lambda_1\int_{-1}^{1}|\nabla k|\,dx\}$$

is given by

$$\delta_1 = \delta_3 = \frac{9\lambda_1}{rp(\tilde{x},r)}, \qquad \delta_2 = \frac{3q(\tilde{x},r)\lambda_1}{4r^2p(\tilde{x},r)} \tag{3.10}$$

where

$$p(x,r) = -12r - 51x^2 + 36x + 5xr, \qquad q(x,r) = 12r + 5x - 12$$

and $\tilde{k}$ is given from (3.7). The parameter λ_1 is assumed to be chosen sufficiently large enough so that there is no shift in the kernel boundary regions.

Proof. The solution $\tilde{k}$ is a minimizer. Therefore, we find δ_is which minimizes this functional by considering the derivative of the functional in terms of each δ_is. The second term, $\lambda_1\int_\Omega|\nabla\tilde{k}|\,dx$ of the functional, can be represented as

$$\lambda_1\left[\left(\frac{1}{2r} - \delta_1 - \delta_2\right)H(x+r) + \left(\frac{1}{2r} - \delta_2 - \delta_3\right)H(x-r)\right]$$

where H is the Heaviside function. The first term $f = \|k \star u - u_0\|_{L^2(-1,1)}$is given from Theorem 3.2.1. Then, we solve

$$\begin{cases} \frac{df}{d\delta_1}(\delta_1, \delta_2, \delta_3) - \lambda_1 & = 0 \\ \frac{df}{d\delta_2}(\delta_1, \delta_2, \delta_3) - 2\lambda_1 & = 0 \\ \frac{df}{d\delta_3}(\delta_1, \delta_2, \delta_3) - \lambda_1 & = 0 \end{cases}$$

to get the equation for each δ_i. □ ∎

From this equation, notice that there is a dependence on the square of the radius of the kernel and the radius of the object in the image. This result reinforces the logic of λ_1 being proportional to the size of the kernel support as was deduced in [23]. In addition, since both image and kernel are symmetrical, the result is also symmetrical.

Note, to utilize this information to compute the kernel adaptively, we first take any $k^{(0)}$ and apply functional (2.1) with a large value for λ_1 and get the output kernel $k^{(1)}$. Then, we compute

$$\delta_{ij} = k^{(0)} - k^{(1)}.$$

Then, given $\tilde{x}$, the size of the object in the image, and r, the radius of the kernel support, we compute

$$\lambda_{1_{ij}} = \delta_{ij} \frac{4r^2 p(\tilde{x}, r)}{3q(\tilde{x}, r)}$$

using p and q from equation (3.10). The algorithm is summarized below in Algorithm 3.4.

Algorithm 3.4 Adaptive Scale Kernel Reconstruction

Require: u_0, the input image

Compute reference image u_r via the shock filter (2.9).
Set $k^{(0)} = \delta(x, y)$.
Compute $k^{(1)}$ by solving $u_r(-x, -y) \star (u_r(x, y) \star k^{(1)} - u_0) - \lambda_1 \nabla \cdot \left(\frac{\nabla k^{(1)}}{|\nabla k^{(1)}|} \right) = 0$ using a large value for λ_1.
Set $\delta_{ij} = k^{(0)} - k^{(1)}$.
Set $\lambda_{1_{ij}} = \delta_{ij} \frac{4r^2 p(\tilde{x}, r)}{3q(\tilde{x}, r)}$.
Compute $k^{(2)}$ by solving $u_r(-x, -y) \star (u_r(x, y) \star k^{(2)} - u_0) - \lambda_1 \nabla \cdot \left(\frac{\nabla k^{(2)}}{|\nabla k^{(2)}|} \right) = 0$ by using $\lambda_{1_{ij}}$.
Output $k^{(2)}$.

The following example illustrates using the adaptive scale selection for λ_1 in equation

(2.4) for the kernel. Figure 3.2 shows that adaptive scale selection gives the same result as manually choosing the optimal parameter. We applied the one dimensional analysis and used the known radius of (the out of focus) blur and used $\tilde{x}$ to be the average radius of the object (which is the size of the cup in the picture). We see that one dimensional analysis can be accurately extended to two dimensional images. Note here we do not pursue $p = 1$ analysis because in general this does not lead to improved results for the calculated image.

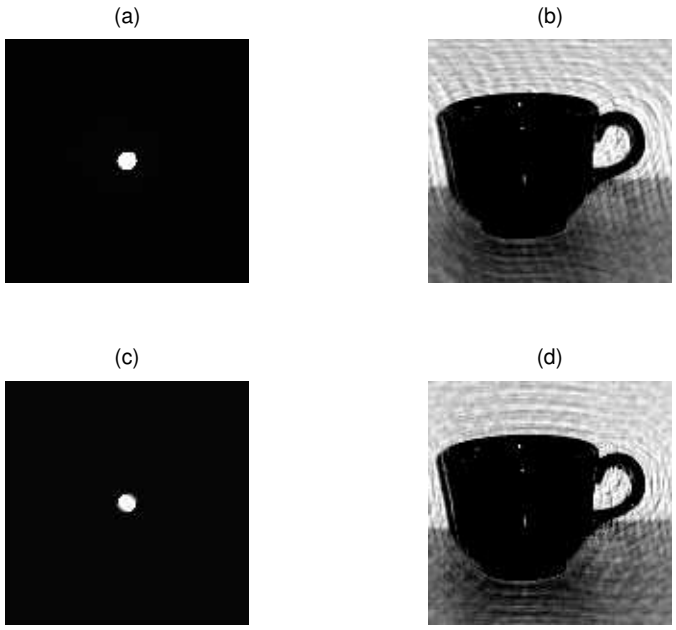

Figure 3.2: Adaptive scale parameter selection versus manual selection of λ_1. In image (a) and (b), λ_1 is manually chosen, while in image (c) and (d), λ_1 is chosen adaptively. Notice that the results for the adaptive λ_1 are as accurate as manual selection of λ_1.

3.3 Adaptive Scale Recognition for L^1

In this section, we derive the equivalent L^1 analysis for an adaptive scale method we derived in section 3.2. We note that all the calculations through Lemma 3.2.3 do not require any recomputation. First, we derive a new calculation for the fidelity term $\|\tilde{k} \star u - u_0\|_{L^1(-1,1)}$ in the following theorem.

Theorem 3.3.1 *We have that*

$$\begin{aligned}\|\tilde{k} \star u - u_0\|_{L^1(-1,1)} &= -\tfrac{1}{2}\tilde{x}\tfrac{1}{(\delta_1+\delta_2)(\delta_2+\delta_3)}\,(\delta_1^2\tilde{x}\delta_2 - 4\delta_3^2\delta_2 - 4\delta_3\delta_2^2 \\ &\quad -4\delta_3^2\delta_1 + \delta_3^2\tilde{x}\delta_2 + 9\delta_3\tilde{x}\delta_2^2 + \delta_3^2\tilde{x}\delta_1 - \\ &\quad 8\delta_2^3 r - 4\delta_1\delta_2^2 - 4\delta_1^2\delta_3 - 4\delta_1^2\delta_2 + 9\delta_1\tilde{x}\delta_2^2 + \delta_1^2\tilde{x}\delta_3 + \\ &\quad 4\delta_1^2 r\delta_2 + 4\delta_1^2 r\delta_3 - 4\delta_1\delta_2^2 r + 4\delta_3^2 r\delta_1 + 4\delta_3^2 r\delta_2 - 4\delta_2^2 r\delta_3 + \\ &\quad 18\delta_1\tilde{x}\delta_2\delta_3 - 8\delta_1\delta_2\delta_3)\end{aligned}$$

Proof. As in Theorem 3.2.2, we split the integral into seven parts:

$$\begin{aligned}\textstyle\int_{-1}^{1} |\tilde{k} \star u - u_0|\,dx &= \textstyle\int_{-1+\tilde{x}}^{-\tilde{x}-r} |\tilde{k} \star u - u_0|\,dx + \int_{-\tilde{x}-r}^{-\tilde{x}+r} |\tilde{k} \star u - u_0|\,dx + \int_{-\tilde{x}+r}^{\tilde{x}-r} |\tilde{k} \star u - u_0|\,dx \\ &\quad + \textstyle\int_{\tilde{x}-r}^{\tilde{x}+r} |\tilde{k} \star u - u_0|\,dx + \int_{\tilde{x}+r}^{1-\tilde{x}} |\tilde{k} \star u - u_0|\,dx + \int_{1-\tilde{x}}^{1} (\tilde{k} \star u - u_0|\,dx \\ &= A + B + C + D + E + F + G\end{aligned}$$

With the exceptions of C and E, the integration proceeds as before and by choosing the correct sign for the absolute value we get

$$\begin{aligned}A &= \tfrac{3}{2}\delta_1\tilde{x}^2 \\ B &= -2\delta_1\tilde{x}(-1 + 2 * x1 + r) \\ D &= -2\delta_1\tilde{x}^2 + 4\delta_1\tilde{x}r - 2\delta_1 r^2 + 4\delta_2 r\tilde{x} - 4\delta_2 r^2 - 2\delta_3\tilde{x}^2 + 4\delta_3\tilde{x}r - 2\delta_3 r^2 \\ F &= -2\delta 3\tilde{x}(-1 + 2\tilde{x} + r) \\ G &= \tfrac{3}{2}\delta_3\tilde{x}^2\end{aligned}$$

For C, we note that the lines u_1 and u_0 cross in the interval $[\tilde{x} - r, -\tilde{x} + r]$, so we must solve the equation

$$2\delta_1\tilde{x} + \left(\frac{1}{2r} - \delta_1 - \delta_2\right)(x + \tilde{x} + r) = \frac{1}{2r}\,(x - (-\tilde{x} - r))$$

which is

$$\hat{x} = -\frac{-\delta_1\tilde{x} + \delta_1 r + \delta_2\tilde{x} + \delta_2 r}{\delta_1 + \delta_2}$$

Thus, for C we have

$$\begin{aligned}C &= \textstyle\int_{-\tilde{x}+r}^{\hat{x}} \tilde{k} \star u - u_0\,dx + \int_{\hat{x}}^{-\tilde{x}+r} u_0 - \tilde{k} \star u\,dx \\ &= \tfrac{2\delta_1^2\tilde{x}^2}{\delta_1+\delta_2} + \tfrac{2(-\delta_1\tilde{x}+\delta_1 r+\delta_2 r)^2}{\delta_1+\delta_2} \\ &= \tfrac{2}{\delta_1+\delta_2)}\,(2\tilde{x}^2 * \delta_1^2 - 2\tilde{x}r\delta_1^2 - 2\tilde{x}r\delta_1\delta_2 + r^2\delta_1^2 + 2r^2\delta_1\delta_2 + r^2\delta_2^2)\end{aligned}$$

Similarly, for E, we find the point of intersection in $[\tilde{x}-r, \tilde{x}+r]$ by solving

$$1 - 2\delta_2 r + 2\delta_3\tilde{x} - 2\delta_3 r + \left(\delta_2 + \delta_3 - \frac{1}{2r}\right)(x - \tilde{x} + r) = -\frac{1}{2r}(x - (\tilde{x} - r)) + 1$$

and get the point

$$\overline{x} = \frac{\delta_2 r - \delta_3\tilde{x} + \delta_3 r + \delta_2\tilde{x}}{\delta_2 + \delta_3}$$

Hence, for E we get

$$\begin{aligned} E &= \textstyle\int_{\tilde{x}-r}^{\overline{x}} u_0 - \tilde{k} \star u \, dx + \int_{\overline{x}}^{\tilde{x}+r} \tilde{k} \star u - u_0 \, dx \\ &= \frac{2(-\delta_3\tilde{x} + \delta_2 r + \delta_3 r)^2}{\delta_2+\delta_3} + \frac{2\delta_3^2\tilde{x}^2}{\delta_2+\delta_3} \\ &= \frac{2}{\delta_2+\delta_3}(2\delta_3^2\tilde{x}^2 - 2\delta_2 r\delta_3\tilde{x} - 2\delta_3^2\tilde{x}r + r^2\delta_2^2 + 2\delta_2 r^2\delta_3 + \delta_3^2 r^2) \end{aligned}$$

Now by computing $A + B + C + D + E + F + G$, we get the desired result. ∎

We can define the function f again and computing the derivatives in preparation for our minimization via the following corollary.

Corollary 3.3.1 *Define*

$$\begin{aligned} f(\delta_1, \delta_2, \delta_3) &= -\tfrac{1}{2}\tilde{x}\tfrac{1}{(\delta_1+\delta_2)(\delta_2+\delta_3)}(\delta_1^2\tilde{x}\delta_2 - 4\delta_3^2\delta_2 - 4\delta_3\delta_2^2 \\ &\quad -4\delta_3^2\delta_1 + \delta_3^2\tilde{x}\delta_2 + 9\delta_3\tilde{x}\delta_2^2 + \delta_3^2\tilde{x}\delta_1 - \\ &\quad 8\delta_2^3 r - 4\delta_1\delta_2^2 - 4\delta_1^2\delta_3 - 4\delta_1^2\delta_2 + 9\delta_1\tilde{x}\delta_2^2 + \delta_1^2\tilde{x}\delta_3 + \\ &\quad 4\delta_1^2 r\delta_2 + 4\delta_1^2 r\delta_3 - 4\delta_1\delta_2^2 r + 4\delta_3^2 r\delta_1 + 4\delta_3^2 r\delta_2 - 4\delta_2^2 r\delta_3 + \\ &\quad 18\delta_1\tilde{x}\delta_2\delta_3 - 8\delta_1\delta_2\delta_3) \end{aligned}$$

Then we have that

$$\begin{aligned} \frac{df}{d\delta_1} &= -\frac{1}{2}\frac{(2\delta_1\tilde{x}\delta_2 + \delta_1^2\tilde{x} + 9\delta_2^2\tilde{x} - 4\delta_1^2 - 4\delta_2^2 - 8\delta_2\delta_1 + 4\delta_1^2 r + 4\delta_2^2 r + 8\delta_1 r\delta_2)\tilde{x}}{(\delta_1+\delta_2)^2} \\ \frac{df}{d\delta_2} &= \frac{4\tilde{x}}{(\delta_1+\delta_2)^2(\delta_2+\delta_3)^2}\left[(-\delta_1^2\tilde{x}\delta_2^2 + \delta_2^4 r - 2\delta_1\tilde{x}\delta_2\delta_3^2 - 2\delta_1^2\tilde{x}\delta_2\delta_3 + 2\delta_1^2 r\delta_2\delta_3 + 4\delta_1\delta_2^2 r\delta_3\right. \\ &\qquad \left. +2\delta_3^2 r\delta_1\delta_2 - 2\delta_1^2\tilde{x}\delta_3^2 - \delta_3^2\tilde{x}\delta_2^2 + \delta_1^2 r\delta_2^2 + 2\delta_1\delta_2^3 r + \delta 2^2 r\delta_3^2 + 2\delta 2^3 r\delta 3 + \delta_1^2 r\delta_3^2\right] \\ \frac{df}{d\delta_3} &= -\frac{1}{2}\frac{(2\delta_2\tilde{x}\delta_3 + 9\delta_2^2\tilde{x} + \delta_3^2\tilde{x} - 4\delta_2^2 - 4\delta_3^2 - 8\delta_2\delta_3 + 4\delta_2^2 r + 4\delta_3^2 r + 8\delta_2 r\delta_3)\tilde{x1}}{(\delta 2 + \delta 3)^2} \end{aligned}$$

Analysis for the λ_1 in the L^1 case can be completed as in the L^2 case (3.10). We use the same setting as in Figure 3.1 in subsection 3.2. By using similar proof as in Theorem 3.2.2, we get the following result for λ_1 for L^1 fitting term.

Theorem 3.3.2 *The solution* $\tilde{k}$ *to*

$$\min_k \{\frac{1}{2}\|k \star u - u_0\|_{L^1(-1,1)} + \lambda_1 \int_{-1}^{1} |\nabla k| dx\}$$

is given by

$$\delta_1 = \delta_3, \quad \delta_2 = -\frac{\delta_3(8r - 7\tilde{x} - 4)}{(8r + 9\tilde{x} - 4)}$$

where $\tilde{k}$ *is defined from equation (3.7). In addition,* λ_1 *is given by*

$$\lambda_1 = -\frac{1}{4} - \frac{81}{64}\tilde{x}^2 + \frac{9}{8}\tilde{x} - r^2 - \frac{1}{4}r\tilde{x} + r. \tag{3.11}$$

The parameter λ_1 *is assumed to be chosen sufficiently large enough so there is no shift in the kernel boundary regions.*

The symmetry is preserved for δ_1 and δ_3 as before in L^2 case. However, there is no dependence on the shift of the intensity value nor on the size of kernel, except for δ_2. In addition, the parameter λ_1 depends solely on the size of the kernel and the size of the object in the image. As a result, we can directly determine λ_1 with the knowledge of the image object size and the support size of the kernel. Hence, the extra step of computation to find the δ_i's are saved and we can get an adaptive form for determining the kernel as in subsection 3.2. The method is detailed in Algorithm 3.5.

Algorithm 3.5 Adaptive Scale Kernel Reconstruction for L^1

Require: u_0, the input image

Compute reference image u_r via $\begin{cases} u_t = -|\nabla u| sign(\mathcal{L}(u)) \\ u(x, y, 0) = u_0 \end{cases}$.

Set $\lambda_1 = -\frac{1}{4} - \frac{81}{64}\tilde{x}^2 + \frac{9}{8}\tilde{x} - r^2 - \frac{1}{4}r\tilde{x} + r$.

Compute k by solving $u_r(-x, -y) \star (u_r(x, y) \star k - u_0) - \lambda_1 \nabla \cdot \left(\frac{\nabla k}{|\nabla k|}\right) = 0$ by using λ_1.

Output k.

We note here while there is no pixel by pixel values, we can change the value of $\tilde{x}$ based on the position and emphasis larger and smaller objects in the image as needed.

3.4 Numerical Comparisons and Experiments

In this section, we consider various experiments using L^1 or L^2 fitting terms for functionals (3.4) and (3.3).

3.4.1 Numerical Implementation

For the numerical implementation of the L^1 fidelity term in Euler-Lagrange form for equations (3.5) and (3.6), we utilize an adaptive form of the Lagged Diffusivity Fixed Point method. We lagged both denominator terms in (3.5) to get an iteration on i of the form

$$u_r(-x,-y) \star \left(\frac{u_r(x,y) \star k^{(i)} - u_0}{|u_r(x,y) \star k^{(i-1)} - u_0|} \right) - \lambda_1 \nabla \cdot \left(\frac{\nabla k^i}{|\nabla k^{(i-1)}|} \right) = 0$$

for $i = 1, 2, \ldots$. The method converges similarly the Lagged Diffusivity Fixed Point method and only requires ten iterations to converge. Similarly, for equation (3.6), we iterate on i for

$$k(-x,-y) \star \left(\frac{k(x,y) \star u^{(i)} - u_0}{|k(x,y) \star u^{(i-1)} - u_0|} \right) - \lambda_2 \nabla \cdot \left(\frac{\nabla u^i}{|\nabla u^{(i-1)}|} \right) = 0$$

for $i = 1, 2, \ldots$ and the method has similar convergence properties.

Additionally, one can consider solving iterate $u^{(n+1)}$ into terms of $\Delta u^{(n+1)} = u^{(n+1)} - u^{(n)}$by solving

$$\left(K^T K - \lambda_2 L\right) \Delta u = -K^T (Ku - u_0) + \lambda_2 L u \tag{3.12}$$

for K being the lagged convolution matrix for kernel k and operator L being the lagged diffusivity term. In Figure 3.3 we see the comparison. In image (a), the solution was computed using Δu, which in image (b), the result was computed directly for u. We can see it is advantageous to solve directly in this case. As a result, we directly solve for the equation (3.6) in our calculations.

The Semi-Blind method with kernel refinements is a straightforward implementation with the new equations. The L^1 version of the method is summarized below in Algorithm 3.6.

3.4.2 Scale Dependence in the L^1 Fidelity Term

In this subsection, we consider the scale dependence we derived in section 3.3. Using equation (3.11), we can see there is a clear dependence on the size of the kernel support region as well as the size of the object in the image. A similar analysis works for image reconstruction and deblurring and the results are similar. Hence, we should be able to see some dependence on the change in λ_2 with respect to the clarity achieved in the output image.

Using L^1, there is a property of scale, as was seen in the work by [13] for the denoising case and this can be also observed for deblurring cases. In Figure 3.4, we see a set of different scaled boxes in the image. In (a) all the boxes have sharp edges still, (b) shows the next

Figure 3.3: (a) Image result after solving via equation (3.12). (b) Image result after a direct solve for u. Note that the direct solution of u has cleaner output.

jump in the residual and only the smaller boxes loosing clarity at edges. In (c) we see a similar result, but with a choice of λ_2 larger. Finally, in (d) we see after the next increase in λ_2, the large boxes are losing distinct edge features and become blurry. In Figure 3.5(a) we see the region where the λ_2 values are changing in Figure 3.4(a)-(d).

In Figure 3.5(c), we see the plot of the value for λ_1 versus the residual norm $\|k \star u - u_0\|_{L^1}$ showing where these jumps in the residual have occurred.

As with the denoising case[13], we see the L^2 version of the graphs, do not have the same sharp jumps. In figure 3.5 we see in (b) the L^1 graph and in (c) the corresponding region in the L^2 recovery of the image. You can see the L^2 version of the solution results in a smooth graph with no sharp jumps in (c).

Algorithm 3.6 L^1 Semi-Blind Method with Kernel Refinements

Require: u_0, the input image

Compute reference image u_r via $\begin{cases} u_t = -|\nabla u| sign(\mathcal{L}(u)) \\ u(x,y,0) = u_0 \end{cases}$.

Solve $u_r(-x,-y) \star \frac{u_r(x,y)\star k^{(1)} - u_0}{|u_r(x,y)\star k^{(1)} - u_0|} - \lambda_1 \nabla \cdot \left(\frac{\nabla k^{(1)}}{|\nabla k^{(1)}|} \right) = 0$ for k.

Apply the shock filter on $k^{(1)}$ to get $k^{(2)}$.

Compute $|D|$ and $|\partial D|$.

Denoise $k^{(2)}$ with small λ to get $k^{(3)}$.

Compute $\delta_{ij} = k^{(2)} - k^{(3)}$.

Set $\lambda_{ij} = \delta_{ij} * \frac{|D|}{|\partial D|}$.

Denoise with λ_{ij} to get $k^{(4)}$.

Solve $k^{(4)}(-x,-y) \star \frac{k^{(4)}(x,y)\star u - u_0}{|k^{(4)}(x,y)\star u - u_0|} - \lambda_2 \nabla \cdot \left(\frac{\nabla u}{|\nabla u|} \right) = 0$ for u.

Output image u.

3.4.3 Fitting Term Comparisons

We experiment with the original AM method [23] with different combinations of p for two coupled equations.

$$\begin{matrix} \min\limits_{k} \dfrac{1}{p_1} \displaystyle\int_\Omega |k \star u - u_0|^{p_1} \, dx \, dy + \lambda_1 \int_\Omega |\nabla k| \, dx \, dy \\ \min\limits_{u} \dfrac{1}{p_2} \displaystyle\int_\Omega |k \star u - u_0|^{p_2} \, dx \, dy + \lambda_2 \int_\Omega |\nabla u| \, dx \, dy \end{matrix} . \tag{3.13}$$

Figure 3.6 shows results of using different combinations of p_i values, either 1 or 2. Note that for this experiment, we choose λ_i to emphasize the effect of details while sacrificing smoothness of the final image to better illustrate the effects of using different p_is. Comparing p_2 values, which is the fitting terms for the image functional in equation (3.13), the first column images with L^1 fitting ($p_2 = 1$) gives better details with less noisy artifacts compared to the second column images with L^2 fitting. This result is consistent with the fact that in general L^1 fitting is better for recovering details of images. Comparing different p_1 values for the kernel functional, the second row using $p_1 = 2$ gives better results in the final image. This choice will depend on the kind of true kernel used in generating u_0. Here we used out of focus blur for the experiments, and L^2 fitting seems to recover kernel much closer to true kernel function. In this case, we conclude that the best choice is image (c), using $p_1 = 2$ and $p_2 = 1$ for AM method: L^2 fitting for kernel functional and L^1 fitting for image functional.

We apply the same comparison for Semi-Blind method in Figure 3.7. When we compare

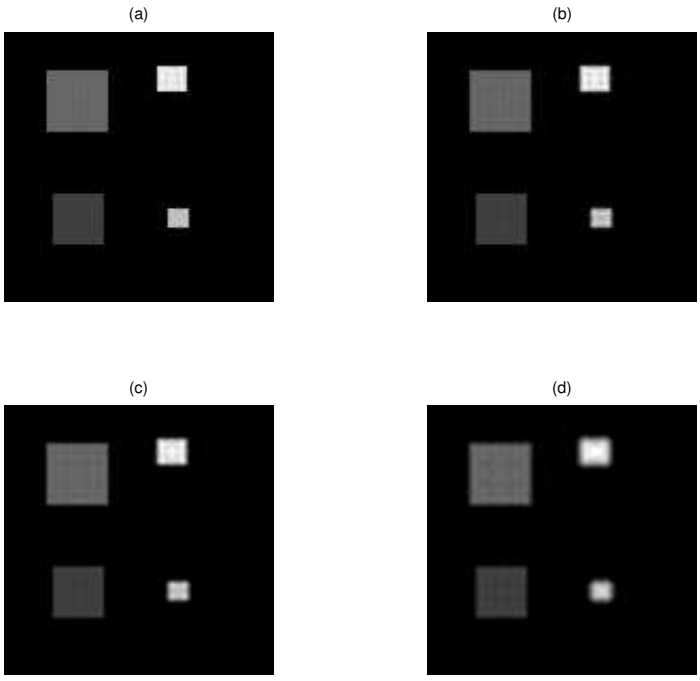

Figure 3.4: Comparison of image recovery using various values of λ_1. (a) $\lambda_1 = 3.2 \times 10^{-5}$: Image before first jump (b) $\lambda_1 = 5.6 \times 10^{-5}$: After first jump, edges on the small boxes are lost (c) $\lambda_1 = 9.0 \times 10^{-5}$: Show before the next jump (d)$\lambda_1 = 2.6 \times 10^{-4}$: Shows after the second jump, where large boxes loose edge detail.

the columns, we see that $p_2 = 1$ (L^1 fitting for image functional (3.3)), the first column, produces better results. By comparing the rows, we see that the second row (L^2 fitting term for kernel functional, $p_1 = 2$) results in better images. This is similar to analysis in AM method settings. In [35], the authors presented a similar analysis using a known kernel function, which is assumed to be Gaussian. Their results show that the L^1 norm fitting is particularly well suited for images and is not affected by outlying data. This is consistent with our results for both the blind deconvolution AM method [23]) as well as Semi-Blind methods for the image functional (3.3).

3.4.4 Examples

Our first example in Figure 3.8 uses the L^2 kernel result with the L^1 image result($p_1 = 2, p_2 = 1$) with a noisy and blurry image from Figure 2.9(b). The result you see is similar.

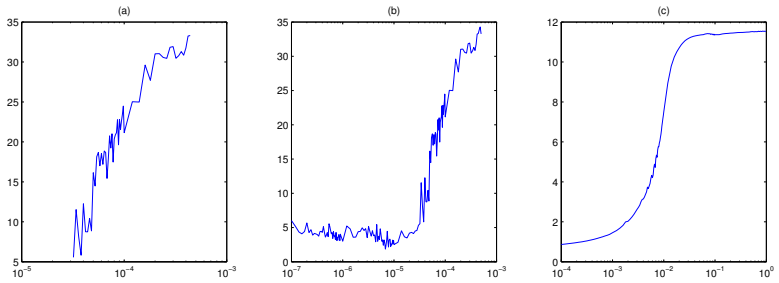

Figure 3.5: (a) The graph of λ_1 versus L^1 norm for the affect image in figure 3.4. (b) The graph of λ_1 versus the L^1 residual norm (c) The graph of λ_1 versus the L^2 residual norm. Note that the L^2 graph is smoother than the L^1 graph.

However, extra smoothing had to be applied as in the L^2 case in Figure 2.9 so the result is not as visually pleasing as we would desire.

The next example is using a direction kernel function, the same one that is used in Figure 2.10. However in Figure 3.9, we see that the results are superior, due to using the L^1 norm for the kernel and image recovery. The reason the L^1 norm is used for the kernel, is since the kernel has such a small region, the computations results in a closer approximate than using the L^2 norm. In addition, we have already shown that the L^1 calculations for image results in cleaner output and this is justifies this further for $p = 2$.

The final example is a natural image, using the L^1 norm for images but L^2 for the kernel and is shown in Figure 3.10. We can see the result appears to be slightly better for this L^1 approximation. If we compare them side-by-side, we can see this is in fact true in Figure 3.11. We see in Figure 3.11(a), the output image from using the L^2 norm, while image (b) is from the using the L^1 norm. We note that we can see less "ringing" effects in the L^1 version of the image which further reinforces that the L^1 image reconstruction is superior to the L^2 reconstructed image.

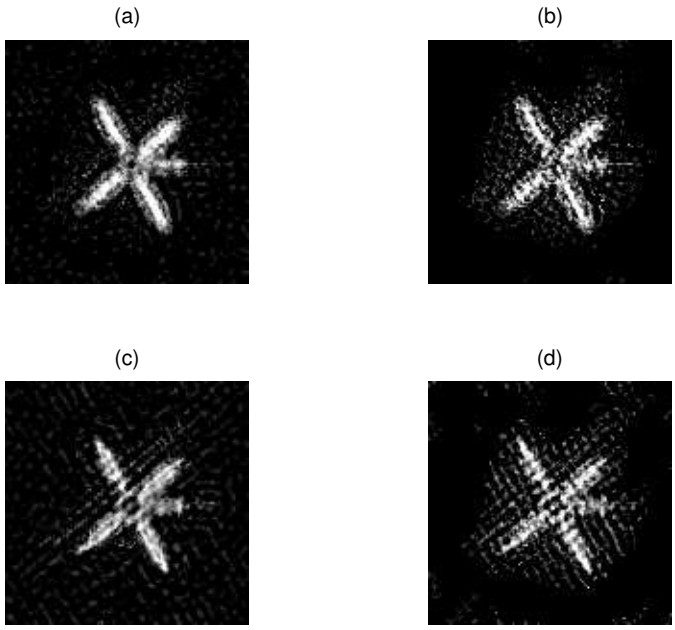

Figure 3.6: Comparison on using different p_1 and p_2 in AM method (3.1) setting. Note that we exaggerated the results by choosing λ_1 to emphasize the details in the image while sacrificing smoothness of the final image. (a) L^1 fitting for both kernel and image functional ($p_1 = p_2 = 1$). (b) L^1 fitting for kernel and L^2 fitting for image ($p_1 = 1$,$p_2 = 2$). (c) L^2 fitting for kernel and L^1 fitting for image ($p_1 = 2$,$p_2 = 1$). (d) The original AM method [23] with $p_1 = p_2 = 2$. The first column images with $p_2 = 1$ have more details recovered, and the second row images with $p_1 = 2$ has clearer results.

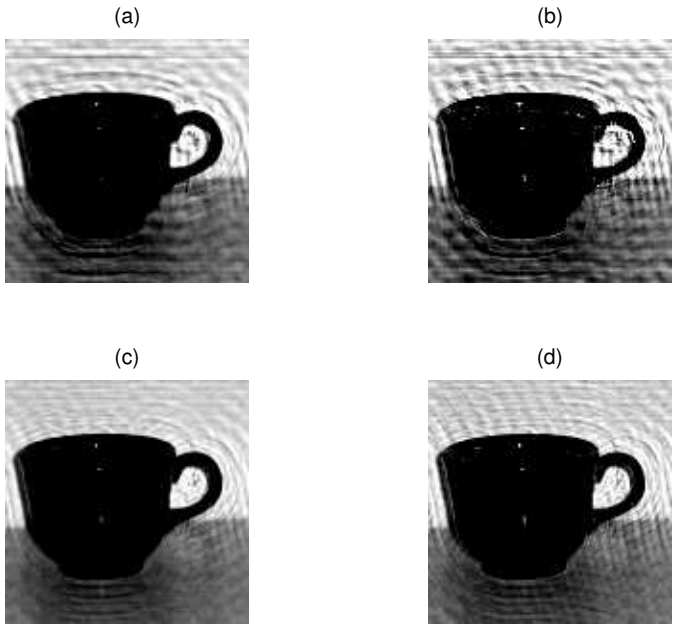

Figure 3.7: Comparison on using different p_1 and p_2 for Semi-Blind method. (a) L^1 fitting for both kernel and image functional ($p_1 = p_2 = 1$). (b) L^1 fitting for kernel and L^2 fitting for image ($p_1 = 1$,$p_2 = 2$). (c) L^2 fitting for kernel and L^1 fitting for image ($p_1 = 2$,$p_2 = 1$). (d) L^2 fitting for both kernel and image functional ($p_1 = p_2 = 2$). The first column images with $p_2 = 1$ have less "ringing" effects, and the second row images with $p_1 = 2$ has clearer results.

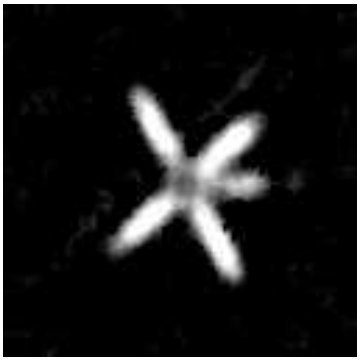

Figure 3.8: Results from the L^1 recovery of the noisy image. Note the extra smoothing that had to be applied.

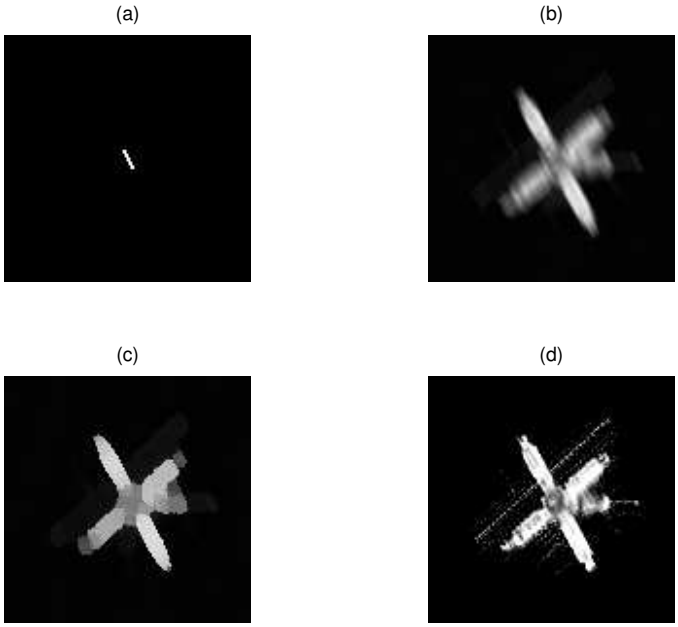

Figure 3.9: Figure (a) is the non-centrosymmetric kernel. Image (b) is the corresponding blurry image. Image (c) is the reference image. Image (d) is the computed image. Note that the output image is accurate and improved over Figure 2.10(d).

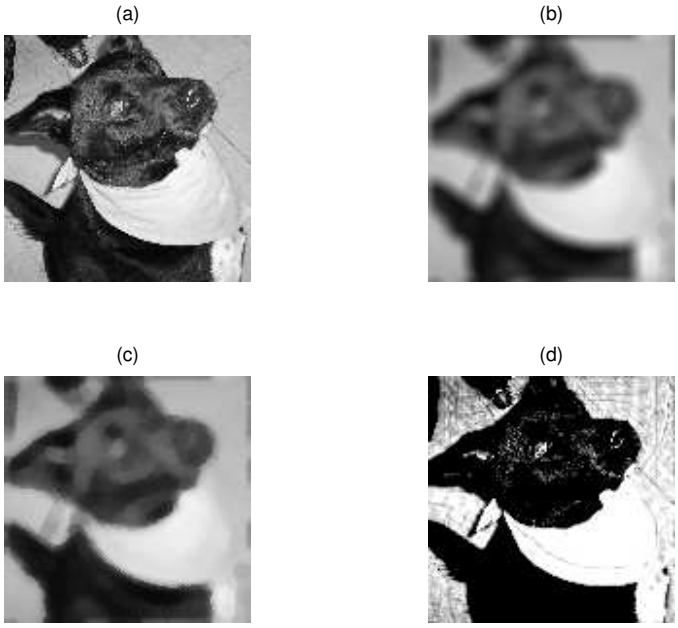

Figure 3.10: Figure (a) is the true image. Image (b) is the corresponding blurry image. Image (c) is the reference image. Image (d) is the computed image.

Figure 3.11: Image (a) is the L^2 norm image using the correct kernel. Image (b) is the same output using the L^1 norm. We can see the L^1 norm is clearly better for natural images since there is less "ringing" effects.

4 Discretized Picard's Method

4.1 Modified Picard Method for PDEs

In the PDE version of Picard's Method[70], one considers

$$\begin{cases} u_t & = P(u, \frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}, \ldots, \frac{\partial^2 u}{\partial x^2}, \frac{\partial^2 u}{\partial x \partial y}, \ldots) \\ u(\cdot, 0) & = q(\cdot) \end{cases}$$

where P and q are n variable polynomials. Parker and Sochacki's method is to compute the iterates

$$\begin{cases} \phi_0(t) & = q(\cdot) \\ \phi_{n+1}(\cdot, t) & = q(\cdot) + \int_0^t P(\phi_n(\cdot, s))\, ds, \quad n = 0, 1, 2, \ldots \end{cases}$$

We truncate the terms with t-degree higher than n at each step since these terms do not contribute to the coefficient for the t^{n+1} term in the next iteration. We denote the *degree of the Picard iterate* as j for $\phi^{(j)}(t)$, given this truncation that is performed. This method is summarized below in Algorithm 4.7.

Algorithm 4.7 Modified Picard Method for PDEs

Require: q, the initial condition, and P the polynomial system
Require: Δt and *numtimesteps*
Require: *degree* the degree of the Picard approximation
  for i from 1 to *numtimesteps* **do**
    $\phi_0(\cdot, t) = q(\cdot)$
    for j from 1 to degree **do**
      $\phi_j(\cdots, t) = q(\cdot) + \int_0^t P(\phi_{j-1}(\cdot, s))\, ds$
      Truncate $\phi_j(\cdot, t)$ to degree j in t.
    end for
    $q(\cdot) = \phi_{degree}(\cdot, \Delta t)$
  end for

This algorithm is called the Modified Picard Method(MPM). While the MPM algorithm easily computes the approximates since it only depends on calculating derivatives and integrals of the underlying polynomials, it has some limitations. In [70], the authors showed how to handle the PDE including the initial conditions. However, the method requires the initial conditions in polynomial form. While in some PDEs this is the case, many times one computes a Taylor polynomial that approximates the initial condition to high degree. This

results in a substantial increase in computational time. For some problems, the initial condition is not explicitly known, but only a digitized form of the data. For example, in image processing, most of the data has already been digitized and we have to interpolate the data using polynomials in order to apply the MPM. If this is done, the resulting polynomial may not effectively approximate the derivatives of the original function. The polynomial approximation might contain large amounts of oscillations that does not represent the underlying data accurately. Finally, we would also like to be able to handle boundary conditions in a simple manner, but keep the extensibility of the MPM, which does not allow for a boundary condition.

4.2 Discretized Picard's Method

To overcome the deficiencies listed in section 4.1, we consider the underlying discrete data directly. We consider the initial condition $u_0 = u_{0_{i_1 i_2 \dots i_m}}$ where $u_0 \in \Re^{n_1 \times n_2 \times \cdots \times n_m}$ is a matrix of m dimensions. Instead of applying the derivatives directly, we consider a set of linear operators L_i where $i = 1, 2, \dots k$ that approximate the derivatives. Then, instead of solving the PDE

$$\begin{cases} u_t & = P(u, \frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}, \dots, \frac{\partial^2 u}{\partial x^2}, \frac{\partial^2 u}{\partial x \partial y}, \dots) \\ u(\cdot, 0) & = q(\cdot) \end{cases}$$

we consider using the L_i to approximate the various derivatives and solving this PDE by approximation by

$$\begin{cases} u_t & = P(u, L_1 u, L_2 u, \dots, L_k u) \\ u(\cdot, 0) & = u_{0_{i_1 i_2 \dots i_m}} \end{cases}$$

We define multiplication of two elements u and v component-wise, instead of the using standard matrix multiplication. Then, we compute the iterates

$$\begin{cases} \phi_0(t) & = u_0 \\ \phi_{n+1}(t) & = u_0 + \int_0^t P(\phi_n(s), L_1\phi_n(s), L_2\phi_n(s), \dots, L_k\phi_n(s))\, ds, \quad n = 0, 1, 2, \dots \end{cases}$$

The resulting method computes the discretized solution of the PDE, but is continuous in the time variable. In section 4.3, we illustrate the importance of requiring the operators L_i to be linear in order to get a similar result to the MPM. Given we are utilizing the underlying discrete data in the space variables, we call this new method the **Discretized Picard Method**(DPM). The new method is listed in Algorithm 4.8.

Algorithm 4.8 Discretized Picard Method

Require: u_0, the initial condition, and P the polynomial system
Require: $L_1, L_2, \ldots, L_k$, the linear approximations to the derivatives
Require: Δt and *numtimesteps*
Require: *degree* the degree of the Picard approximation
  for i from 1 to *numtimesteps* **do**
    $\phi_0(\cdot, t) = u_0$
    for j from 1 to degree **do**
      $\phi_j(t) = u_0 + \int_0^t P(\phi_{j-1}(s), L_1(\phi_{j-1}(s), \ldots, L_k(\phi_{j-1}(s))\, ds$
    end for
    $u_0 = \phi_{degree}(\Delta t)$
    Enforce boundary conditions on u_0.
  end for

4.2.1 Computation of L_i

For the linear operator, there are many discrete operators available for L_i[see [62, 52]]. For example, one could use finite differences, finite elements, or Galerkin methods. In this chapter, the operator chosen is the finite difference(FD) operator. For example, if $u_t = u_{xx}$, we can choose the operator L to satisfy central difference scheme

$$Lu_j = \frac{u_{j+1} - 2u_j + u_{j-1}}{\Delta x}.$$

The operator L is extended easily to the two and three dimension case. In section 4.4, we show how the choice of the operator determines the stability condition for the maximum timestep size. In addition, the first and last terms in the one dimension case, and all the boundary terms in the two and three dimension cases will have to be handled separately. We discuss this further in section 4.2.2.

Recall, from section 1.2.1, that a PDE $u_t = f(u, \frac{\partial u}{\partial x}, \ldots)$, is considered projectively polynomial if it can be rewritten as a system of equations in n-variables so that $Y' = P(Y, \frac{\partial Y_1}{\partial x}, \ldots)$ where $Y = [Y_1, \ldots, Y_N]$ and P is polynomial.

For a general class of linear operators based on a linear finite difference(FD) scheme, we deduce that the system remains projectively polynomial, which is summarized by the lemma and theorem below.

Lemma 4.2.1 *Consider solving via the DPM the PDE*

$$\begin{cases} u_t & = Mu \\ u(\cdot, 0) & = u_0 \end{cases}$$

for some linear differential operator M and initial matrix u_0. Assume that $L \approx M$ is corresponding linear finite difference operator. Assume L is defined by

$$Lu_{i_1 i_2 \ldots i_m} = \sum_{j_1, j_2, \ldots, j_m} \alpha_{j_1, j_2, \ldots, j_m} u_{i_1 + j_1, i_2 + j_2, \ldots, i_m + j_m}$$

Then the PDE is projectively polynomial.

Proof. This follows directly from the definition since Lu is the sum of degree one terms. □

Since the linear operator L is projectively polynomial, we see by extension, the general problem is also projectively polynomial.

Theorem 4.2.1 *Consider solving the PDE*

$$\begin{cases} u_t & = P(u, \frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}, \ldots, \frac{\partial^2 u}{x^2}, \ldots) \\ u(\cdot, 0) & = u_0(\cdots) \end{cases}$$

by using the DPM method of

$$\begin{cases} u_t & = P(u, L_1 u, L_2 u, \ldots, L_m u) \\ u(\cdot, 0) & = u_{0_{i_1 i_2 \ldots i_m}} \end{cases}$$

where each L_i, $i = 1, \ldots m$ are linear as in Lemma 4.2.1. Then the system is projectively polynomial.

Proof. From Lemma 4.2.1, we know that each L_i is polynomial and in fact linear. The resulting system is the composition of polynomial terms and has to be projectively polynomial. □

As a result, the results of the MPM method with regards to truncating terms can be extended to DPM. Thus, after each iterate is computed, we truncate the terms to degree n, assuming we have computed the n-th iterate.

4.2.2 Boundary Conditions

The boundary conditions need to be handled carefully in DPM due to the use of higher degree iterates. When the degree of the iterate is one, normal boundary conditions are applied, similar to a FD scheme. However, since the degree one iterate is used to compute the second degree iterate, and similarly for degree three and higher, we must calculate the values at the boundary. The approach we take is to compute one side derivatives for the FD scheme at the boundaries. Figure 4.1, illustrates the problem with boundary conditions. When using a degree one iterate, the terms at point x_1 and x_J need to be calculated, where J is the number of discrete data points and the linear operator has a 3 point stencil. If we do not enforce the one sided derivatives at this stage, the data at x_1 and x_J is invalid for the degree two iterate, and then, x_2 and x_{J-1} is invalid after the second iterate is computed. This continues, reducing the available data as the degree of the Picard iterate increases, unless we enforce one sided derivatives at each step.

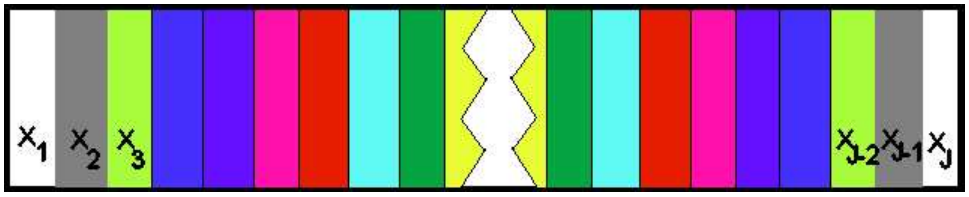

Figure 4.1: Boundary Conditions The similarly shaded regions are lost if one sided derivatives are not enforced as the degree of the iterates increase.

As a result, we enforce the linear operator to compute one sided derivatives at the edges of the domain. For example, in the one dimension example of $u_t = u_{xx}$ with L being the centered difference scheme, we use the end condition in one dimension to be

$$Lu_J = \frac{u_J - 2u_{J-1} + u_{J-2}}{\Delta x^2}$$

and a similar term for Lu_1. Now, we have all the values, and there is no ambiguity in the values at the boundary for any of the degrees of the iterates.

4.3 Comparison of MPM with DPM and Finite Differences

In this section, we compare the MPM to the DPM. While the MPM computes the power series form for the function u, the DPM computes the exact same result, but with an approximation

to the derivatives at each step. For example, we consider solving the following PDE

$$\begin{cases} u_t & = u_x \\ u(x,0) & = u_0(x) \end{cases}$$

compared to the DPM method of

$$\begin{cases} u_t & = Lu \\ u(x,0) & = u_0(x) \end{cases} \tag{4.1}$$

where L is the operator for central difference scheme. If we compute the iterates for MPM we get,

$$\begin{aligned} p^{(0)}(t) &= u_0 \\ p^{(1)}(t) &= u_0 + u_{0_x} t \\ p^{(2)}(t) &= u_0 + u_{0_x} t + u_{0_{xx}} t^2/2 \\ p^{(3)}(t) &= u_0 + u_{0_x} t + u_{0_{xx}} t^2/2 + u_{0_{xxx}} t^3/6 \\ &\ldots\ldots \end{aligned}$$

while the DPM computes

$$\begin{aligned} \phi^{(0)}(t) &= u_0 \\ \phi^{(1)}(t) &= u_0 + L(u_0)t \\ \phi^{(2)}(t) &= u_0 + L(u_0)t + L^2(u_0)t^2/2 \\ \phi^{(3)}(t) &= u_0 + L(u_0)t + L^2(u_0)t^2/2 + L^3(u_0)t^3/6 \\ &\ldots\ldots \end{aligned}$$

and we note that L^2 would be a 5 point approximation to u_{xx} and L^3 would be a 7 point approximation to u_{xxx}. By choosing L to be the centered difference scheme, (4.1) corresponds to the approximated derivatives.

If we consider a nonlinear example, the correspondence between derivatives and the linear operator is still true. If we consider Burger's equation

$$\begin{cases} u_t + (\frac{u^2}{2})_x = 0 \\ u(x,0) = \alpha(x) \end{cases},$$

we can first project to a simpler polynomial system to ease our calculations. Let $w = \frac{u^2}{2}$ to

get the equivalent system

$$\begin{cases} u_t + w_x = 0 & u(x,0) = \alpha(x) \\ w_t + uw_x = 0 & w(x,0) = \frac{\alpha^2(x)}{2} = \beta(x) \end{cases}.$$

Consider the following integral form of this system

$$u(x,t) = \alpha(x) - \int_0^t w_x(x,\tau)d\tau$$

$$w(x,t) = \beta(x) - \int_0^t u(x,\tau)w_x(x,\tau)d\tau$$

and the Picard iteration for this system

$$u^{(k+1)}(x,t) = \alpha(x) - \int_0^t w_x^{(k)}(x,\tau)d\tau$$

$$w^{(k+1)}(x,t) = \beta(x) - \int_0^t u^{(k+1)}(x,\tau)w_x^{(k+1)}(x,\tau)d\tau.$$

Now let L be a linear approximation for $\frac{\partial}{\partial x}$. This leads to the following discrete in space approximation

$$u_j^{(k+1)}(t) = \alpha_j - \int_0^t L[w_j^{(k)}(\tau)]d\tau$$

and

$$w_j^{(k+1)}(t) = \beta_j - \int_0^t u_j^{(k+1)}(\tau)L[w_j^{(k+1)}(\tau)]d\tau$$

to this iteration where j indicates $x_j = j\Delta x$. We let

$$u_j^{(0)} = \alpha_j \text{ and } w_j^{(0)} = \beta_j.$$

The Picard iterates are for $k = 0$ are

$$u_j^{(1)}(t) = \alpha_j - \int_0^t L[w_j^{(0)}(\tau)]d\tau = \alpha_j - L[w_j^{(0)}]t$$

$$w_j^{(1)}(t) = \beta_j - \int_0^t u_j^{(0)}(\tau)L[w_j^{(0)}(\tau)]d\tau = \beta_j - u_j^{(0)}L[w_j^{(0)}]t$$

Similarly for $k = 1$, we get

$$\begin{aligned} u_j^{(2)}(t) = &\ \alpha_j - \int_0^t L[w_j^{(1)}(\tau)]d\tau = \alpha_j - \int_0^t L[\beta_j - u_j^{(0)} L[w_j^{(0)}]\tau]d\tau \\ &= \alpha_j - L[w_j^{(0)}]t + L[u_j^{(0)} L[w_j^{(0)}]]\tfrac{t^2}{2} \end{aligned}$$

and

$$\begin{aligned} w_j^{(2)}(t) = &\ \beta_j - \int_0^t u_j^{(1)}(\tau) L[w_j^{(1)}(\tau)]d\tau = \beta_j - \int_0^t (\alpha_j - L[w_j^{(0)}]\tau) L[\beta_j - u_j^{(0)} L[w_j^{(0)}]\tau)]d\tau \\ &= \beta_j - u_j^{(0)} L[w_j^{(0)}]t + (u_j^{(0)} L[u_j^{(0)} L[w_j^{(0)}]] + L[w_j^{(0)}]^2)\tfrac{t^2}{2} \end{aligned}$$

Then for $k = 2$ we have

$$\begin{aligned} u_j^{(3)}(t) = &\ \alpha_j - \int_0^t L[w_j^{(2)}(\tau)]d\tau \\ = &\ \alpha_j - \int_0^t L[\beta_j - u_j^{(0)} L[w_j^{(0)}]\tau + (u_j^{(0)} L[u_j^{(0)} L[w_j^{(0)}]] + L[w_j^{(0)}]^2)\tfrac{\tau^2}{2}d\tau \\ = &\ \alpha_j - L[w_j^{(0)}]t + L[u_j^{(0)} L[w_j^{(0)}]]\tfrac{t^2}{2} - L[u_j^{(0)} L[u_j^{(0)} L[w_j^{(0)}]] + (L[w_0^j])^2)]\tfrac{t^3}{3!} \end{aligned}$$

and

$$\begin{aligned} w_j^{(3)}(t) = &\ \beta_j - \int_0^t u_j^{(2)}(\tau) L[w_j^{(2)}(\tau)]d\tau = \beta_j - \int_0^t (\alpha_j - L[w_j^{(0)}]\tau + L[u_j^{(0)} L[w_j^{(0)}]]\tfrac{\tau^2}{2}) * \\ &\quad L[\beta_j - u_j^{(0)} L[w_j^{(0)}]\tau + (u_j^{(0)} L[u_j^{(0)} L[w_j^{(0)}]] + L[w_j^{(0)}]^2)\tfrac{\tau^2}{2}]d\tau \\ &= \beta_j - u_j^{(0)} L[w_j^{(0)}]t + (u_j^{(0)} L[u_j^{(0)} L[w_j^{(0)}]] + L[w_j^{(0)}]^2)\tfrac{t^2}{2} \\ &\quad -(u_j^0 L[u_j^{(0)} L[u_j^{(0)} L[w_j^{(0)}]] + L[w_j^{(0)}]^2] + \\ &\quad 3L[w_j^0] L[u_j^0 L[w_j^0] + L[w_j^0] L[u_j^{(0)} L[w_j^{(0)}]])\tfrac{t^3}{3!} \end{aligned}$$

And we can continue for higher values of k. However, we can now replace w_j^0 with $(u_j^0)^2/2$ and have

$$\begin{aligned} u_j^{(1)}(t) &= \alpha_j - L[\tfrac{(u_j^0)^2}{2}]t \\ u_j^{(2)}(t) &= \alpha_j - L[\tfrac{(u_j^0)^2}{2}]t + L[u_j^{(0)} L[\tfrac{(u_j^0)^2}{2}]]\tfrac{t^2}{2} \\ u_j^{(3)}(t) &= \alpha_j - L[\tfrac{(u_j^0)^2}{2}]t + L[u_j^{(0)} L[\tfrac{(u_j^0)^2}{2}]]\tfrac{t^2}{2} - L[u_j^{(0)} L[u_j^{(0)} L[\tfrac{(u_j^{(0)})^2}{2}]] + (L[\tfrac{(u_j^{(0)})^2}{2}])^2]\tfrac{t^3}{3!} \end{aligned}$$

We note that these iterates are the same as the MPM iterates, except with the linear approximation L applied instead of differentiating at each step. The pattern can now be extended as well for other nonlinear problems. This process also works on generating a space discretization with time Picard iteration on any equation of the form

$$\begin{cases} u_t + (f(u))_x = 0 \\ u(x, 0) = \alpha \end{cases}$$

where f is polynomial.

The DPM method iterates of degree one and two are related to standard FD schemes. The forward time FD scheme is related to the degree one iterate of DPM. When the degree of DPM is two, we get the DPM method is equivalent to the Lax-Wendroff scheme when the appropriate operator is choosen. The following theorem illustrates the relations between the forward time difference scheme and the Lax-Wendroff scheme.

Theorem 4.3.1 *Consider applying the Discretized Picard Method to the equation*

$$\begin{cases} u_t & = Mu \\ u(\cdot, 0) & = u_0 \end{cases}$$

for some linear differential operator M and initial matrix u_0. Assume that $L \approx M$ is corresponding linear finite difference operator. Then, the degree one Picard iterate is the same as the finite difference scheme using the operator L and the degree two Picard iterate is the Lax-Wendroff scheme, if the operator L is chosen to use a stencil with half steps.

Proof. For the degree one iterate, we compute the iterate

$$\phi^{(1)}(t) = u_0 + \int_0^t Lu_0 \, ds$$

Evaluating, we get

$$\phi^{(1)}(t) = u_0 + Lu_0 t$$

and by rearranging we get

$$\begin{aligned} \phi^{(1)}(t) &= u_0 + Lu_0 t \\ \frac{\phi^{(1)}(t) - u_0}{t} &= Lu_0 \\ \frac{\phi^{(1)}(t) - \phi^{(0)}(t)}{t} &= L[\phi(0)(t)] \end{aligned}$$

Letting $u^{n+1} = \phi^{(1)}(t)$ and $u^n = \phi^{(1)}(t)$ we get

$$\frac{u^{n+1} - u^n}{t} = Lu^n$$

Now letting $t = \Delta t$, we get the desired result.

For the second degree iterate, we compute

$$\phi^{(2)}(t) = u_0 + \int_0^t L(\phi^{(1)}(t)(s))\, ds$$

By expanding and rearranging, we obtain:

$$\begin{aligned}\phi^{(2)}(t) &= u_0 + \textstyle\int_0^t L(u_0 + Lu_0 s)\, ds \\ &= u_0 + \textstyle\int_0^t Lu_0 + L^2 u_0 s\, ds \\ &= u_0 + Lu_0 t + L^2 u_0 t^2/2\end{aligned}$$

But, we note that the Lax Wendroff method computes

$$u_0 + u_t t + u_{tt} t^2/2$$

and using that $u_{tt} = L(Lu) = L^2 u$, and choosing the correct operator L with half step points for the stencil, the proof is complete. □ ∎

4.4 Stability

In this section, we consider the stability of the DPM as the degree of the Picard iterates increase. In general, we cannot determine a condition for any degree m, but we show that the stability region is increasing for all our examples. For the first example, we consider solving the transport equation

$$\begin{cases} u_t & = u_x \\ u(\cdot, 0) & = u_0 \end{cases}$$

using the central difference scheme

$$Lu_j = \frac{u_{j+1} - u_{j-1}}{2\Delta x}$$

with one sided difference at the boundary and in one dimension. The first assertion we make is about the term $L^n u$, since this is needed to compute the Von-Neumann analysis for stability.

Lemma 4.4.1 *For the linear operator $Lu_j = \frac{u_{j+1}-u_{j-1}}{2\Delta x}$, we have that*

$$L^n u_j = \frac{\sum_{i=0}^{n} (-1)^i \binom{n}{i} u_{j-2i+n}}{(2\Delta x)^n}$$

Proof. We illustrate a method that is less algebraic and relies on functionology and combinatorics for a proof. For further reference, please see [71, 79]. We define a sequence (U_n) in $\mathbb{R}[[x]]$ by $U_0(x) = \sum_j u_j x^j$ and $U_n(x) = \sum_j L^n(u_j)x^j$. Since L is linear, we have the relation

$$L^n(u_j) = \frac{L^{n-1}(u_{j+1}) - L^{n-1}(u_{j-1})}{2\Delta x}$$

for $n > 0$. Multiplying by x^j and summing over all $j \in \mathbb{Z}^+$ we get that

$$\begin{aligned} U_n(x) &= \sum_j \left[\frac{L^{n-1}(u_{j+1}) - L^{n-1}(u_{j-1})}{2\Delta x}\right] x^j \\ &= \frac{1}{2\Delta x}\left[\frac{U_{n-1}(x)}{x} - xU_{n-1}(x)\right] \\ &= \frac{1}{2\Delta x}\frac{1-x^2}{x} U_{n-1}(x) \end{aligned}$$

Hence, we have $U_n(x) = \left(\frac{1}{2\Delta x}\frac{1-x^2}{x}\right)^n U_0(x)$. Thus, we have

$$\begin{aligned} L^n(u_j) &= [x^j]\left(\frac{1}{2\Delta x}\frac{1-x^2}{x}\right)^n U_0(x) \\ &= \left(\frac{1}{2\Delta x}\right)^n [x^{j+n}](1-x^2)^n U_0(x) \end{aligned}$$

where $[x^j]$ denotes the j-th coefficient of the expansion immediately to the right. If we apply the binomial theorem to the right hand side we see that

$$\begin{aligned} L^n(u_j) &= \left(\frac{1}{2\Delta x}\right)^n \sum_{i=0}^{n} \binom{n}{i}(-1)^i u_{(j+n)-(2i)} \\ &= \left(\frac{1}{2\Delta x}\right)^n \sum_{i=0}^{n} \binom{n}{i}(-1)^i u_{j-2i+n} \end{aligned}$$

which completes the proof. □ ∎

Now, given we have each term explicitly, we can now compute the stability polynomial for any degree of our Picard iterate.

Theorem 4.4.1 *The Picard iterates of degree m for*

$$\begin{cases} u_t & = u_x \\ u(\cdot, 0) & = u_0 \end{cases}$$

using the central scheme result in the stability polynomial

$$\lambda = 1 + \sum_{n=1}^{m} \left[\frac{\nu^n}{n!} \sum_{l=1}^{n} (-1)^l \binom{n}{l} e^{i(n-2l)} \right]$$

where $\nu = \frac{\Delta t}{2\Delta x}$.

Proof. From the Picard iterates, we compute the degree m iterate to be

$$\phi^{(m)}(t) = u_0 + Lu_0 t + L^2 u_0 t^2/2! + \dots L^m u_0 t^m/m!$$

Let $u^m = \phi^{(m)}(t)$. Then, applying the formula above, we get

$$u_j^m = u_{0_j} + Lu_{0_j} t + \cdots + L^m u_{0_j} t^m/m!$$

Then, settings $t = \Delta t$ and $\nu = \frac{\Delta t}{2\Delta x}$, we obtain

$$u_j^m = u_{0_j} + \nu L u_{0_j} + \nu^2/2! L u_{0_j} + \cdots + \nu^m/m! L^m u_{0_j}$$

or

$$u_j^m = u_{0_j} + \sum_{n=1}^{m} L^m u_{0_j} \nu^n/n!$$

By applying theorem 4.4.1, we obtain

$$u_j^m = u_{0_j} + \sum_{n=1}^{m} \frac{\nu^n}{n!} \left[\sum_{l=0}^{n} (-1)^l \binom{n}{l} u_{j-2l+n} \right]$$

Then, letting $u_j = \lambda^n e^{ij\delta x}$ we get

$$\lambda = 1 + \sum_{n=1}^{m} \frac{\nu^n}{n!} \left[\sum_{l=0}^{n} (-1)^l \binom{n}{l} e^{i(n-2l)} \right]$$

and this completes the proof. □ ∎

Now, let us consider the case of the first four iterates to illustrate the change in the stability condition as the degree increases:

Theorem 4.4.2 *The stability condition for the first four iterates of*

$$\begin{cases} u_t & = u_x \\ u(\cdot, 0) & = u_0 \end{cases}$$

using the central difference scheme are

Degree	**Stability Condition**
1	*unstable*
2	*unstable*
3	$\nu \leq \frac{\sqrt{3}}{2}$
4	$\nu \leq \sqrt{2}$

for $\nu = \frac{\Delta t}{2\Delta x}$.

Proof. While the result for $m = 1$ case can be obtained by the usual means for FD scheme, we wish to illustrate an alternate method that makes the computation slightly easier and more straightforward. We consider the stability polynomial

$$\lambda = 1 + \nu \left[e^{ij\Delta x} - e^{-ij\Delta x} \right]$$

for degree one or

$$\lambda = 1 + 2i\nu \sin\theta$$

where $\theta = j\Delta x$. We have

$$|\lambda| = \lambda\overline{\lambda} = 1 + 4\nu^2 \sin^2\theta$$

showing the scheme is unstable. To complete our formal analysis, define

$$f(\nu, \theta) := 1 + 4\nu^2 \sin^2\theta$$

Then, we fix ν and find the minimum of θ by calculus:

$$f_\theta = 8\nu^2 \sin\theta\cos\theta = 0$$

Hence, we have $\theta = 0, \pi, \pi/2, -\pi/2$. Filling in those values, we obtain the set of polynomials

$$f(\nu, 0) = f(\nu, \pi) = 1$$

$$f(\nu, \pi/2) = f(\nu, -\pi/2) = 1 + 4\nu^2$$

and we want both these to to be less than one for $\nu \geq 0$, ie:

$$\begin{cases} 1 & \leq 1 \\ 1 + 4\nu^2 & \leq 1 \end{cases}$$

However, no choice of ν satisfies all these requirements and we conclude that the degree one polynomial is unstable.

Now, we complete a similar analysis on degree two and get the same result. But for degree $m = 3$, we have

$$\lambda = 1 + 2i\nu \sin\theta + \nu^2(\cos 2\theta - 1) + \frac{\nu^3}{3} i \left[\sin(3\theta) - 3\sin\theta\right]$$

We define

$$f(\nu, \theta) := |\lambda|^2$$

and compute $\frac{\partial f}{\partial \theta}(\nu, \theta) = 0$ and get the real solutions are

$$\theta = 0, -\frac{\pi}{2}, \frac{\pi}{2}.$$

Thus, we have the polynomial conditions

$$\begin{cases} f(\nu, 0) = f(\nu, \pi) = 1 \leq 1 \\ f(\nu, -\pi/2) = f(\nu, \pi/2) = 1 - 4/3\nu^4 + 16/9\nu^6 \leq 1 \end{cases}$$

which is satisfied when $\nu \leq \frac{\sqrt{3}}{2}$. The bound for DPM iterate of degree four is similar to derive and the calculations result in $\nu \leq \sqrt{2}$. □ ∎

In the case of the degree three and four iterates, the CFL condition is violated. Thus, we need not choose any higher degree iterate than three for the DPM. As a result, we use a degree three iterate with $\nu \leq 1$.

For the heat equation in one dimension, a similar analysis can be completed and is listed below.

Theorem 4.4.3 *The stability condition for the first four iterates of*

$$\begin{cases} u_t & = u_{xx} \\ u(\cdot, 0) & = u_0 \end{cases}$$

using the central difference scheme are

Degree	**Stability Condition**
1	$\nu \leq \frac{1}{2}$
2	$\nu \leq \frac{1}{2}$
3	$\nu \leq \frac{\sqrt[3]{4+\sqrt{17}}}{4} - \frac{1}{4\sqrt[3]{4+\sqrt{17}}} + \frac{1}{4}$
4	$\nu \leq \frac{1}{12}\sqrt[3]{172+36\sqrt{29}} - \frac{5}{3\sqrt[3]{172+36\sqrt{29}}} + \frac{1}{3}$

for $\nu = \frac{\Delta t}{(\Delta x)^2}$.

A similar analysis work for the two dimension datasets. We consider the process of applying the heat equation in two dimensions and we get a corresponding analysis for stability from the theorem below.

Theorem 4.4.4 *The stability condition for the first four iterates for solving*

$$\begin{cases} u_t & = u_{xx} + u_{yy} \\ u(\cdot, 0) & = u_0 \end{cases}$$

via DPM using the central difference scheme is

Degree	**Stability Condition**
1	$\frac{1}{4}$
2	$\frac{1}{4}$
3	$\nu \leq \frac{1}{2}\left[\frac{\sqrt[3]{4+\sqrt{17}}}{4} - \frac{1}{4\sqrt[3]{4+\sqrt{17}}} + \frac{1}{4}\right] \approx 0.3140931658$
4	$\nu \leq \frac{1}{2}\left[\frac{\sqrt[3]{172+36\sqrt{29}}}{12} - \frac{5}{3\sqrt[3]{172+36\sqrt{29}}} + \frac{1}{3}\right] \approx 0.3481616954$

for $\nu_x = \nu_y = \nu = \frac{\Delta t}{(\Delta x)^2}$.

Proof. We can handle the two dimension case similar to the one dimensional case. Here we need to form $f(\nu_x, \nu_y, \theta, \omega) = \lambda$ and then solve

$$\begin{cases} f_\theta(\nu_x, \nu_y, \theta, \omega) = 0 \\ f_\omega(\nu_x, \nu_y, \theta, \omega) = 0 \end{cases}$$

For the degree two iterate, we get

$$\begin{cases} \theta = 0 & \omega = 0 \\ \theta = 0 & \omega = \pi \\ \theta = \pi & \omega = 0 \\ \theta = \pi & \omega = \pi \end{cases}$$

Then we compute $f(\nu, nu, \cdot, \cdot)$ for each value of θ and ω and we get

$$\begin{cases} -1 \leq 1 \leq 1 \\ -1 \leq 1 - 4\nu + 8\nu^2 \leq 1 \\ -1 \leq -1 \leq 1 - 4\nu + 8\nu^2 \leq 1 \\ -1 \leq 1 - 8\nu \leq 1 \end{cases}$$

Solving for all cases and combining the answer we get that$\nu \leq 1/4$. We can apply the same analysis and compute the result for degree three and four. ∎

We note here, that we can let $\nu_x \neq \nu_y$ by writing $\nu_y = c\nu_x$ for some constant c and apply the same analysis above and get a similar result when the space grid is not square.

4.5 Numerical Implementation and Examples

All the examples are implemented in Matlab using a 2Ghz Pentium IV. In order to implement the DPM, an object class for computing the iterates was developed that utilizes matrix coefficients. This object class implements all the basic mathematical operations and includes an integral operator over the time domain. The linear operators are implemented as pluggable modules for the DPM routine which makes the method versatile when considering different types of PDEs and testing different operators used for each derivative. All the floating point arithmetic is computed in double precision.

The first example we consider is

$$\begin{cases} u_t = u_x \\ u(x,0) = \sin x \end{cases}.$$

We use the centered difference operator for the first derivative, which is $Lu_j = \frac{u_{j+1}-u_{j-1}}{2\Delta x}$. We choose $\Delta x = 1/100$, and ran the method for a total of 200 iterations using a degree three iterate with $\Delta t = \Delta x$, the maximum value allowed by the CFL condition. The result is shown in Figure 4.2 for times $t = 0, 2, 4$. We note that while the first two iterates are unstable, using the degree three iterate results in a stable method.

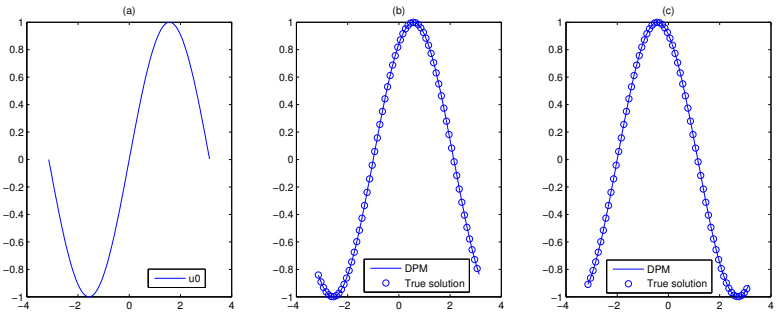

Figure 4.2: Degree 4 iterate for solving $u_t = u_x$ using a centered difference scheme.

The second example is using the heat equation in one dimension. We used the centered difference scheme $Lu_j = \frac{u_{j+1}-2u_j+u_{j-1}}{(\Delta x)^2}$. The degree four iterate is used again for computation and the result is shown in Figure 4.3. We note the computational cost of computing using the higher degree iterate allows us to compute the final result in less timesteps.

The third example we present is to solve inviscid form of Burger's equation, which is

$$\begin{cases} u_t & = -uu_x \\ u(0,x) & = f(x) \end{cases}. \tag{4.2}$$

We choose $f(x) = -3/\pi \tan^{-1} x + 3/2$. We see the computed result up to the start of the shock formation in Figure 4.4(a) using DPM. In (b), the same result is computed using the Lax-Wendroff scheme. However, the stability condition is $\mathcal{O}(\Delta t/(\Delta x)^2)$ for Lax-Wendroff, but the third degree DPM only requires $\Delta t/\Delta x \leq 1/4$. As a result, 21000 iterates must be computed for the Lax-Wendroff versus 420 for the DPM method. The computational

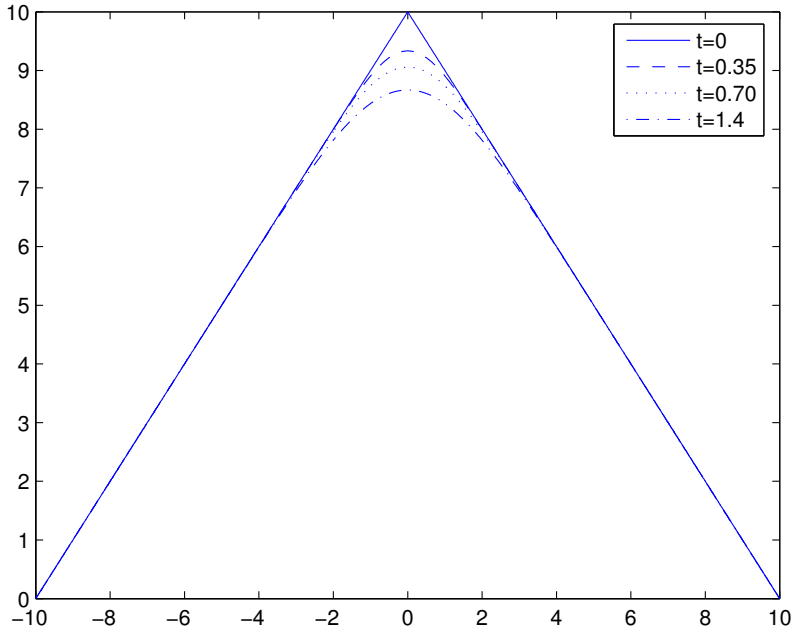

Figure 4.3: Degree 4 iterate for solving $u_t = u_{xx}$ using a centered difference scheme.

savings, even with computing the higher degree iterates, is substantial.

The final example we present is an image smoothing example. Using the fourth degree iterate for solving $u_t = \Delta u$ with the noisy initial image in Figure 4.5(a), we compute the result in less time. The intermediate and final results are shown in Figure 4.5(b) and (c). Here, we chose the maximum value for $\nu = \Delta t/(\Delta x)^2$ in Theorem 4.4.4.

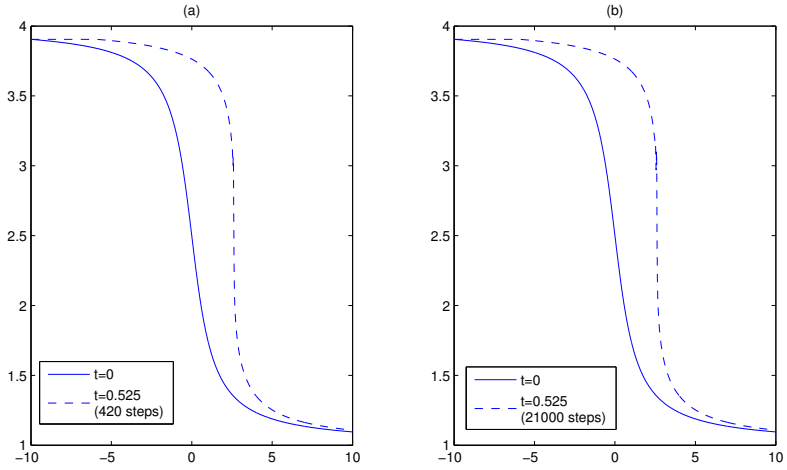

Figure 4.4: Degree 3 iterate for solving $u_t = -uu_x$ in the present of a shock. (a) is computed via DPM. (b) is the same result using Lax-Wendroff

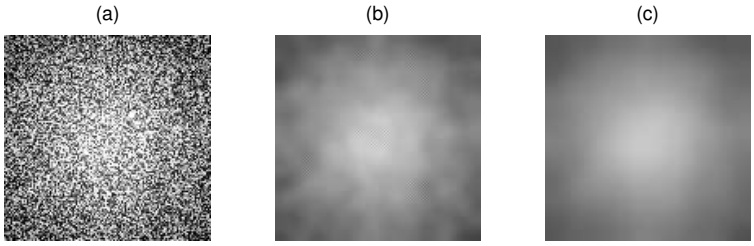

Figure 4.5: Degree 3 iterate for solving $u_t = \Delta u$ in 2D using a centered difference scheme. Image (a) is the initial noisy image. Image (b) is the result after 5 iterations. Image (c) is the result after 10 iterations.

5 EIGIFP

5.1 Inverse Free Krylov Subspace Method

In this chapter, we consider locating a small number of the algebraically smallest eigenvalues of the pencil (A, B). We assume that A and B are symmetric and sparse, and that $B > 0$. We utilize the theory of Golub and Ye [40] for an inverse free method for the generalized eigenvalue problem by computing approximations to the shifted pencil $(A - \rho_k B, B)$. By computing the shifted pencil, no direct inversion of B is required and we avoid problems that arise when B is ill-conditioned.

The core of `eigifp` is this inverse free preconditioned Krylov subspace projection method[40]. First, we describe the basic method and we describe some enhancements incorporated into `eigifp`. We note that the basic method with a different development of preconditioning strategy is described by Knyazev [43] (see also [3, p.360]). A prior version of this work also appears in [51].

Throughout, we shall consider the smallest eigenvalue of (A, B). Indeed, a direct call to `eigifp` computes the k smallest eigenvalues. To compute the largest eigenvalue of (A, B), we just need to compute the smallest eigenvalue of $(-A, B)$ and reverse the sign.

5.1.1 Basic Method

Given an approximate eigenvector x_k, we construct a new approximation x_{k+1} by the Rayleigh-Ritz projection of (A, B) onto the Krylov subspace

$$K_m(A - \rho_k B, x_k) := \text{span}\{x_k, (A - \rho_k B)x_k, \ldots, (A - \rho_k B)^m x_k\}$$

where $\rho_k = x_k^T A x_k / x_k^T B x_k$ is the Rayleigh quotient and m is a parameter to be chosen. Specifically, let Z_m be the matrix consisting of the basis vectors of K_m. We then form the matrices

$$A_m = Z_m^T (A - \rho_k B) Z_m$$

and

$$B_m = Z_m^T B Z_m,$$

and find the smallest eigenpair (μ_1, v_1) for (A_m, B_m). Then the new approximate eigenvector is

$$x_{k+1} = Z_m v_1$$

and, correspondingly, the Rayleigh quotient

$$\rho_{k+1} = \rho_k + \mu_1$$

is a new approximate eigenvalue.

Iterating with k, the above forms the outer iteration of the method. Now, to construct the basis vectors Z_m, an inner iteration will be used, where m is dimension of the subspace in the approximation. Since m can vary independently of k, we call this construction of the basis vectors an inner iteration, typically using the Lanczos or Arnoldi algorithms. We use either the Lanczos algorithm to compute an orthonormal basis or the Arnoldi algorithm to compute a B-orthonormal basis. While in theory the outer iteration is independent of the bases constructed, they have different numerical stability. Experiments(see [40]) lead us to use an orthonormal basis by the Lanczos method when the outer iteration is not preconditioned and to use a B-orthonormal basis by the Arnoldi algorithm when the outer iteration is preconditioned (see below).

Golub and Ye [40] showed that ρ_k converges to an eigenvalue and x_k converges in direction to an eigenvector. Furthermore, they have the following local convergence result.

Theorem 5.1.1 *Let $\lambda_1 < \lambda_2 \leq \cdots \leq \lambda_n$ be the eigenvalues of (A, B) and (ρ_{k+1}, x_{k+1}) be the approximate eigenpair obtained from (ρ_k, x_k) by one step of the inverse free Krylov subspace method. Let $\sigma_1 < \sigma_2 \leq \cdots \leq \sigma_n$ be the eigenvalues of $A - \rho_k B$. If $\lambda_1 < \rho_k < \lambda_2$, then*

$$\rho_{k+1} - \lambda_1 \leq (\rho_k - \lambda_1)\epsilon_m^2 + \mathcal{O}((\rho_k - \lambda_1)^{3/2}) \tag{5.1}$$

where

$$\epsilon_m = \min_{p \in \mathcal{P}_m, p(\sigma_1)=1} \max_{i \neq 1} |p(\sigma_i)| \leq 2\left(\frac{1-\sqrt{\psi}}{1+\sqrt{\psi}}\right)^m,$$

$\mathcal{P}_m$ denotes the set of all polynomials of degree not greater than m, and

$$\psi = \frac{\sigma_2 - \sigma_1}{\sigma_n - \sigma_1}.$$

This bound shows that the speed of convergence depends on m and the relative gap between σ_1 and σ_2, ψ. We also note that a key condition of the theorem is that $\rho_k \in (\lambda_1, \lambda_2)$.

We accelerate the convergence of `eigifp` by increasing the spectral gap ψ through an equivalent transformation that we call preconditioning. One way of doing this is the congruence transformation

$$(\hat{A}, \hat{B}) := (L^{-1}AL^{-T}, L^{-1}BL^{-T}), \tag{5.2}$$

which preserves the eigenvalues λ_i but changes σ_i. Indeed, applying our algorithm to $(\hat{A}, \hat{B})$, the speed of convergence depends on the spectral gap of

$$\hat{A} - \rho_k \hat{B} = L^{-1}(A - \rho_k B)L^{-T}.$$

By choosing L to be the factor in the LDL^T factorization of $A - \rho_k B$ with D being a diagonal matrix of ± 1, we obtain an ideal situation where $\psi = 1$ and hence $\epsilon_m = 0$. In practice, we can use an incomplete LDL^T factorization to arrive at a small ϵ_m.

As in the preconditioned conjugate gradient method, preconditioning transformation (5.2) can be carried out implicitly; Golub and Ye [40] give a detailed algorithm. Indeed, the only operation involving the preconditioning transformation is $L^{-T}L^{-1}$. Thus, if M is approximately $A - \lambda_1 B$ and is symmetric positive definite, then we only need M^{-1} to implement the preconditioned iteration. We call M^{-1} a preconditioner, which need not be in the factorized form $L^{-T}L^{-1}$.

We also note the method has recently been extended to the block case similar to the Lanczos block algorithm in [63]. This block method allows one to calculate multiple or clustered eigenvalues in a fraction of the time the non-block version of `eigifp` required or failed to faithful calculate to the desired tolerance.

5.1.2 LOBPCG Type Subspaces Enhancement

Our algorithm reduces to the steepest descent method when $m = 1$. Knyazev [45, 44] has derived a method, called locally optimal preconditioned conjugate gradient method (LOBPCG), where the previous approximate eigenvector x_{k-1} in the steepest descent method is added to $\text{span}\{x_k, (A - \rho_k B)x_k\}$ and a new approximate eigenvector is constructed from $\text{span}\{x_{k-1}, x_k, (A - \rho_k B)x_k\}$ by projection. It results in a conjugate gradient like algorithm and Knyazev has observed a dramatic speedup in convergence over the steepest descent method.

Here, we also apply this technique to our method to enhance the Krylov subspace $K_m(A - \rho_k B, x_k)$, namely, at step k, we use the Rayleigh-Ritz projection on the enhanced subspace $\text{span}\{x_{k-1}, x_k, (A - \rho_k B)x_k, \ldots, (A - \rho_k B)^m x_k\}$. In `eigifp`, we compute $x_k - x_{k-1}$ at every

step and, when a basis Z_m has been constructed for $K_m(A - \rho_k B, x_k)$, we orthogonalize $x_k - x_{k-1}$ against Z_m to obtain z_{m+2} and then extend the basis matrix to

$$\hat{Z}_m = \begin{bmatrix} Z_m & z_{m+2} \end{bmatrix}.$$

Our experiments have shown that this provides noticeable speedup in convergence; yet it only incurs very little extra cost.

5.1.3 Deflation

The algorithm we have described finds the smallest eigenvalue. Once we have this, we can continue to find the next smallest eigenvalue by the same procedure through deflation. Because of the form of the Krylov subspace, the deflation needs to be done slightly differently from standard methods like the Lanczos algorithm.

When p eigenpairs have been found, let V_p be the matrix consisting of the p eigenvectors with $V_p^T B V_p = I$ and Λ_p be the diagonal matrix consisting of the corresponding eigenvalues, i.e. $AV_p = BV_p\Lambda_p$. We consider

$$(A_p, B) := (A + (BV_p)\Sigma(BV_p)^T, B) \tag{5.3}$$

where $\Sigma = \text{diag}\{\sigma_i - \lambda_i\}$ with σ_i any value chosen to be greater than λ_{p+2}. Then, it is easy to check that the eigenvalues of (5.3) are the union of $\{\lambda_{p+1}, \lambda_{p+2}, \cdots, \lambda_n\}$ and $\{\sigma_1, \cdots, \sigma_p\}$. Thus, its smallest eigenvalue is λ_{p+1}, which is computed by applying our method to (5.3).

5.1.4 Black-box Implementations

In order to implement our method as a black-box routine, we need to address the following issues:

1. How do we carry out preconditioned iterations without a user supplied preconditioner?
2. How do we choose the number of inner iterations to optimize the overall performance?
3. When do we terminate the iteration?

We describe now how these are dealt with in `eigifp`. We note that users always have options to override the default settings (see Section 3).

To implement the preconditioned iteration, we use an approximate eigenvalue σ and compute an incomplete LDL^T factorization of $A - \sigma B$ using the MATLAB threshold ILU

routine `luinc` (with a default threshold 10^{-3}). If an initial approximation σ is not provided by the user, `eigifp` will start with the non-preconditioned iterations and switch to the preconditioned one when a good approximate eigenvalue is identified. For this, we first estimate the error $\lambda_1 - \rho_k$ using the eigenvector residual and an estimated gap between the first two eigenvalues of $A - \rho_k B$. We then switch to the preconditioned iterations when the error is below a certain threshold. As a safeguard against this switching occurring too early, we revert back to the non-preconditioned iteration if the subsequent approximate eigenvalues ρ_k significantly drift away from the shift point chosen.

The only parameter to be chosen in our method is the number of inner iterations m. Experiments have shown that an optimal value of m is larger if the problem is more difficult while it is smaller if the problem is easier (e.g., if we have a good preconditioner). However, we do not know which value works best for a given matrix. By default, we adaptively choose m in `eigifp` as follows. Starting with a small value of m (2 for non-preconditioned iterations and 1 for preconditioned iterations), we double the value m and compute its convergence rate after some iterations. We continue increasing m as long as the rate of convergence has roughly double, but reset it to the previous value when the rate of convergence is not increased proportionally.

Finally, we terminate the iteration when the 1-norm of the residual $r_k = (A - \rho_k B)x_k$ drops below a certain threshold. The default threshold is

$$\frac{\|r_k\|_1}{\|x_k\|_1} \leq p(n)\epsilon(\|A\|_1 + \|\rho_k B\|_1), \tag{5.4}$$

where ϵ is the machine roundoff unit and we set $p(n) = 10\sqrt{n}$. We note that if $p(n)$ is the maximal number of non-zero entries in each row of the matrices, (5.4) is approximately the size of roundoff errors encountered in computing the residual $r_k = (A - \rho_k B)x_k$ and would be the smallest threshold one could expect.

When `eigifp` terminates with a converged eigenpair, its residual satisfies the termination criterion (5.4). It can be easily checked that (ρ_k, x_k) is an exact eigenvalue and eigenvector of a slightly perturbed problem, i.e.

$$(A + E)x_k = \rho_k(B + F)x_k$$

where

$$\frac{\|E\|_1}{\|A\|_1} \leq p(n)\sqrt{n}\epsilon \quad \text{and} \quad \frac{\|F\|_1}{\|B\|_1} \leq p(n)\sqrt{n}\epsilon.$$

Here a stronger result (with E and F symmetric and the factor $\sqrt{n}$ removed) is obtained if we use the 2-norm instead of the 1-norm, but we have adopted the 1-norm so that the threshold in (5.4) can be computed for large sparse matrices.

5.1.5 Relation to Total Variation Image Deblurring

We note here that the pencil (A, B) problem can be related to the TV image deblurring problem. In the Lagged Diffusivity Fixed Point method, one computes the solution

$$K^T (Ku - u_0) - \lambda Lu = 0$$

or

$$K^T K u = \lambda L u + K^T u_0$$

which we can rewrite as

$$Au = \lambda Lu + f$$

where $A = K^T K$ and $f = K^T u_0$. This form of the problem is called the *nonhomogeneous eigenvalue* problem. The solution of this problem can be found using the eigenpairs of the pencil (A, L) and solving a corresponding constraint on the eigenvector u. However, all the eigenvectors and eigenvalues must be known, which makes a direct utilization of `eigifp` infeasible. In addition, the corresponding eigenpair that we desire in the computation is not clearly understood at this point requiring a calculation to compute all eigenpairs for the nonhomogeneous problem.

5.2 Numerical Comparisons

In this section, we present some numerical comparisons between `eigifp` and some existing programs for computing the smallest eigenvalue of large matrices. The test matrices are a set of symmetric matrices taken from the Harwell-Boing collection [30], as listed in Table I. All the executions were carried out using MATLAB version 6.0 with the most recent patches from MathWorks on a Pentium III Xeon 1.8Ghz with 1GB of RAM.

The performance comparison parameters we considered were the residual of the approximate eigenpair obtained ($\texttt{Res} := \|Ax - \rho_k Bx\|_1 / (\|A\|_1 + |\rho_k| \|B\|_1)$), the number of matrix-vector multiplications, and the CPU time. The CPU time was gathered with on-screen outputs suppressed. Where applicable, it was also important to consider whether it is the *smallest* eigenvalue that had been obtained.

Since these programs have different functionalities, we carried out the testing in two environments. For the first, the input parameters were solely the matrices, assumed no other knowledge about the problem. In this case, `eigifp` was compared with `eigs` of MATLAB 6. For the second, we assumed that an approximate eigenvalue was available, from which a preconditioner was computed and supplied to the programs. Then, `eigifp` was compared to `lobpcg` (version 4.0) [45] and `jdcg` [55]. (Note that `jdcg` is a variation of the Jacobi-Davidson method [68].)

Table II presents the results for the first test, where we compared `eigifp(A,B,1)` with `eigs(A,B,1,'SA')` for computing the algebraically smallest eigenvalue. For most problems, we see that `eigifp` outperformed `eigs` in both matrix-vector multiplications and CPU time. In several cases, `eigifp` also gave a smaller eigenvalue than `eigs`.

Table III presents the results for the second test. Here, we took the smallest eigenvalue λ_1 (as computed by `eigifp`) and perturbed it to arrive at the approximate eigenvalue $\mu = \lambda_1(1 - 0.1 * sgn(\lambda_1))$. This perturbation yielded $\mu < \lambda_1$, which was required by `lobpcg` and `jdcg`, but not by `eigifp`. Using μ, we computed the incomplete Cholesky factorization with the threshold 10^{-3} (by `choinc(A-`μ `B, 1e-3)` in MATLAB). The incomplete Cholesky factor was then supplied as a preconditioner to `eigifp`, `lobpcg` and `jdcg`. Here, the stopping tolerance was set according to (5.4)

In this test, `eigifp` outperformed `jdcg` and was comparable to `lobpcg` in terms of matrix-vector multiplications (MV). In terms of CPU time, it was slightly faster than `lobpcg` but `jdcg` performed best. `eigifp` also gave slightly better results than `jdcg` and `lobpcg` in terms of convergence of residuals. The eigenvalues returned by all three programs, when they converged, were comparable and we omit their listing in the table.

We conclude that `eigifp` is a very competitive program overall. It has the advantage

Table 5.1: Harwell-Boing Test Matrices Used

No.	Matrix	Size	No.	Matrix	Size
1	CAN 1072	1072	10	E40R0000	17281
2	ZENIOS	2873	11	DWT 2680	2680
3	BCSPWR10	5300	12	JAGMESH9	1349
4	BCSSTK13	2003	13	NOS7	729
5	BCSSTK18	11948	14	LSHP3466	3466
6	BCSSTK25	15439	15	PLAT1919	1919
7	BCSSTK27	1224	16	1138 BUS	1138
8	BCSSTK33	8738	17	ERIS1176	1176
9	SSTMODEL	3345	18	S3DKT3M2	90449

that while minimal user input is required it can exploit any extra information available to improve the performance.

Finally, we remark that our test was limited to the smallest eigenvalue only. Since the programs considered here use different mechanisms for computing several eigenvalues, they may perform differently when several eigenvalues are sought. `eigifp` computes several eigenvalues through deflation, but when implemented with preconditioning, the main cost is in computing the initial approximation and then the preconditioner; following this more eigenvalues are typically computed using only a few iterations with the same preconditioner and would represent only a small overhead.

Table 5.2: Res - normalized residual; MV - number of multiplications by A; Prec - number of multiplications by preconditioner; CPU - CPU time (in seconds); λ_1 - eigenvalue obtained.

	EIGS				EIGIFP				
	Res	MV	CPU	λ_1	Res	MV	Prec	CPU	λ_1
1	4e-17	150	0.3	-4.3	1e-15	31	6	1.3	-4.3
2	9e-17	60	0.1	-1.4	2e-15	26	3	0.1	-1.4
3	9e-17	140	0.7	-3.1	3e-17	47	4	0.5	-3.1
4	1e-08	20050	61	1.6e+5	1e-07	4065	0	18	4.3e+4
5	3e-09	119500	1511	2.0	6e-09	24421	0	536	1.8
6	3e-09	154411	6995	1.7e+3	7e-13	33993	16	1633	1.1e-3
7	4e-07	12260	28	1.6e+2	3e-15	3548	9	13	1.4e+2
8	1e-17	210	4	-2.5e1	9e-16	72	7	25	-2.5e1
9	1e-17	90	0.2	-5.6	6e-17	35	4	0.3	-5.6
10	5e-09	172830	4006	-3.1	4e-07	43408	40763	35649	-3.1
11	4e-17	170	0.4	-4.1	5e-16	82	5	0.8	-4.1
12	4e-17	560	0.8	-2.0	2e-14	96	9	0.4	-2.0
13	2e-10	7310	8	1.1e-2	7e-14	114	69	0.6	4.2e-3
14	2e-17	960	3	-2.0	1e-15	140	9	1	-2.0
15	2e-08	19210	41	1.2e-7	5e-08	3989	0	12	4.4e-9
16	2e-08	11400	14	3.5e-3	1e-14	709	10	1	3.5e-3
17	1e-17	90	0.1	-4.9	1e-15	39	4	0.2	-4.9
18	1e-10	904510	119082	3.8e-5	3e-11	181101	0	41930	2.7e-8

Table 5.3: Res - normalized residual; MV - number of multiplications by A; CPU - CPU time (in seconds), err - error encountered.

	LOBPCG			JDCG			EIGIFP		
	Res	MV	CPU	Res	MV	CPU	Res	MV	CPU
1	9e-15	26	0.5	3e-15	35	0.4	1e-15	27	0.4
2	1e-14	15	0.2	4e-15	25	0.1	1e-15	16	0.1
3	5e-15	25	0.9	1e-14	34	0.5	1e-15	26	0.6
4	err	err	err	3e-14	115	2	8e-15	144	3
5	err	err	err	8e-14	74	4	7e-14	63	5
6	2e-14	30879	1568	9e-12	30879	2408	1e-19	1355	276
7	1e-14	77	1	1e-14	96	0.7	3e-15	88	1
8	5e-15	42	6	6e-15	53	7	4e-15	42	7
9	2e-15	19	0.4	4e-15	28	0.3	1e-15	19	0.3
10	2e-12	17282	3523	4e-10	17282	2334	9e-12	17305	3397
11	7e-15	34	0.7	3e-15	44	0.5	1e-15	34	0.6
12	1e-13	1350	15	7e-15	109	0.5	3e-16	271	2
13	5e-15	20	0.1	1e-14	32	0.1	2e-14	19	0.1
14	5e-15	226	6	5e-15	117	2	6e-15	115	2
15	6e-02	1920	50	err	err	err	1e-05	29106	960
16	1e-14	69	0.6	2e-14	70	0.2	6e-14	70	0.4
17	5e-15	13	0.1	7e-15	22	0.1	2e-16	14	0.1
18	6e-08	501	667	1e-07	501	413	4e-09	4119	4943

6 Conclusion

We presented a method for using a reference image in combination with the Chan and Wong [23] TV minimizing functional for blind deconvolution. Using this Semi-Blind method we developed, we allow non-centrosymmteric kernels and images with non-black background images to be computed with fine details intact. We have analyzed the error in computing the kernel function when using this reference image. In particular, the shock filter works well as a reference image since it provides good information on edges and relative position information.

We have presented an extension to the TV functional for the general L^p fidelity term. Using the Euler-Lagrange form, we found solutions using the cases when $p = 1$ and $p = 2$ for corresponding kernel and image functionals. We conclude that using the L^2 norm for kernel recovery and L^1 norm for the image recovery results in the best computed images for both the blind and Semi-Blind methods. We compute the optimal values for the Lagrange multipliers in the functionals for the one dimensional case and showed that utilizing this information is accurate even in a two dimensional setting. In the case of the L^1 fidelity norm, we saw that there is loss of continuity in the residuals as compared to using the L^2 fidelity term.

We presented a modification to Picard's Method that utilizes the discrete data directly in computations, yet keeps the extensibility of the Modified Picard Method. This method was compared to the MPM and showed that it produces the same result minus the error due to approximating the derivatives. The DPM was proven to be equivalent to the forward time difference FD scheme, as well as the Lax-Wendroff scheme for the appropriate degree of the iterate. We showed how to effectively compute the higher degree stability conditions for linear problems.

We developed an black-box implementation of the inverse free preconditioned method by Golub and Ye [40]. The method adaptively chooses the degree of the subspace utilized based on the convergence pattern of the prior approximations to the solution. Appropriate assumptions were made to improve the algorithms permformance. `eigifp` was compared to existing methods and was shown to be competitive to those routines in numerical tests.

REFERENCES

[1] Luis Alvarez and Luis Mazorra. Signal and image restoration using shock filters and anisotropic diffusion. *SIAM Journal on Numerical Analysis*, 31(2):590–605, 1994.

[2] James Baglama, Daniela Calvetti, and Lothar Reichel. Algorithm 827: irbleigs: A MATLAB program for computing a few eigenpairs of a large sparse Hermitian matrix. *ACM Transactions on Mathematical Software*, 29(3):337–348, September 2003.

[3] Zhaojun Bai, James Demmel, Axel Ruhe Jack. Dongarra, and Henk van der Vorst. *Templates for the solution of Algebraic Eigenvalue Problems: A Practical Guide*. SIAM, Philadelphia, 2000.

[4] Leah Bar, Alexander Brook, Nir Sochen, and Nahum Kiryati. Color image deblurring with impulsive noise. *Lecture Notes in Computer Science*, 3752:49–60, 2005.

[5] Leah Bar, Nir Sochen, and Nahum Kiryati. Variational pairing of image segmentation and blind restoration. *Lecture Notes in Computer Science*, 3022:166–177, 2004.

[6] Leah Bar, Nir Sochen, and Nahum Kiryati. Image deblurring in the presence of salt-and-pepper noise. *Lecture Notes in Computer Science*, 3459:107–118, 2005.

[7] Leah Bar, Nir Sochen, and Nahum Kiryati. Semi-blind image restoration via Mumford-Shah regularization. *IEEE Transactions on Image Processing*, 15(2):483–493, 2006.

[8] Marcelo Bertalmio, Gerald Shapiro, Vicent Caselles, and Coloma Ballester. Image inpainting. SIGGRAPH, 2000.

[9] David C. Carothers, G. Edgar Parker, James S. Sochacki, and Paul G. Warne. Some properties of solutions of polynomial systems of differential equations. *Electronic Journal of Differential Equations*, 2005(40):1–17, 2005.

[10] Francine Catt, Pierre-Louis Lions, Jean-Michel Morel, and Tomeu Coll. Image selective smoothing and edge detection by nonlinear diffusion. *SIAM Journal on Numerical Analysis*, 29(1):182–193, 1992.

[11] Antonin Chambolle, Ronald A. DeVore, Namyong Lee, and Bradley J. Lucier. Nonlinear wavelet image processing: variational problems, compression, and noise removal through wavelet shrinkage. *IEEE Transactions on Image Processing*, 7(3):319–335, 1998.

[12] Raymond Chan, Tony F. Chan, Lixin Shen, and Zuowei Shen. Wavelet deblurring algorithms for spatially varying blur from high-resolution image reconstruction. *Linear Algebra and It's Applications*, 366:139–155, 2003.

[13] Tony F. Chan and Selim Esedoglu. Aspects of total variation regularized l^1 function approximation. *SIAM Journal on Applied Mathematics*, 75(5):1817–1837, 2005.

[14] Tony F. Chan, Selim Esedoglu, Frederick Park, and Andy Yip. Recent developments in total variation image restoration. *In "Handbook of Mathematical Models in Computer Vision",Springer Verlag*, 2005.

[15] Tony F. Chan and Sung Ha Kang. Error analysis for image inpainting. Technical report, UCLA, 2004.

[16] Tony F. Chan, Sung Ha Kang, and Jianhong Shen. Euler's elastica and curvature-based inpainting. *SIAM Journal on Applied Mathematics*, 63(2):564–592, 2002.

[17] Tony F. Chan, Stanley Osher, and Jianhong Shen. The digital TV filter and nonlinear denoising. *IEEE Transactions on Image Processing*, 63(2):231–241, 2001.

[18] Tony F. Chan and Jianhong Shen. Mathematical models for local deterministic inpaintings. *SIAM Journal on Applied Mathematics*, 62(3):1019–1043, 2001.

[19] Tony F. Chan and Jianhong Shen. *Image Processing and Analysis - Variational, PDE, wavelet, and stochastic methods*. SIAM, 2005.

[20] Tony F. Chan and Jianhong Shen. Variational image inpainting. *Communications of Pure and Applied Mathematics*, 58:579–619, 2005.

[21] Tony F. Chan and Jianhong Shen. Theory and computation of variational image deblurring. *IMS Lecture Notes*, 2006.

[22] Tony F. Chan and Luminita A. Vese. Active contours without edges. *IEEE Transactions on Image Processing*, 10(2):266 – 277, 2001.

[23] Tony. F. Chan and Chiu-Kwong Wong. Total variation blind deconvolution. *IEEE Transactions on Image Processing*, 7(3):370–275, 1998.

[24] Tony F. Chan, Andy M. Yip, and Frederick Park. Simultaneous total variation image inpainting and blind deconvolution. *International Journal of Imaging Systems and Technology*, 15(1):92–102, 2005.

[25] Ingrid Daubechies and Gerd Teschke. Variational image restoration by means of wavelets: Simultaneous decomposition, deblurring, and denoising. *Applied and Computational Harmonic Analysis*, 19(1):1–16, 2005.

[26] Ronald A. DeVore, Bjorn Jawerth, and Bradley J. Lucier. Image compression through wavelet transform coding. *IEEE Transactions on Information Theory*, 38(2):719 – 746, 1992.

[27] Ronald A. DeVore, Bjorn Jawerth, and Vasil Popov. Compression of wavelet coeffecients. *American Journal of Mathematics*, 114:737 – 785, 1992.

[28] David L. Donoho. De-noising by soft-thresholding. *IEEE Transactions on Information Theory*, 41(3):613–627, 1995.

[29] David L. Donoho and Iain M. Johnstone. Ideal spatial adaptation by wavelet shrinkage. *Biometrika*, 81:425–455, 1994.

[30] I. Duff, R. Grimes, and J. Lewis. Sparse matrix test problems. *ACM Transactions on Mathematical Software*, 1989.

[31] Lawrence Evans. *Partial Differential Equations*, pages 221–233. American Mathematical Society, 1998.

[32] D. Fish, A. Brinicombe, E. Pike, and J. Walker. Blind deconvolution by means of the richardsonlucy algorithm. *Journal of the Optical Society of America A*, 12(1):58–65, 1995.

[33] Diederik Fokkema, Gerard Sleijpen, and H. Van der Vorst. Jacobi-davidson style qr and qz algorithms for the reduction of matrix pencils. *SIAM Journal on Scientific Computing*, 20:94–125, 1999.

[34] B. Roy Freiden. Restoring with maximum likelihood and maximum entropy. *Journal of Optical Society of America*, 62:511–518, 1972.

[35] Haoying Fu, Michael K. Ng, Mila Nikolova, and Jesse L. Barlow. Efficient minimization methods of mixed l^2-l^1 and l^1-l^1 norms for image restoration. *SIAM Journal on Scientific Computing*, 27(6):1881–1902, 2006.

[36] Stewart Geman and Donald Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6:721–741, 1984.

[37] Charles F. Van Loan Gene H. Golub. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, 1983.

[38] Guy Gilboa, Nir Sochen, and Yehoshua Y. Zeevi. Regularized shock filters and complex diffusion. *ECCV 2002, LVCS 2350, Springer-Verlag*, pages 399–413, 2002.

[39] Guy Gilboa, Nir Sochen, and Yehoshua Y. Zeevi. Image enhancement and denoising by complex diffusion processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(8):1020–1036, 2004.

[40] Gene H. Golub and Qiang Ye. An inverse free preconditioned Krylov subspace methods for symmetric generalized eigenvalue problems. *SIAM Journal of Scientific Computation*, 24:312–334, 2002.

[41] B. R. Hunt. The application of constrained least squares estimation to image restoration by digital computer. *IEEE Transactions on Computers*, 22:805, 1973.

[42] Stefan Kindermann, Stanley Osher, and Peter W. Jones. Deblurring and denoising of images by nonlocal functionals. *Multiscale Modeling and Simulation: A SIAM Interdisciplinary Journal*, 4(4):1091–1115, 2005.

[43] Andrew V. Knyazev. Convergence rate estimates for iterative methods for a mesh symmetric eigenvalue problem. *Soviet journal of numerical analysis and mathematical modelling*, 2:371–396, 1987.

[44] Andrew V. Knyazev. Preconditioned eigensolvers - an oxymoron? *Electronic Transactions on Numerical Analysis*, 7:104–123, 1998.

[45] Andrew V. Knyazev. Toward the optimal preconditioned eigensolver: Locally optimal block preconditioned conjugate gradient method. *SIAM Journal of Scientific Computation*, 23:517–541, 2001.

[46] Rich Lehoucq, Danny Sorenson, and Chao Yang. *ARPACK Users' Guides, Solution of Large Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Method*. SIAM, Philadelphia, 1998.

[47] He Lin, Antonio Marquina, and Stanley J. Osher. Blind deconvolution using TV regularization and Bregman iteration. *International Journal of Imaging Systems and Technology*, 15(1):74–83, 2005.

[48] L. B. Lucy. An iterative technique for the rectifications of observed distributions. *Astronomy Journal*, 79:745–754, 1974.

[49] G. Dal Maso, Jean-Michel Morel, and Sergio Solimini. A variation method in image segmentation: existence and approximation results. *Acta Metematica*, 168:89 – 151, 1992.

[50] James Money. A general ODE and PDE solver using Picard's method. MAA Sectional Meetings, 1998.

[51] James Money and Qiang Ye. Algorithm 845: EIGIFP: a MATLAB program for solving large symmetric generalized eigenvalue problems. *ACM Transactions on Mathematical Software*, 31(2):270–279, 2005.

[52] Keith W. Morton and David F. Mayers. *Numerical Solution of Partial Differential Equations*. Cambridge University Press, 1994.

[53] David Mumford and Jayant Shah. Optimal approximation by piecewise smooth functions and associated variation problems. *Communications on Pure and Applied Mathematics*, 42:577 – 685, 1989.

[54] James G. Nagy and Dianne P. O'Leary. Restoring images degraded by spatially variant blur. *SIAM Journal on Scientific Computing*, 19(4):1063–1082, 1998.

[55] Yvan Notay. Combination of Jacobi-Davidson and conjugate gradient for the partial symmetric eigenproblem. *Numerical Linear Algebra with Applications*, 9:21–44, 2002.

[56] Stanley Osher and Leonid Rudin. Feature-oriented image enhancement using shock filters. *SIAM Journal of Numerical Analysis*, 27(4):919–940, 1990.

[57] Stanley Osher, Leonid Rudin, and Emad Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D*, 60:259–268, 1992.

[58] Beresford N. Parlett. *The Symmetric Eigenvalue Problem*. Prentice-Hall, Englewood Cliffs, N.J., 1980.

[59] David L. Phillips. A technique for the numerical solution of certain integral equations of the first kind. *Journal of ACM*, 9(1):84–97, 1962.

[60] Emile Picard. *Traite D'Analyse*, volume 3. Gauthier-Villars, 1922-1928.

[61] C. David Pruett, Joseph W. Rudmin, and Justin M. Lacy. An adaptive N-body algorithm of optimal order. *Journal of Computational Physics*, 187:298–317, 2003.

[62] Alfio Quarteroni, Riccardo Sacco, and Fausto Saleri. *Numerical Mathematics*. Springer, 2000.

[63] Patrick D. Quillen. *Generalizations of an inverse free Krylov subspace method for the symmetric generalized eigenvalue problem*. PhD thesis, University of Kentucky, 2005.

[64] William H. Richardson. Bayesian-based iterative method of image restoration. *Journal of Optical Society of America*, 62:55–59, 1972.

[65] J.W. Rudmin. Application of the Parker-Sochacki method to celestial mechanics. Technical report, James Madison University, 1998.

[66] Yousef Saad. *Numerical Methods for Large Eigenvalue Problems*. Manchester University Press, Manchester, UK, 1992.

[67] Kaleem Siddiqi, Yves B. Lauziere, Allen Tannenbaum, and Steven W. Zuker. Area and length minimizing flows for shape segmentations. *IEEE Transations on Image Processing*, 7:433 – 443, 1998.

[68] Gerard Sleijpen and Henk van der Vorst. A Jacobi-Davidson iteration method for linear eigenvalue problems. *Journal of Matrix Analysis and Applications*, 17:401–425, 1996.

[69] James S. Sochacki and G. Edgar Parker. Implementing the Picard iteration. *Neural, Parallel, and Scientific Computation*, 4:97–112, 1996.

[70] James S. Sochacki and G. Edgar Parker. A Picard-Maclaurin theorem for initial value PDE's. *Abstract and Applied Analysis*, 5(1):47–63, 2000.

[71] Richard P. Stanley. *Enumerative Combinatorics*, volume 1. Cambridge Press, 1997.

[72] David Strong, Peter Blomgren, and Tony F. Chan. Spatially adaptive local feature-driven total variation minimizing image restoration. *Proceedings of the SPIE Annual Meeting*, 3137:222–233, 1997.

[73] David Strong and Tony F. Chan. Relation of regularization parameter and scale in total variation based image denoising. Technical report, UCLA, CAM 96-07.

[74] S. Twomey. On the numerical solution of Fredholm integral equations of the first kind by the inversion of the linear system produced by quadrature. *Journal of ACM*, 10(1):97–101, 1963.

[75] Curtis Vogel. *Computational Methods for Inverse Problems*. SIAM, 2002.

[76] Curtis Vogel and Mary Oman. Iterative methods for total variation denoising. *SIAM Journal on Scientific Computing*, 17(1):227–238, 1996.

[77] Paul G. Warne, Debra A. P. Warne, James S. Sochacki, G. Edgar Parker, and David C. Carothers. Explicit a-priori error bounds and adaptive error control for approximation of nonlinear initial value differential systems. *Computers and Mathematics with Applications*, to appear.

[78] Martin Welk, David Theis, and Joachim Weickert. Variational deblurring of images with uncertain and spatially variant blurs. *Pattern Recognition, Lecture Notes in Computer Science*, 3663:485–492, 2005.

[79] Herbert S. Wilf. *generatingfunctionology*. Academic Press, 2 edition, 1994.

[80] Yu-Li You and Mostafa Kaveh. Anisotropic blind image restoration. *IEEE Transactions on Image Processing*, 8(3), 1999.

[81] Olgierd C. Zienkiewicz, Robert L. Taylor, and Jian Z. Zhu. *The Finite Element Method: Its Basis and Fundamentals*. Butterworth-Heinemann, 6th edition, 2005.

VITA

1. Background

   (a) Date of Birth: October 27, 1975

   (b) Place of Birth: Charlotte, NC

2. Academic Degrees

   (a) B.S., Computer Science(Mathematics minor), James Madison University, 1998.

   (b) M.S., Mathematics, University of Kentucky, 2003.

3. Professional Experience

   (a) Teaching Assistant, Mathematics Department, University of Kentucky, Fall 2001-Spring 2006.

   (b) Research Assistant under Dr. Ren-Cang Li, University of Kentucky, Summer 2003-Spring 2004.

   (c) Research Assistant under Dr. Qiang Ye, University of Kentucky, Summer 2002.

   (d) Software Engineer, BAE Systems, Reston, VA January 1999-August 2001.

   (e) System Administrator, Department of Mathematics, James Madison University, May 1998-January 1999.

4. Publications

   (a) Algorithm 845: EIGIFP: a MATLAB program for solving large symmetric generalized eigenvalue problems. (*with Qiang Ye*) ACM Transactions on Mathematical Software, 2005, 31(2), p.270-279.