University of Louisville
ThinkIR: The University of Louisville's Institutional Repository

Electronic Theses and Dissertations

5-2009

Model extensions and applications in mathematical imaging.

John Marion Cochran
University of Louisville

Follow this and additional works at: https://ir.library.louisville.edu/etd

Recommended Citation

Cochran, John Marion, "Model extensions and applications in mathematical imaging." (2009). *Electronic Theses and Dissertations.* Paper 261.
https://doi.org/10.18297/etd/261

This Doctoral Dissertation is brought to you for free and open access by ThinkIR: The University of Louisville's Institutional Repository. It has been accepted for inclusion in Electronic Theses and Dissertations by an authorized administrator of ThinkIR: The University of Louisville's Institutional Repository. This title appears here courtesy of the author, who has retained all other copyrights. For more information, please contact thinkir@louisville.edu.

MODEL EXTENSIONS AND APPLICATIONS IN MATHEMATICAL IMAGING

By

John Marion Cochran
B.S., Rose-Hulman Institute of Technology, 1997
M.S., Rose-Hulman Institute of Technology, 1999
M.A., University of Louisville, 2008

A Dissertation
Submitted to the Faculty of the
Graduate School of the University of Louisville
in Partial Fulfillment of the Requirements
for the Degree of

Doctor of Philosophy

Department of Mathematics
University of Louisville
Louisville, KY

May 2009

MODEL EXTENSIONS AND APPLICATIONS IN MATHEMATICAL IMAGING

Submitted by

John Marion Cochran

A Dissertation Approved on

May 9, 2009
(Date)

by the Following Reading and Examination Committee:

Yongzhi Steve Xu, Dissertation Director

Bingtuan Li

Thomas Riedel

David Swanson

Shi-Yu Wu

ACKNOWLEDGMENTS

There are always many people involved in an accomplishment such as this dissertation. Above all I thank Jesus Christ my Lord and Savior with whom all things are possible. I also thank my dissertation committee Dr. Yongzhi Xu, Dr. Thomas Riedel, Dr. Bingtuan Li, Dr. David Swanson, and Dr. Shi-Yu Wu. I thank two men who have been my mentors, advisors, and friends for over fifteen years: Dr. Carl F. Abegg and Dr. G. Elton Graves. I thank my brother Morris Franklin Cochran who has put up with me for far too long. Finally, I thank my mother Judy Elizabeth Cochran and my father Morris Ray Cochran who instilled in me an appreciation for education and hard work and who labored constantly to make certain I had all of the opportunities to showcase my knowledge and skills.

ABSTRACT

MODEL EXTENSIONS AND APPLICATIONS IN MATHEMATICAL IMAGING

John Marion Cochran

May 9, 2009

Mathematical imaging consists of many different applications including image segmentation, image classification, and inpainting. This work deals more specifically with image segmentation: the partition of an image into the background and the objects present in the image. The main focus is the active contours without gradient model by Tony Chan and Luminita Vese which deals with fitting a curve imbedded in the plane image. The fitting of the curve comes from an evolutionary partial differential equation.

The dissertation contributes three novel ideas: a linearized version of the active contours without gradient model published in [20]; a new procedure using fourth order fitting terms in place of the second order fitting terms which gives faster segmentation and may be used to provide a good initial condition; a novel way of tracking regions present in bulk data in order to gain an understanding of macroscopic details associated with some physical application.

Results include images showing the accuracy of the segmentation for the methods, a discussion of the choice of initial condition, and discussion of feasibility for the data tracking. These results compare to those obtained with the nonlinear model and serve as a proof-of-concept for further investigation. The dissertation ends with a discussion of future research.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

Mathematical imaging has seen many new models over the past three decades covering many different and diverse application areas including medicine and military science. When a human looks at an image, the brain processes the image in such a way that the person may distinguish objects in the image. The question is this: how can a computer be programmed to "see" an image? In other words, how may we use computers to process images in such a way that objects of interest may be distinguished easily with little or no human interaction? This concept essentially means that we wish to segment a given image such that all objects - or at least objects of interest - are separated from the image background. This chapter serves as an introduction to the dissertation and provides an overview of general techniques. We by no means give a complete review of the models and techniques which are now prevalent in the literature; however, for completeness, we treat the cornerstone models and their limitations.

1.1 Mumford-Shah

The Mumford-Shah functional was first introduced in [39] as a means to partition an image into regions of constant intensity. We let $u_0(x, y)$ represent the image and Ω be a bounded open set (i.e., the domain of the image). Then the idea is to find (u, K) that minimizes

$$F(u, K) = \int_{\Omega - K} (u - u_0)^2 dxdy + \alpha \int_{\Omega - K} |\nabla u|^2 dxdy + \beta \int_K d\sigma \tag{1.1}$$

where K represents the set of discontinuities in the segmentation and the last term represents the length of K. In order for the above to be meaningful (i.e., the length of K to be meaningful) we replace the last term by the $(N-1)$-dimensional Hausdorff measure. The

existence and uniqueness results are already in the literature. See [5] for a good review of the technical details.

Because we must find both the function u and the set K, there are many numerical difficulties which arise from the above representation. There have been several different methods for approximating the numerical solution. The main approaches include using

1. Elliptic functionals [2]
2. Second order singular perturbations [11] (see also [9][13])
3. Nonlocal terms [12]
4. Finite differences [17][28]

We now turn from image partition to object detection within an image.

1.2 Snakes

In 1987, Kass et.al. developed a new model they called "snakes". The name comes from the way that objects are detected. The idea is to evolve curves using energy minimization so that the curves stop on the edges of the image. The way these curves move resemble snakes, hence the name. For a detailed discussion of the model see [5][31].

Let Γ represent the set of edges (i.e., the boundaries of the objects) in the image. Let C_j be C^1 closed curves in $\Re^2$. We define the function $g : \Re \to \Re$ as a detector function such that

1. g is regular monotone decreasing,
2. $g(0) = 1$,
3. $\lim_{s\to\infty} g(s) = 0$.

We assume that the domain of the image, Ω, is bounded and we assume $|\nabla I| \in W^{1,\infty}(\Omega)$ where I represents the intensity of the image. (Recall that $W^{1,\infty}(\Omega)$ is the Sobolev space

defined by $W^{1,\infty}(\Omega) = \{v : \Omega \to \Re : v \in L^\infty(\Omega), Dv \in L^\infty(\Omega)\}$.) Define the set C as $C = \{f : [a,b] \to \Omega, f \text{ piecewise } C^1(\Omega), f(a) = f(b)\}$ where a and b are real.

With the above notational conventions and assumptions, we form the energy integral $J(c)$ as follows

$$J(c) = \int_a^b |c'(q)|^2 dq + \beta \int_a^b |c''(q)|^2 dq + \lambda \int_a^b g^2(|\nabla I(c(q))|)dq \tag{1.2}$$

where $c(q) = \{c_1(q), c_2(q)\}$, $c'(q) = \{c_1'(q), c_2'(q)\}$, and $c''(q) = \{c_1''(q), c_2''(q)\}$. The first two terms of $J(c)$ represent the internal energy while the last term represents the external energy. The problem is to find c which minimizes $J(c)$. The associated Euler-Lagrange equation (see below) is given by

$$\begin{aligned} -c'' + \beta c^{(iv)} + \lambda \nabla F(c) &= 0 \\ c(a) &= c(b) \end{aligned} \tag{1.3}$$

where $F(c) = g^2(|\nabla I(c_1, c_2)|)$.

There are many problems with this method. They include

1. $J(c)$ is not convex which means we can only expect to find a local minimum.
2. $J(c)$ depends upon the parametrization of c in the form of $c(q)$. It may be possible to obtain another solution by using different parametrizations with the same initial data.
3. The model does not handle topological changes.
4. The initial curve must be close to the edges of the desired object for accurate detection.
5. There are also several numerical problems which may develop. See [46] for a complete discussion.

There have been many models introduced to try and solve the above problems. To address the parametrization problems, [23] uses B-splines to model the contours while [3] models the curve evolution using a linked chain of control points.

Cohen [21] defines external forces in terms of pressure and treats the level set curves as balloons. This allows the initial curve to be located further from the edges of the image and still converge to the edges. Xu and Prince [50] introduce a new model using a different approach toward the external force which they call the *gradient vector flow.* The new model gives better convergence and also allows the initial curve to be placed anywhere for appropriate images.

There have also been attempts to solve the problem of topology changes. McInerney et.al. [38] develop a model that does split and merge to accommodate more sophisticated objects. Many other models [16][15][32][37], however, approach the problem from a geometrical point of view similar to that of the Geodesic Active Contour model.

1.3 Geodesic Active Contours

The geodesic active contour model [5][16][15] begins by letting $\beta = 0$ in (1.2). This gives the following

$$J_1(c) = \int_a^b |c'(q)|^2 dq + \lambda \int_a^b g^2(|\nabla I(c(q))|) dq. \tag{1.4}$$

The functional $J_1(c)$ is still not intrinsic (i.e., it depends upon the paremetrization $c(q)$). To solve this problem, define $J_2(c)$ as

$$J_2(c) = 2\sqrt{\lambda} \int_a^b g(|\nabla I(c(q))|)|c'(q)| dq. \tag{1.5}$$

In [6], the authors show that minimizing $J_1(c)$ is equivalent to minimizing $J_2(c)$. Recalling that the curvature is given by

$$\kappa = \text{div}(\frac{\nabla u}{|\nabla u|})$$

and letting $g = g(|\nabla I(c(q))|)$ to simplify notation, the associated Euler-Lagrange equation is

$$c_t = (g\kappa - \nabla g \cdot \vec{N})\vec{N}. \tag{1.6}$$

We may improve the detection of nonconvex objects and increase the speed of convergence [16][15] by adding the term αg to the above model to obtain

$$c_t = (g\kappa - \nabla g \cdot \vec{N} + \alpha g)\vec{N}. \tag{1.7}$$

Using this, the authors develop a level-set representation of the model. This gives the geodesic active contour model

$$u_t = g(|\nabla I|)(\kappa + \alpha)|\nabla u| + \nabla g \cdot \nabla u \tag{1.8}$$

where u is the function whose steady state gives the edge segmentation of the image.

Many other related works follow the same procedure including the independent work by Kichenassamy et.al. in [32][33]. The four works [16][15][32][33] all have similar ideas and begin with energy minimization as seen above. [14],[37], and [36] develop the model based directly upon the level-set approach. In [48], the authors make a change to the "constant inflation term" by relating it to an area minimizing flow.

Drawbacks of the geodesic model [5] include:

1. Interior contours are not detected automatically.

2. When the curve detects an object, it stops since the model is defined in terms of gradient. In other words, large gradients define edges. Once the gradient becomes large, the curve evolution stops.

3. The level-set method assumes closed curves. These closed curves do not allow the detection of certain types of "open" objects.

4. The initial surface is usually set to be the distance function to the level set. It is desireable to maintain the distance function properties and description, but the model does not preserve the distance function. The formal reasons may be found in [8]. This requires a distance function "reconciliation". This procedure is discussed in more detail next.

These problems have been at least partly addressed by Chan and Vese in [18].

1.4 Reconciling the Distance Function

In [29], Gomes and Faugeras give two reasons why the reconciliation is needed.

1. Given any surface S, the distance function, u, of S is uniquely defined. Further, if a function u satisfies the property $|Du| = 1$, then it is the distance function of some curve S up to a constant [4]. Knowing properties of one of the two allows us to determine properties of the other.

2. When computing the derivatives of a function u, we usually take a step size of, say, Δx. This step size will need to be changed according to the behavior of u in each neighborhood. So, if the derivative is large, we need to take smaller steps. If the derivative is small, we may take larger steps. The natural step size for an image is one pixel (i.e., $\Delta x = 1$). This means we need to know about the derivatives of u. If u is a distance function, we know that $|\nabla u| = 1$. So if u remains a distance function, we can be certain that the derivatives remain bounded and we avoid possible numerical problems.

From the practical view, we need to reinitialize at least every 20 iterations of the model [5].

There are two methods used to reconcile the distance function [52]. Both methods concern the solution of the eikonal equation. The difference is the form of the equation. The first is a pde given by

$$\begin{aligned} u_t + sign(v_0)(|Du| - 1) &= 0 \\ u(x,0) &= v_0. \end{aligned} \tag{1.9}$$

The parameter t is a dummy time variable which allows us to treat the above equation as an evolutionary pde. Details including the discretization of the equation may be found in [5]. See also [25] and [26].

The second method deals with solving the stationary pde

$$|Du| = 1 \tag{1.10}$$

and developing efficient numerical schemes to solve the equation. Two of the main methods are the fast marching method [30] [47] [49] [43] [1] and the fast sweeping method [10]

[51] [52]. Fast marching deals with updating the function grid point by grid point as the solution front moves. In [22], Covello and Rodrigue expand the method to include highly distorted grids and randomly located nodes. Fast sweeping updates the solution by sweeping along the grid in alternating diagonal directions. We use the fast sweeping method in the following.

For ease of discussion, we assume a two-dimensional image, although in [52], Zhao presents the necessary framework for multi-dimensional solutions. Let U be a subet of $\Re^2$. Let Γ denote the zero level set. In other words, Γ is the boundary to which we will construct the distance function u in U. We wish to find a solution u of the pde

$$\begin{aligned} |\nabla u| &= 1 \text{ in } U \\ u &= 0 \text{ on } \Gamma. \end{aligned} \tag{1.11}$$

As in [52] we use the Godunov upwind difference scheme [44] with the step size $h = 1$

$$[(u_{i,j} - u_{xmin})^+]^2 + [(u_{i,j} - u_{ymin})^+]^2 = 1 \tag{1.12}$$

for all internal points $i = 2, \ldots, I-1$, $j = 2, \ldots, J-1$ where we take a grid consisting of I points in the x-direction and J points in the y-direction. At the boundaries of the image, we use one-sided finite difference schemes. We also define

$$\begin{aligned} u_{xmin} &= \min(u_{i-1,j}, u_{i+1,j}) \\ u_{ymin} &= \min(u_{i,j-1}, u_{i,j+1}) \end{aligned}$$

and

$$v^+ = \begin{cases} v & \text{if } v > 0 \\ 0 & \text{if } v \leq 0 \end{cases}.$$

Equation (1.12) may be solved explicitly to obtain

$$\bar{u} = \begin{cases} \min(u_{xmin}, u_{ymin}) + 1 & |u_{xmin} - u_{ymin}| \geq 1 \\ \frac{u_{xmin} + u_{ymin} + \sqrt{2 - (u_{xmin} - u_{ymin})^2}}{2} & |u_{xmin} - u_{ymin}| < 1 \end{cases}.$$

The solution scheme to compute the viscosity solution $u(x) \geq 0$ then consists of the following steps.

1. Set Γ.

2. Initialize the function u by determining all grid points on the boundary Γ. These points are given values of zero since they are on the zero level set.

3. Determine the grid points which are near the boundary. These grid points have neighbors which are on or inside Γ. We need to determine the distance from these boundary points to the zero level set. These values will not be updated or modified in any way. They are fixed. All other grid points are set to some large value that must be at least as large as the maximum possible distance expected.

4. Alternate between the four diagonal directions for sweeping and solve Equation (1.12). The updated value for $u_{i,j}$ is the minimum of either $\bar{u}$ or the current value of $u_{i,j}$.

5. Once the maximum error is within a specified tolerance, stop updating and display the results.

The initialization step is the most difficult since we must not only locate Γ but also set the actual distance values for all boundary points. In the following numerical examples, we locate the zero level set by either looking for zero crossings at a grid location or checking for a sign change between grid points. All points where one of these criteria are met are flagged as boundary points. For all boundary points, we set the distance to be zero for any grid location on the zero level set. If a sign change occurs between two grid points, we use linear interpolation to find the approximate location of the zero level set. The values at each boundary point are then set to be the minimum distance to these approximations of Γ.

It is also important to note that for imaging applications, we usually use signed distance functions. In other words, if Γ is the zero level set where the value of u is positive inside Γ and negative outside, we would like to initialize u to be the distance function of Γ but retain the sign of u. This was done by saving the sign of the initial function in

another matrix, reinitializing the function to be a distance function, and then adjusting the sign accordingly.

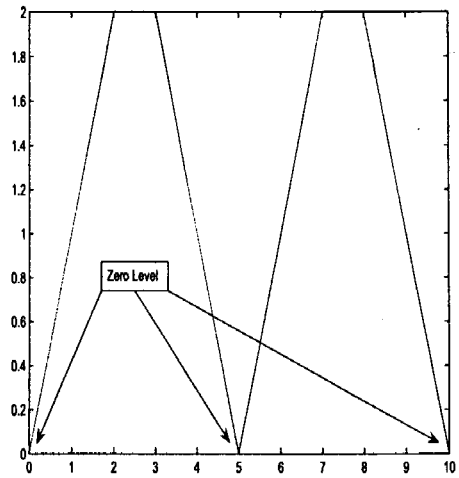

FIGURE 1.1 – One dimensional example of distance function calculation. Program executed for 2 iterations, 0.08 seconds. The level set was defined to be the points $\{0, 5, 10\}$.

All of the following numerical results came from a Windows XP machine, 2 Ghz, 1 Gig RAM, running Matlab. Figure 1.1 shows the distance function calculated using a level set of $\{0, 5, 10\}$. The program swept from left to right and then right to left. With a tolerance of 0.001, the program ran for two iterations and 0.08 seconds. Figure 1.2 shows another distance function for the level set defined by the interval $[4, 6]$.

Figure 1.3 shows a two dimensional distance function contour plot. The zero level set is $\{(42, 42), (42, 84), (84, 42), (84, 84)\}$ and the tolerance is 0.0001. A mesh plot of this solution is provided in Figure 1.4. The image domain is 126×126. Figures 1.5 and 1.6 show a signed distance function contour plot and corresponding mesh for a level set defined by a rectangular bar centered at $(100, 100)$ with a length and width of 30. The image domain is 200×200. Here, we have included a signed distance function where the values are positive inside the rectangle and negative outside.

1.5 Active Contours Without Gradient

The idea behind the active contours model is to evolve a curve C such that the

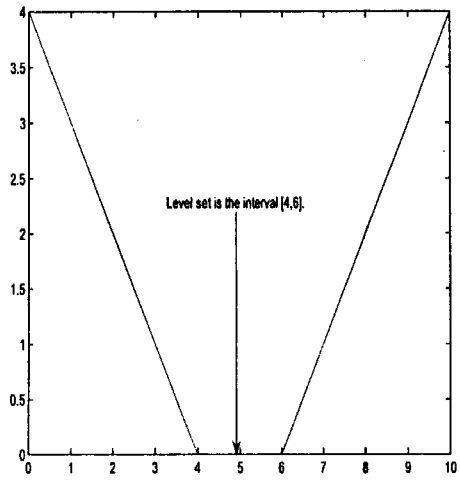

FIGURE 1.2 – One dimensional example of distance function calculation. Program executed for 3 iterations, 0.08 seconds. The level set was defined to be the interval $[4, 6]$.

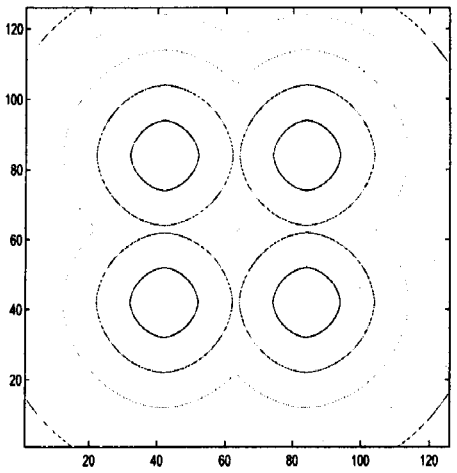

FIGURE 1.3 – Two dimensional example of distance function calculation. The level set is defined to be $\{(42, 42), (42, 84), (84, 42), (84, 84)\}$.

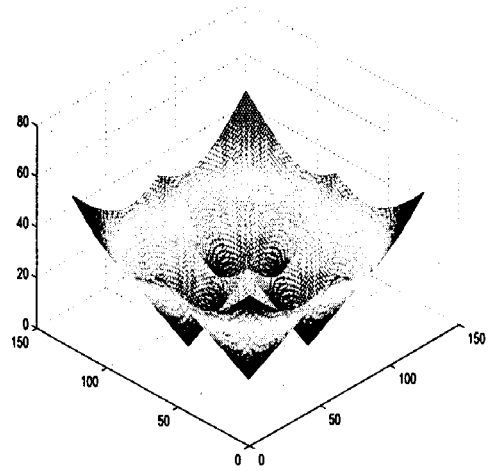

FIGURE 1.4 – Two dimensional example of distance function calculation. The level set is defined to be $\{(42, 42), (42, 84), (84, 42), (84, 84)\}$. This is the mesh plot showing the actual distance values.

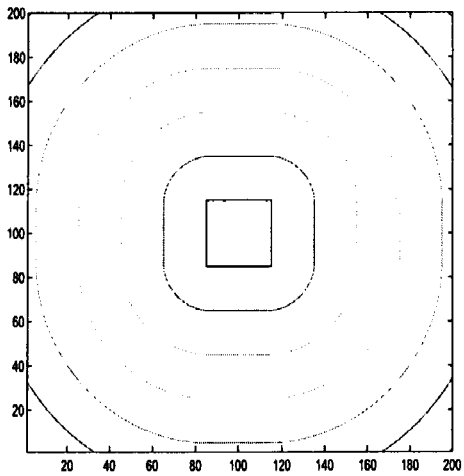

FIGURE 1.5 – Two dimensional example of distance function calculation. The level set is defined to be a rectangular bar centered at $(100, 100)$ with width and length of 30.

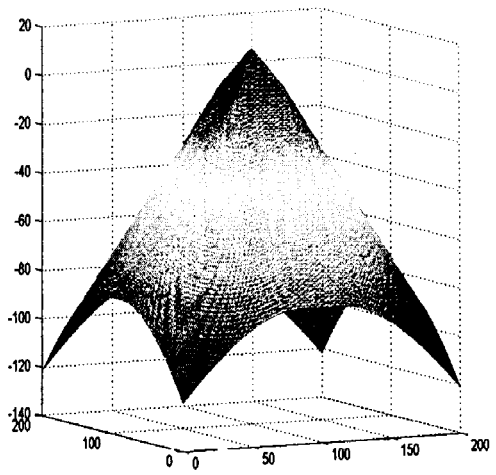

FIGURE 1.6 – Two dimensional example of distance function calculation. This is the mesh plot showing the actual distance values for the rectangular zero level set.

curve stops on the edges of a given image u_0. We summarize the model from [18]. To motivate the model, assume that we have an image which consists of two intensities given by u_0^i and u_0^o which represent the inside and the outside, respectively, of an object to be detected. We form the integral equation

$$F_1(C) + F_2(C) = \int_{inside(C)} |u_0 - c_1|^2 dxdy + \int_{outside(C)} |u_0 - c_2|^2 dxdy$$

where c_1 and c_2 are the average intensities of the image inside C and outside C, respectively. We wish to find the curve C which minimizes this integral equation. It is not difficult to show that the integral equation is minimized when the curve C lies on the edges of the object. In fact, if the object is completely contained inside C, $F_1(C) > 0$ and $F_2(C) \approx 0$. Likewise, if the object is outside C we have $F_1(C) \approx 0$ and $F_2(C) > 0$. If the object is both inside and outside the curve C, then both integrals are positive. Finally, if the curve C lies on the edges of the object, both integrals are almost zero. With this in mind, we define the function F as follows

$$\begin{aligned} F(C, c_1, c_2) &= \mu \text{ length(C)} + \nu \text{ area(inside C)} + \\ &\quad \lambda_1 \int_{inside(C)} |u_0 - c_1|^2 dxdy + \\ &\quad \lambda_2 \int_{outside(C)} |u_0 - c_2|^2 dxdy \end{aligned} \tag{1.13}$$

where ν, μ, λ_1, and λ_2 are constant parameters. We wish to minimize this function. Chan and Vese use a level-set approach to the solution.

Let $C \subset \Omega$ be the zero level set [18] [42] of a Lipschitz function $\phi : \Omega \to \Re$ such that

$$\begin{aligned} C &= \partial\omega = \{(x,y) \in \Omega : \phi(x,y) = 0\} \\ \textit{inside(C)} &= \omega = \{(x,y) \in \Omega : \phi(x,y) > 0\} \\ \textit{outside(C)} &= \Omega \setminus \bar{\omega} = \{(x,y) \in \Omega : \phi(x,y) < 0\}. \end{aligned}$$

Define the Heaviside function H and the Dirac function δ_0 (in the sense of distributions) as

$$\begin{aligned} H(x) &= \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases} \\ \delta_0(x) &= \frac{d}{dx} H(x). \end{aligned}$$

We may now replace C by the function ϕ as follows

$$\begin{aligned} \text{length(C)} &= \int_{\partial\{\phi=0\}} |\nabla\phi| dxdy \\ &= \int_\Omega \delta_0(\phi)|\nabla\phi| dxdy \\ \text{area(inside(C))} &= \int_{\{\phi>0\}} dxdy \\ &= \int_\Omega H(\phi) dxdy \\ \int_{\textit{inside(C)}} |u_0 - c_1|^2 dxdy &= \int_\Omega |u_0 - c_1|^2 H(\phi) dxdy \\ \int_{\textit{outside(C)}} |u_0 - c_2|^2 dxdy &= \int_\Omega |u_0 - c_2|^2 (1 - H(\phi)) dxdy. \end{aligned}$$

Finally, we have the model equation given by

$$\begin{aligned} F(\phi, c_1, c_2) &= \mu \int_\Omega \delta_0(\phi)|\nabla\phi| dxdy + \nu \int_\Omega H(\phi) dxdy + \\ &\quad \lambda_1 \int_\Omega |u_0 - c_1|^2 H(\phi) dxdy + \\ &\quad \lambda_2 \int_\Omega |u_0 - c_2|^2 (1 - H(\phi)) dxdy. \end{aligned} \tag{1.14}$$

Minimizing F is equivalent to minimizing G

$$\begin{aligned} G(\phi, c_1, c_2) &= \mu \int_\Omega \delta_0(\phi)|\nabla\phi| dxdy + \nu \int_\Omega H(\phi) dxdy + \\ &\lambda_1 \int_\Omega (u_0 - c_1)^2 H(\phi) dxdy + \\ &\lambda_2 \int_\Omega (u_0 - c_2)^2 (1 - H(\phi)) dxdy. \end{aligned} \tag{1.15}$$

If we hold ϕ constant and minimize F with respect to c_1 and c_2 we have

$$\begin{aligned} \frac{d}{dc_1} G(\phi, c_1, c_2) &= 0 = -2\lambda_1 \int_\Omega (u_0 - c_1) H(\phi) dxdy \\ c_1(\phi) &= \frac{\int_\Omega u_0 H(\phi) dxdy}{\int_\Omega H(\phi) dxdy} \\ \frac{d}{dc_2} G(\phi, c_1, c_2) &= 0 = -2\lambda_2 \int_\Omega (u_0 - c_2)(1 - H(\phi)) dxdy \\ c_2(\phi) &= \frac{\int_\Omega u_0 (1 - H(\phi)) dxdy}{\int_\Omega (1 - H(\phi)) dxdy}. \end{aligned}$$

Chan and Vese have used the regularized version of $H(x)$ and $\delta_0(x)$ given by

$$\begin{aligned} H_{2\epsilon}(x) &= \frac{1}{2}(1 + \frac{2}{\pi} \arctan(\frac{x}{\epsilon}) \\ \delta_{2\epsilon}(x) &= \frac{d}{dx} H_{2\epsilon}(x). \end{aligned}$$

Letting F_ϵ denote the associated regularized F gives the following model

$$\begin{aligned} F_\epsilon(\phi, c_1, c_2) &= \mu \int_\Omega \delta_{2\epsilon}(\phi)|\nabla\phi| dxdy + \nu \int_\Omega H_{2\epsilon}(\phi) dxdy + \\ &\lambda_1 \int_\Omega |u_0 - c_1|^2 H_{2\epsilon}(\phi) dxdy + \\ &\lambda_2 \int_\Omega |u_0 - c_2|^2 (1 - H_{2\epsilon}(\phi)) dxdy. \end{aligned} \tag{1.16}$$

We may now develop a pde model using the Euler-Lagrange equation.

1.6 Euler-Lagrange Description of Model

A full treatment of the Euler-Lagrange equation of the model follows. We adopt the notation and procedure presented in [24]. Define the Lagrangian as $L : \Re^n \times \Re \times \Re^n \to \Re$ such that $L = L(p, w, x)$. Define the integral functional $I[u]$ as

$$I[u] = \int L(Du, u, x) dx. \tag{1.17}$$

Let us now assume that the function $u : U \to \Re$ is the minimizer of $I[u]$ for $U \subset \Re^n$. For any function $v : U \to \Re$ for $v \in C_c^\infty(U)$ we define $i(\tau)$ as

$$i(\tau) = I[u + \tau v] = \int L(Du + \tau Dv, u + \tau v, x)dx. \tag{1.18}$$

We will now minimize $i(\tau)$ by recognizing that $i'(0) = 0$ since u minimizes $I[u]$. Differentiating with respect to τ gives

$$i'(\tau) = \int \sum_{i=1}^{n} L_{p_i}(Du + \tau Dv, u + \tau v, x)v_{x_i} + L_w(Du + \tau Dv, u + \tau v, x)vdx. \tag{1.19}$$

Integrating the first term by parts and factoring gives

$$i'(\tau) = \int (-\sum_{i=1}^{n}(L_{p_i}(Du + \tau Dv, u + \tau v, x))_{x_i} + L_w(Du + \tau Dv, u + \tau v, x))vdx. \tag{1.20}$$

Evaluating at $\tau = 0$ gives

$$i'(0) = 0 = \int (-\sum_{i=1}^{n}(L_{p_i}(Du, u, x))_{x_i} + L_w(Du, u, x))vdx. \tag{1.21}$$

This equation holds for any function $v \in C_c^\infty(U)$; hence, we may write

$$-\sum_{i=1}^{n}(L_{p_i}(Du, u, x))_{x_i} + L_w(Du, u, x) = 0. \tag{1.22}$$

In the case of the model equation,

$$\begin{aligned} L(p, w, x) &= \mu\delta_{2\epsilon}(w)|p| + \nu H_{2\epsilon}(w) + \lambda_1|u_0 - c_1|^2 H_{2\epsilon}(w) + \\ &\quad \lambda_2|u_0 - c_2|^2(1 - H_{2\epsilon}(w)). \end{aligned} \tag{1.23}$$

Differentiating the above with respect to w and p_i gives the following

$$\begin{aligned} L_{p_i} &= \mu\delta_{2\epsilon}(w)\frac{p_i}{|p|} \\ L_w &= \mu|p|\delta'_{2\epsilon}(w) + \nu\delta_{2\epsilon}(w) + \lambda_1|u_0 - c_1|^2\delta_{2\epsilon}(w) - \lambda_2|u_0 - c_2|^2\delta_{2\epsilon}(w). \end{aligned}$$

Substituting into equation (1.22) and replacing $p = \nabla\phi$ and $w = \phi$ gives

$$\begin{aligned} 0 &= -\sum_{i=1}^{n}(\mu\delta_{2\epsilon}(\phi)\frac{\phi_{x_i}}{|\nabla\phi|})_{x_i} + \mu|\nabla\phi|\delta'_{2\epsilon}(\phi) + \\ &\quad \nu\delta_{2\epsilon}(\phi) + \lambda_1|u_0 - c_1|^2\delta_{2\epsilon}(\phi) - \lambda_2|u_0 - c_2|^2\delta_{2\epsilon}(\phi). \end{aligned}$$

Simplifying the above gives

$$\delta_{2\epsilon}(\phi)(-\sum_{i=1}^{n}(\mu(\frac{\phi_{x_i}}{|\nabla\phi|})_{x_i})+\nu+\lambda_1|u_0-c_1|^2-\lambda_2|u_0-c_2|^2)=0$$

and recognizing that the first term may be written using the divergence, we have

$$\delta_{2\epsilon}(\phi)(-\mu\text{div}(\frac{\nabla\phi}{|\nabla\phi|})+\nu+\lambda_1|u_0-c_1|^2-\lambda_2|u_0-c_2|^2)=0.$$

If we now assign a parameter t to allow us to use a time dependent PDE, we may write this equation in the final form

$$\begin{aligned}\frac{\partial\phi}{\partial t} &= \delta_{2\epsilon}(\phi)(\mu\text{div}(\frac{\nabla\phi}{|\nabla\phi|})-\nu-\lambda_1(u_0-c_1)^2+\lambda_2(u_0-c_2)^2)\\ \phi(0,x) &= \phi_0(x)\\ \frac{\partial\phi}{\partial\vec{n}} &= 0\end{aligned} \tag{1.24}$$

where $\vec{n}$ denotes the unit normal vector to the boundary of the image.

1.7 Organization of Dissertation

The remaining chapters are divided as follows.

1. Chapter 2 contains an analysis of a linearized model of active contours without gradient. We begin with the linearization, specify a possible numerical scheme, and then prove the convergence of the numerical scheme. Results include numerical examples for simple images. Results from this chapter have already been published in [20].

2. Chapter 3 continues the treatment of the Chan-Vese model by motivating a new procedure designed to aid in convergence. The material also considers numerical examples for both the model and the modified procedure.

3. Chapter 4 introduces a new application. The application involves the tracking of regions of interest for physical models and opens with reasons why such a treatment may be of use.

4. Chapter 5 completes the dissertation with ideas for future research.

The Appendix contains the Matlab code used for the numerical results.

CHAPTER 2
LINEARIZATION OF CHAN-VESE MODEL

2.1 Linearized Model

To better understand the dynamics behind the Chan-Vese model, we consider a linearized form. We consider first the divergence term given by

$$\phi_t = div(\frac{\nabla\phi}{|\nabla\phi|}). \tag{2.1}$$

Using the following substitutions we may simplify notation.

$$\begin{aligned} \phi_x &= s \\ \phi_y &= t \\ \phi_{xx} &= u \\ \phi_{yy} &= v \\ \phi_{xy} &= w \\ \phi_{yx} &= z. \end{aligned}$$

The divergence may then be written as a function of these variables as

$$g(s,t,u,v,w,z) = \frac{u+v}{\sqrt{s^2+t^2}} - \frac{1}{\sqrt{s^2+t^2}}(s^2u + stw + stz + t^2v).$$

The idea is now to linearize the divergence term using the standard Taylor's expansion. We will truncate the second-order and higher terms and linearize around the initial condition

$\phi(x,y,0) = \phi^0(x,y)$. The derivatives are then given by

$$\begin{aligned}
g_s &= -\frac{(u+v)s}{(s^2+t^2)^{3/2}} + \frac{3s}{(s^2+t^2)^{5/2}}(s^2u + stw + stz + t^2v) - \\
&\quad \frac{1}{(s^2+t^2)^{3/2}}(2su + tw + tz) \\
g_t &= -\frac{(u+v)t}{(s^2+t^2)^{3/2}} + \frac{3t}{(s^2+t^2)^{5/2}}(s^2u + stw + stz + t^2v) - \\
&\quad \frac{1}{(s^2+t^2)^{3/2}}(sw + sz + 2tv) \\
g_u &= -\frac{1}{(s^2+t^2)^{1/2}} - \frac{s^2}{(s^2+t^2)^{3/2}} \\
g_v &= -\frac{1}{(s^2+t^2)^{1/2}} - \frac{t^2}{(s^2+t^2)^{3/2}} \\
g_w &= -\frac{st}{(s^2+t^2)^{3/2}} \\
g_z &= -\frac{st}{(s^2+t^2)^{3/2}}.
\end{aligned}$$

Substituting the initial condition into the Taylor's series and recognizing that

$$\nabla\phi^0 \circ D^2\phi^0\nabla\phi^0 = (\phi^0_x)^2\phi^0_{xx} + \phi^0_x\phi^0_y\phi^0_{xy} + \phi^0_x\phi^0_y\phi^0_{yx} + (\phi^0_y)^2\phi^0_{yy}$$

(where $D^2\phi^0$ represents the Hessian matrix) we can then write the entire linearization as

$$\begin{aligned}
div(\frac{\nabla\phi}{|\nabla\phi|}) &= [-\frac{\triangle\phi^0\phi^0_x}{|\nabla\phi^0|^3} + \frac{3\phi^0_x}{|\nabla\phi^0|^5}(\nabla\phi^0 \circ D^2\phi^0\nabla\phi^0) - \\
&\quad \frac{2\phi^0_x\phi^0_{xx} + \phi^0_y\phi^0_{xy} + \phi^0_y\phi^0_{yx}}{|\nabla\phi^0|^3}](\phi_x - \phi^0_x) + \\
&\quad [-\frac{\triangle\phi^0\phi^0_y}{|\nabla\phi^0|^3} + \frac{3\phi^0_y}{|\nabla\phi^0|^5}(\nabla\phi^0 \circ D^2\phi^0\nabla\phi^0) - \\
&\quad \frac{\phi^0_x\phi^0_{xy} + \phi^0_x\phi^0_{yx} + 2\phi^0_y\phi^0_{yy}}{|\nabla\phi^0|^3}](\phi_y - \phi^0_y) + \\
&\quad [-\frac{1}{|\nabla\phi^0|} - \frac{(\phi^0_x)^2}{|\nabla\phi^0|^3}](\phi_{xx} - \phi^0_{xx}) + \\
&\quad [-\frac{1}{|\nabla\phi^0|} - \frac{(\phi^0_y)^2}{|\nabla\phi^0|^3}](\phi_{yy} - \phi^0_{yy}) + \\
&\quad [-\frac{\phi^0_x\phi^0_y}{|\nabla\phi^0|^3}](\phi_{xy} - \phi^0_{xy}) + \\
&\quad [-\frac{\phi^0_x\phi^0_y}{|\nabla\phi^0|^3}](\phi_{yx} - \phi^0_{yx}).
\end{aligned}$$

We may rewrite the above into a slightly more compact form as

$$div(\frac{\nabla\phi}{|\nabla\phi|}) = \vec{P}(x,y) \circ \nabla\phi - \frac{\triangle\phi}{|\nabla\phi^0|} - \frac{\nabla\phi^0 \circ D^2\phi\nabla\phi^0}{|\nabla\phi^0|^3} + Q(x,y)$$

where

$$\begin{aligned}\vec{P}(x,y) &= -\frac{\triangle\phi^0\nabla\phi^0}{|\nabla\phi^0|^3} + \frac{3(\nabla\phi^0 \circ D^2\phi^0\nabla\phi^0)}{|\nabla\phi^0|^5}\nabla\phi^0 - \\ &\quad \frac{1}{|\nabla\phi^0|^3}(2\phi^0_x\phi^0_{xx} + \phi^0_y\phi^0_{xy} + \phi^0_y\phi^0_{yx}, \phi^0_x\phi^0_{xy} + \phi^0_x\phi^0_{yx} + 2\phi^0_y\phi^0_{yy}) \\ Q(x,y) &= \frac{\triangle\phi^0}{|\nabla\phi^0|} - \vec{P}(x,y) \circ \nabla\phi^0.\end{aligned}$$

The linear model - with all terms - is then given by

$$\begin{aligned}\phi_t &= \mu(\vec{P}(x,y) \circ \nabla\phi - \frac{\triangle\phi}{|\nabla\phi^0|} - \frac{\nabla\phi^0 \circ D^2\phi\nabla\phi^0}{|\nabla\phi^0|^3} \\ &\quad + Q(x,y)) - \lambda_1(\vec{u}_0 - c_1(\vec{\phi}))^2 + \lambda_2(\vec{u}_0 - c_2(\vec{\phi}))^2. \qquad (2.2)\end{aligned}$$

Notice we have not included the term $\delta_{2\epsilon}(\phi)$. The reason for this is discussed below when we analyze the full generalized numerical scheme. The linearization will be valid for very simple images and for level set functions which do not significantly change. We will use this linearization to better understand the convergence of the Chan-Vese model. The discretization of this linear model is discussed next.

2.2 Discretization

We will use the standard notation for the discretization of the derivatives. We also assume that the step sizes are $h = k = 1$ where h corresponds to the x coordinate and k corresponds to the y coordinate. In order to simplify the notation, we note that the coefficients in the above linearization depend upon the specific values of x and y - and, consequently, they will vary according to the grid used - but we do not specifically write

this dependence. Recalling that $\phi_{i,j} = \phi(jh, ik)$, the finite differences are then given by

$$\begin{aligned}
\phi_x &= \frac{\phi_{i,j+1} - \phi_{i,j-1}}{2h} \\
\phi_y &= \frac{\phi_{i+1,j} - \phi_{i-1,j}}{2k} \\
\phi_{xx} &= \frac{\phi_{i,j+1} - 2\phi_{i,j} + \phi_{i,j-1}}{h^2} \\
\phi_{yy} &= \frac{\phi_{i+1,j} - 2\phi_{i,j} + \phi_{i-1,j}}{k^2} \\
\phi_{xy} &= \frac{1}{2k}\left(\frac{\phi_{i+1,j+1} - \phi_{i+1,j-1}}{2h} - \frac{\phi_{i-1,j+1} - \phi_{i-1,j-1}}{2h}\right) \\
\phi_{yx} &= \frac{1}{2h}\left(\frac{\phi_{i+1,j+1} - \phi_{i-1,j+1}}{2k} - \frac{\phi_{i+1,j-1} - \phi_{i-1,j-1}}{2k}\right)
\end{aligned}$$

where, as we would expect, the last two are equivalent. Define the coefficients $A_{\circ,\circ}$ as

$$\begin{aligned}
A_{00} &= -\frac{\phi_x^0 \phi_y^0}{2hk} \\
A_{01} &= -\left(\frac{P_2}{2k} + \frac{1 + (\phi_y^0)^2}{k^2}\right) \\
A_{02} &= \frac{\phi_x^0 \phi_y^0}{2hk} \\
A_{10} &= -\left(\frac{P_1}{2h} + \frac{1 + (\phi_x^0)^2}{h^2}\right) \\
A_{11} &= -2\left(\frac{1 + (\phi_x^0)^2}{h^2} + \frac{1 + (\phi_y^0)^2}{k^2}\right) \\
A_{12} &= \frac{P_1}{2h} - \frac{1 + (\phi_x^0)^2}{h^2} \\
A_{20} &= \frac{\phi_x^0 \phi_y^0}{2hk} \\
A_{21} &= \frac{P_2}{2k} - \frac{1 + (\phi_y^0)^2}{k^2} \\
A_{22} &= -\frac{\phi_x^0 \phi_y^0}{2hk}.
\end{aligned}$$

With these definitions and some simplification, the numerical scheme may be written as

$$\begin{bmatrix} A_{00} & A_{01} & A_{02} \\ A_{10} & A_{11} & A_{12} \\ A_{20} & A_{21} & A_{22} \end{bmatrix} \phi_{i,j} + Q_{i,j}.$$

This is a graphical representation of how to calculate the divergence on a grid point by using an eight point scheme. Relabeling the grid points correctly produces a tri-diagonal matrix which corresponds to the above scheme. The solution will then be a vector containing these grid points as the elements.

One approach to proving the stability and also convergence of the scheme is to use the semi-discrete form. The idea is to discretize the spatial components - as given above - and treat the grid as a system of continuous functions of time. This will produce a system of ordinary differential equations of the form

$$\frac{d\vec{u}}{dt} = A\vec{u} + \vec{b}.$$

An equilibrium point of this system will correspond to a steady-state numerical solution of the partial differential equation. Stability of the equilibrium point(s) may then be established using the standard Lyapanov theory. To this end, we now assume that the grid used above has been relabeled in a reasonable way and we write all components of the linearized model including the fitting terms from the Chan-Vese model. This gives the following system of odes

$$\frac{d\vec{\phi}}{dt} = A\vec{\phi} + \vec{Q} - \lambda_1(\vec{u}_0 - c_1(\vec{\phi}))^2 + \lambda_2(\vec{u}_0 - c_2(\vec{\phi}))^2 \tag{2.3}$$

where $\vec{Q}$ and $\vec{u}_0$ represent the function Q and the image u_0 evaluated at the corresponding grid points contained in the vector $\vec{\phi}$.

2.2.1 Numerical Example

Define the image as a 5×5 gray-scale with a white pixel having intensity of 100 in the center of the domain. With the standard boundary conditions as given by Chan-Vese, we will expect nine grid points which will need to be updated according to the numerical scheme proposed above. We choose the initial level-set function ϕ^0 to be the distance function with the zero-level set at the boundary of the image. See Figure 2.1. Due to the nature of this particular image and since we are using the linearized model, the values for the constants c_1 and c_2 will not significantly change for each iteration of the numerical scheme. Thus, we will assume that the constants will be $c_1 = 100$ and $c_2 = 0$. Also, note that we use $\mu = 1$ in the calculations and we have dropped the dependence of the term $\delta(\phi)$. We will include a discussion of this later when we treat the more general system.

ϕ_1 $\phi^0=0$ $u^0=0$	ϕ_2 $\phi^0=0$ $u^0=0$	ϕ_3 $\phi^0=0$ $u^0=0$	ϕ_4 $\phi^0=0$ $u^0=0$	ϕ_5 $\phi^0=0$ $u^0=0$
ϕ_6 $\phi^0=0$ $u^0=0$	ϕ_7 $\phi^0=1$ $u^0=0$	ϕ_8 $\phi^0=1$ $u^0=0$	ϕ_9 $\phi^0=1$ $u^0=0$	ϕ_{10} $\phi^0=0$ $u^0=0$
ϕ_{11} $\phi^0=0$ $u^0=0$	ϕ_{12} $\phi^0=1$ $u^0=0$	ϕ_{13} $\phi^0=2$ $u^0=100$	ϕ_{14} $\phi^0=1$ $u^0=0$	ϕ_{15} $\phi^0=0$ $u^0=0$
ϕ_{16} $\phi^0=0$ $u^0=0$	ϕ_{17} $\phi^0=1$ $u^0=0$	ϕ_{18} $\phi^0=1$ $u^0=0$	ϕ_{19} $\phi^0=1$ $u^0=0$	ϕ_{20} $\phi^0=0$ $u^0=0$
ϕ_{21} $\phi^0=0$ $u^0=0$	ϕ_{22} $\phi^0=0$ $u^0=0$	ϕ_{23} $\phi^0=0$ $u^0=0$	ϕ_{24} $\phi^0=0$ $u^0=0$	ϕ_{25} $\phi^0=0$ $u^0=0$

FIGURE 2.1 – The values of the vector $\vec{\phi}$, the image, and the level set function for the numerical example of linearized convergence.

With these assumptions and using Matlab for the numerical computation, we define A and Q as

$$\begin{bmatrix}
-8.6250 & -0.5625 & 0 & -0.5625 & -0.1250 & 0 & 0 & 0 & 0 \\
-1.0000 & -8.0000 & -1.0000 & 0 & -2.0000 & 0 & 0 & 0 & 0 \\
0 & -0.5625 & -8.6250 & 0 & -0.1250 & -0.5625 & 0 & 0 & 0 \\
-1.0000 & 0 & 0 & -8.0000 & -2.0000 & 0 & -1.0000 & 0 & 0 \\
0 & -1.0000 & 0 & -1.0000 & -6.0000 & -1.0000 & 0 & -1.0000 & 0 \\
0 & 0 & -1.0000 & 0 & -2.0000 & -8.0000 & 0 & 0 & -1.0000 \\
0 & 0 & 0 & -1.6875 & -0.1250 & 0 & -9.7500 & -0.5625 & 0 \\
0 & 0 & 0 & 0 & -2.0000 & 0 & -1.0000 & -8.0000 & -1.0000 \\
0 & 0 & 0 & 0 & -0.1250 & -1.6875 & 0 & -0.5625 & -9.7500
\end{bmatrix}$$

$$Q = \begin{bmatrix}
-3.1250 \\
0 \\
-3.1250 \\
0 \\
-4.0000 \\
0 \\
-3.1250 \\
0 \\
-3.1250
\end{bmatrix}.$$

In this case, the matrix A is invertible. The equilibrium point may then be found by setting the time derivatives to zero to obtain

$$\begin{aligned} 0 &= A\vec{\phi} + \vec{Q} - (\vec{u}_0 - c_1(\vec{\phi}))^2 + (\vec{u}_0 - c_2(\vec{\phi}))^2 \\ \vec{\phi} &= A^{-1}(-\vec{Q} + (\vec{u}_0 - c_1(\vec{\phi}))^2 - (\vec{u}_0 - c_2(\vec{\phi}))^2) \\ \vec{\phi} &= 1000 \begin{bmatrix} -0.9745 \\ -1.7154 \\ -0.9745 \\ -1.7554 \\ 2.8363 \\ -1.7554 \\ -0.6549 \\ -1.7953 \\ -0.6549 \end{bmatrix}. \end{aligned}$$

Note that this equilbrium point corresponds to a steady-state solution of the linear pde. In this case, the segmentation is very good since the level-set would be surrounding the middle pixel (i.e., the "object" present in the image).

Since A is invertible, we may rewrite the semi-discrete form as

$$\frac{d\vec{\phi}}{dt} = A(\vec{\phi} + A^{-1}(\vec{Q} - \lambda_1(\vec{u}_0 - c_1(\vec{\phi}))^2 + \lambda_2(\vec{u}_0 - c_2(\vec{\phi}))^2)).$$

The quantity in parentheses is a constant vector and so we define the vector function $\vec{\psi}$ as

$$\vec{\psi} = \vec{\phi} + A^{-1}(\vec{Q} - \lambda_1(\vec{u}_0 - c_1(\vec{\phi}))^2 + \lambda_2(\vec{u}_0 - c_2(\vec{\phi}))^2.$$

The steady-state values will then be shifted by this same constant vector. The reason for doing this is so that we may use properties of the matrix A to show that the Lyapanov function is strictly negative. The semi-discrete form we will now work with is given by

$$\frac{d\vec{\psi}}{dt} = A\vec{\psi}.$$

Define the function $L : \Re^9 \to \Re$ as

$$L(\vec{\psi}) = |\vec{\psi} - \vec{\psi}_{SS}|^2$$

where ψ_{SS} is the shifted equilibrium point found above. Note that L is just the square of the standard Euclidean distance from the equilibrium point in $\Re^9$. This function is a Lyapanov function since it satisfies

$$L(\vec{\psi}_{SS}) = 0$$

and $L(\vec{\psi}) > 0$ for all $\vec{\psi} \neq \vec{\psi}_{SS}$. We also recall that $A\vec{\psi}^{SS} = 0$. Then we may write

$$\frac{d\vec{\psi}}{dt} = \frac{d(\vec{\psi} - \vec{\psi}^{SS})}{dt} = A(\vec{\psi} - \vec{\psi}^{SS}).$$

Differentiating L with respect to time gives

$$\begin{aligned}
\frac{dL}{dt} &= 2\Sigma_{i=1}^{9}(\psi_i - \psi_{SS}^i)\frac{d\psi_i}{dt} \\
&= 2(\vec{\psi} - \vec{\psi}_{SS}) \circ \frac{d\vec{\psi}}{dt} \\
&= 2(\vec{\psi} - \vec{\psi}_{SS}) \circ \frac{d(\vec{\psi} - \vec{\psi}^{SS})}{dt} \\
&= 2(\vec{\psi} - \vec{\psi}_{SS}) \circ A(\vec{\psi} - \vec{\psi}^{SS}).
\end{aligned}$$

To simplify notation, let $\vec{X} = \vec{\psi} - \vec{\psi}^{SS}$. Then, we have

$$\frac{dL}{dt} = \vec{X} \circ A\vec{X}.$$

Calculating the eigenvalues of A reveals that all eigenvalues are negative. This implies that the quadratic form given above is negative definite; hence, $\vec{X} \circ A\vec{X} < 0$ for all $\vec{X}$. This in turn implies that $\frac{dL}{dt} < 0$ for all $\vec{X}$. Since $\frac{dL}{dt}$ is strictly less than zero for any choice of $\vec{X}$ and, hence, any choice of $\vec{\phi}$, we conclude by classical Lypanov theory that the equilibrium point found above is asymptotically stable. This means that the numerical scheme will converge to ϕ_{SS} given above. This proves both stability and convergence of the linearized scheme for this particular initial condition. In this case, the steady-state solution has given a good segmentation of the "image". Note that we cannot guarantee that a convergent numerical solution will give a good segmentation for every image.

2.2.2 Comparison of Initial Conditions

The semi-discrete approach allows us to actually solve the system of odes; therefore, we can obtain continuous functions which represent the actual value of the grid points as

functions of t. This allows us to compare the performance of various initial conditions. We have chosen six possible initial conditions which represent typical choices for the nonlinear system. By treating c_1 and c_2 as constants throughout the evolution (i.e., we do not update the values for each time increment), the linear system does not depend upon the actual values of ϕ at the grid points. Instead, the derivatives determine both the matrix A and the vector $\vec{Q}$. This allows us to consider a seventh initial condition which gives a more general result. The inital conditions are given in matrix form by the following.

$$IC_1 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 2 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$IC_2 = \begin{bmatrix} -1 & -1 & -1 & -1 & -1 \\ -1 & 0 & 0 & 0 & -1 \\ -1 & 0 & 1 & 0 & -1 \\ -1 & 0 & 0 & 0 & -1 \\ -1 & -1 & -1 & -1 & -1 \end{bmatrix}$$

$$IC_3 = \begin{bmatrix} -\sqrt{2} & -1 & -\sqrt{2} & -1 & -\sqrt{2} \\ -1 & 0 & -1 & 0 & -1 \\ -\sqrt{2} & -1 & -\sqrt{2} & -1 & -\sqrt{2} \\ -1 & 0 & -1 & 0 & -1 \\ -\sqrt{2} & -1 & -\sqrt{2} & -1 & -\sqrt{2} \end{bmatrix}$$

$$IC_4 = \begin{bmatrix} -1 & 0 & -1 & 0 & -1 \\ 0 & 1 & 0 & 1 & 0 \\ -1 & 0 & -1 & 0 & -1 \\ 0 & 1 & 0 & 1 & 0 \\ -1 & 0 & -1 & 0 & -1 \end{bmatrix}$$

$$IC_5 = \begin{bmatrix} 1 & 0 & -1 & 0 & 1 \\ 0 & -1 & -\frac{1}{2} & -1 & 0 \\ -1 & -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & -1 \\ 0 & -1 & -\frac{1}{2} & -1 & 0 \\ 1 & 0 & -1 & 0 & 1 \end{bmatrix}$$

$$IC_6 = \begin{bmatrix} -2\sqrt{2} & -\sqrt{5} & -2 & -\sqrt{5} & -2\sqrt{2} \\ -\sqrt{5} & -\sqrt{2} & -1 & -\sqrt{2} & -\sqrt{5} \\ -2 & -1 & 0 & -1 & -2 \\ -\sqrt{5} & -\sqrt{2} & -1 & -\sqrt{2} & -\sqrt{5} \\ -2\sqrt{2} & -\sqrt{5} & -2 & -\sqrt{5} & -2\sqrt{2} \end{bmatrix}.$$

The first initial condition, IC_1, is the initial condition used above in the numerical example and represents a level set function where the level set is on the boundary of the image. IC_2 is similar to IC_1. Conditions IC_3-IC_5 represent level sets which consist of seed squares. The last condition shows a function where the level set is contained within the object to be detected. Each of these is a typical initial condition for the nonlinear segmentation problem. We also note that, although the details have been omitted, all of these initial conditions are stable and convergent by the same techniques introduced above.

We can also treat a more general initial condition as stated above. We define a general signed distance function whose level set represents a circle centered at the center of the image with radius R. This function can be written explicitly as

$$\phi(x, y) = R - \sqrt{(x-a)^2 + (y-b)^2} \tag{2.4}$$

where the point (a, b) gives the center of the image. Taking a radius of $R = 2$ and the center as $(2, 2)$ - which corresponds to the center of the image - gives the initial condition

Condition	Time($\tau = 1$)	Time($\tau = 0.1$)	Time($\tau = 0.01$)
IC_1	2.1551	3.1238	6.5829
IC_2	2.1060	2.8558	49.8644
IC_3	2.0679	2.8342	11.8013
IC_4	2.0675	2.8337	11.8011
IC_5	2.0416	3.1323	99.9414
IC_6	2.1109	2.7299	5.6137
IC_7	2.1021	2.7069	

TABLE 2.1 – Initial conditions and stability for the linearized model.

IC_7 below.

$$IC_7 = \begin{bmatrix} -0.8284 & -0.2361 & 0 & -0.2361 & -0.8284 \\ -0.2361 & 0.5858 & 1.0000 & 0.5858 & -0.2361 \\ 0 & 1.0000 & 2.0000 & 1.0000 & 0 \\ -0.2361 & 0.5858 & 1.0000 & 0.5858 & -0.2361 \\ -0.8284 & -0.2361 & 0 & -0.2361 & -0.8284 \end{bmatrix}.$$

For a given tolerance, we want to determine the time at which all values of the grid points are sufficiently close to their respective steady state values. We define the time as follows. Let $\tau > 0$ be a given tolerance. Let Y be a vector containing the solutions to the system of differential equations described above and let Y_{SS} represent a vector containing the steady-state solution of the system. The time,T, is then given by

$$T = min\{t \in [0, \infty] : |Y_i - Y_{SS}^i| < \tau \text{ for all } i \in [1, 9]\}.$$

Table 2.1 shows the various initial conditions with the time estimates.

As shown in the table, the seed squares tend to give the fastest response and initial detection, but the slowest overall convergence. It is not surprising, then, that IC_6 is the best overall choice, since this condition actually starts closer to the object and its corresponding steady state vector. The last initial condition, representing a level set consisting of a circle of radius R surrounding the object, gives respectable results. The last entry is blank, however, because the solution tended to oscillate around the steady-state

values. Even so, this shows that an entire class of initial conditions given by Equation (2.4) converge and give a good segmentation.

2.3 Generalized Numerical Analysis of the Linear Model

In this section, we complete a more general analysis of the linearized model. To this end, we note that the preceding stability analysis does not lend itself to a general treatment. The reason is that without concrete examples, the form of the matrix A will not be obvious. This makes an analysis of the eigensystem quite challenging since the matrix depends not only upon the initial condition but also on the actual grid used for the discretization. Recalling that i corresponds to the y-coordinate, j corresponds to the x-coordinate, and the grid size is considered to be square ($h = k$), consider now the general numerical scheme given by

$$\begin{aligned}\frac{\phi_{i,j}^{n+1} - \phi_{i,j}^{n}}{\Delta t} &= \mu(P_1(\frac{\phi_{i,j+1}^{n} - \phi_{i,j-1}^{n}}{2h}) + P_2(\frac{\phi_{i+1,j}^{n} - \phi_{i-1,j}^{n}}{2h}) \\ &-(1 + (\phi_x^0)_{i,j}^2)(\frac{\phi_{i,j+1}^{n} - 2\phi_{i,j}^{n} + \phi_{i,j-1}^{n}}{h^2}) \\ &-(1 + (\phi_y^0)_{i,j}^2)(\frac{\phi_{i+1,j}^{n} - 2\phi_{i,j}^{n} + \phi_{i-1,j}^{n}}{h^2}) \\ &-2((\phi_x^0)_{i,j}(\phi_y^0)_{i,j})(\frac{1}{2h})(\frac{\phi_{i+1,j+1}^{n} - \phi_{i+1,j-1}^{n}}{2h} - \frac{\phi_{i-1,j+1}^{n} - \phi_{i-1,j-1}^{n}}{2h})) \\ &+W_{i,j}^{n} \qquad (2.5)\end{aligned}$$

where $W_{i,j}^{n}$ is given by

$$W_{i,j}^{n} = [\frac{\Delta\phi^0}{|\nabla\phi^0|} - \vec{P}(x,y) \circ \nabla\phi^0 - \lambda_1(\vec{u}_0 - c_1(\vec{\phi}))^2 + \lambda_2(\vec{u}_0 - c_2(\vec{\phi}))^2]_{i,j}.$$

Notice that W is just the value of Q with the addition of the fitting terms at each grid point (ik, jh). In general, we need to show that the numerical scheme is consistent and stable. These two items are then needed to show that the numerical scheme converges. Each of these are presented in the following. In order to prove these properties, we will be making liberal use of Taylor's Series expansions. We must then assume that the solutions to the linearized pde are sufficiently smooth such that the derivatives in the following analysis have meaning. Other assumptions will be stated as needed. Here, we have included the

parameter μ but we again leave the term $\delta(\phi)$ out of the model. In fact, the numerical results all use $\mu = 1$. We shall see that taking larger values of this parameter will require a more strict condition on the time step to maintain stability.

2.3.1 Consistency

We will adopt much of the notation of [35] and the concept of the Local Truncation Error (LTE) at each grid point, $\tau_{i,j}^n$. The LTE is found by replacing the actual solution, say $v(x, y, t)$, in the difference equation representing the numerical scheme. We cannot expect the actual solution to satisfy that equation exactly, and so the LTE is the error associated with using the numerical scheme. Consistency may now be defined as follows.

Definition 2.3.1. *(Consistency) A numerical scheme is* ***consistent*** *if* $\tau_{i,j}^n \to 0$ *as* $h \to 0$ *and* $\Delta t \to 0$.

In other words, the truncation error goes to zero as the grid becomes finer; therefore, if we could take a perfect grid (a continuous grid), the numerical scheme would approximate the pde without error. Determining the LTE is simply a matter of using a Taylor's expansion with the actual solution to the pde, $v(x, y, t)$. The expansions may

then be written

$$
\begin{aligned}
v_{i,j}^{n+1} &= v_{i,j}^n + (v_{i,j}^n)_t\Delta t + \frac{1}{2}(v_{i,j}^n)_{tt}\Delta t^2 + \frac{1}{6}(v_{i,j}^n)_{ttt}\Delta t^3 + O(\Delta t^4) \\
v_{i-1,j-1}^n &= v_{i,j}^n - (v_{i,j}^n)_x h - (v_{i,j}^n)_y h + \frac{1}{2}(v_{i,j}^n)_{xx}h^2 + (v_{i,j}^n)_{xy}h^2 + \frac{1}{2}(v_{i,j}^n)_{yy}h^2 + \\
&\quad \frac{1}{6}[-(v_{i,j}^n)_{xxx}h^3 - 3(v_{i,j}^n)_{xxy}h^3 - 3(v_{i,j}^n)_{xyy}h^3 - (v_{i,j}^n)_{yyy}h^3] + O(h^4) \\
v_{i+1,j-1}^n &= v_{i,j}^n - (v_{i,j}^n)_x h + (v_{i,j}^n)_y h + \frac{1}{2}(v_{i,j}^n)_{xx}h^2 - (v_{i,j}^n)_{xy}h^2 + \frac{1}{2}(v_{i,j}^n)_{yy}h^2 + \\
&\quad \frac{1}{6}[-(v_{i,j}^n)_{xxx}h^3 + 3(v_{i,j}^n)_{xxy}h^3 - 3(v_{i,j}^n)_{xyy}h^3 + (v_{i,j}^n)_{yyy}h^3] + O(h^4) \\
v_{i-1,j+1}^n &= v_{i,j}^n + (v_{i,j}^n)_x h - (v_{i,j}^n)_y h + \frac{1}{2}(v_{i,j}^n)_{xx}h^2 - (v_{i,j}^n)_{xy}h^2 + \frac{1}{2}(v_{i,j}^n)_{yy}h^2 + \\
&\quad \frac{1}{6}[(v_{i,j}^n)_{xxx}h^3 - 3(v_{i,j}^n)_{xxy}h^3 + 3(v_{i,j}^n)_{xyy}h^3 - (v_{i,j}^n)_{yyy}h^3] + O(h^4) \\
v_{i+1,j+1}^n &= v_{i,j}^n + (v_{i,j}^n)_x h + (v_{i,j}^n)_y h + \frac{1}{2}(v_{i,j}^n)_{xx}h^2 + (v_{i,j}^n)_{xy}h^2 + \frac{1}{2}(v_{i,j}^n)_{yy}h^2 + \\
&\quad \frac{1}{6}[(v_{i,j}^n)_{xxx}h^3 + 3(v_{i,j}^n)_{xxy}h^3 + 3(v_{i,j}^n)_{xyy}h^3 + (v_{i,j}^n)_{yyy}h^3] + O(h^4) \\
v_{i+1,j}^n &= v_{i,j}^n + (v_{i,j}^n)_y h + \frac{1}{2}(v_{i,j}^n)_{yy}h^2 + \frac{1}{6}(v_{i,j}^n)_{yyy}h^3 + O(h^4) \\
v_{i-1,j}^n &= v_{i,j}^n - (v_{i,j}^n)_y h + \frac{1}{2}(v_{i,j}^n)_{yy}h^2 - \frac{1}{6}(v_{i,j}^n)_{yyy}h^3 + O(h^4) \\
v_{i,j+1}^n &= v_{i,j}^n + (v_{i,j}^n)_x h + \frac{1}{2}(v_{i,j}^n)_{xx}h^2 + \frac{1}{6}(v_{i,j}^n)_{xxx}h^3 + O(h^4) \\
v_{i,j-1}^n &= v_{i,j}^n - (v_{i,j}^n)_x h + \frac{1}{2}(v_{i,j}^n)_{xx}h^2 - \frac{1}{6}(v_{i,j}^n)_{xxx}h^3 + O(h^4)
\end{aligned}
$$

where $O(h^l)$ represents the l-order terms of the expansion. Using the numerical scheme introduced at the beginning of this section and substituting $v_{i,j}^n$ gives

$$
\begin{aligned}
\tau_{i,j}^n &= \frac{v_{i,j}^{n+1} - v_{i,j}^n}{\Delta t} - \mu(P_1(\frac{v_{i,j+1}^n - v_{i,j-1}^n}{2h}) - P_2(\frac{v_{i+1,j}^n - v_{i-1,j}^n}{2h}) \\
&\quad +(1 + (\phi_x^0)_{i,j}^2)(\frac{v_{i,j+1}^n - 2v_{i,j}^n + v_{i,j-1}^n}{h^2}) \\
&\quad +(1 + (\phi_y^0)_{i,j}^2)(\frac{v_{i+1,j}^n - 2v_{i,j}^n + v_{i-1,j}^n}{h^2}) \\
&\quad +2((\phi_x^0)_{i,j}(\phi_y^0)_{i,j})(\frac{1}{2h})((\frac{v_{i+1,j+1}^n - v_{i+1,j-1}^n}{2h} - \frac{v_{i-1,j+1}^n - v_{i-1,j-1}^n}{2h})) \\
&\quad -W_{i,j}.
\end{aligned}
$$

Each term in the previous expression may now be evaluated based upon the Taylor's expansions above. Then, for example, the first term may be written

$$
\frac{v_{i,j}^{n+1} - v_{i,j}^n}{\Delta t} = (v_{i,j}^n)_t + \frac{1}{2}(v_{i,j}^n)_{tt}\Delta t + \frac{1}{6}(v_{i,j}^n)_{ttt}\Delta t^2 + O(\Delta t^3).
$$

Using this same approach for all of the terms and simplifying with the pde shows that the LTE is $O(\Delta t + h^2)$. In other words, the method is first order accurate in time and second order accurate in space. This proves consistency since as the time step and spatial grid decrease, the LTE tends to zero.

2.3.2 Stability

We need to choose an appropriate norm to prove stability. Consider the sup-norm (or ℓ^∞ norm) applied to an $N \times N$ matrix $Z = \{(z)_{i,j} | i, j = 1...N\}$

$$||Z||_\infty = sup_{i,j}|z_{i,j}|.$$

Now define the operator B as

$$\begin{aligned} B\phi^n_{i,j} &= \phi^n_{i,j} + \mu\Delta t[A_{00}\phi^n_{i-1,j-1} + A_{01}\phi^n_{i-1,j} + A_{02}\phi^n_{i-1,j+1} + A_{10}\phi^n_{i,j-1} \\ &\quad + A_{11}\phi^n_{i,j} + A_{12}\phi^n_{i,j+1} + A_{20}\phi^n_{i+1,j-1} + A_{21}\phi^n_{i+1,j} + A_{22}\phi^n_{i+1,j+1}] \end{aligned}$$

where the coefficients $A_{o,o}$ are defined as before. The numerical scheme may now be written

$$\phi^{n+1}_{i,j} = B\phi^n_{i,j} + W^n_{i,j}.$$

Each of these terms will require special treatment. Let us begin with $W^n_{i,j}$.

Recall that $W^n_{i,j}$ is given by

$$W^n_{i,j} = [\frac{\Delta\phi^0}{|\nabla\phi^0|} - \vec{P}(x,y) \circ \nabla\phi^0 - \lambda_1(\vec{u}_0 - c_1(\phi^n))^2 + \lambda_2(\vec{u}_0 - c_2(\phi^n))^2]_{i,j}.$$

The image, u_0, may be taken to be bounded between some reasonable constants. For instance, for a grayscale image, we may be assume that the intensities found in the image are between $[0, 255]$ representing a range of intensities from black to white. From the Chan-Vese model, we know that the constants c_1 and c_2 represent the average intensity of the image inside and outside the zero level set, respectively. Thus, both of these constants are bounded by the image. This allows us to write

$$\begin{aligned} |W^n_{i,j}| &\leq |\frac{\Delta\phi^0}{|\nabla\phi^0|} - \vec{P}(x,y) \circ \nabla\phi^0|_{i,j} + \lambda_1((\vec{u}_0 - c_1(\phi^n))^2)_{i,j} + \lambda_2((\vec{u}_0 - c_2(\phi^n))^2)_{i,j} \\ &\leq |\frac{\Delta\phi^0}{|\nabla\phi^0|} - \vec{P}(x,y) \circ \nabla\phi^0|_{i,j} + (\lambda_1 + \lambda_2)||u_0||^2_\infty. \end{aligned}$$

The last inequality comes from the fact that the fitting terms will not have values larger than the maximum intensity present in the image. Finally, we have

$$||W^n||_\infty \leq ||\frac{\triangle\phi^0}{|\nabla\phi^0|} - \vec{P}(x,y) \circ \nabla\phi^0||_\infty + (\lambda_1 + \lambda_2)||u_0||^2_\infty.$$

Now, we consider terms of the form $Bv^n_{i,j}$ where $||v||_\infty$ is finite. Returning to the definition of the operator B gives

$$\begin{aligned} |Bv^n_{i,j}| &= |v^n_{i,j} + \mu\Delta t[A_{00}v^n_{i-1,j-1} + A_{01}v^n_{i-1,j} + A_{02}v^n_{i-1,j+1} + A_{10}v^n_{i,j-1} \\ &\quad + A_{11}v^n_{i,j} + A_{12}v^n_{i,j+1} + A_{20}v^n_{i+1,j-1} + A_{21}v^n_{i+1,j} + A_{22}v^n_{i+1,j+1}]| \\ &\leq |1 + \mu\Delta t\Sigma^2_{r,s=0}A_{r,s}|||v^n||_\infty. \end{aligned}$$

Recalling the definitions of $A_{\circ,\circ}$ allows us to bound the sum. This bound is then

$$|1 + \mu\Delta t\Sigma^2_{f,g=0}A_{f,g}| \leq |1 + (\mu\Delta t)(-\frac{8}{h^2} - \frac{4}{h^2}|\nabla\phi^0|^2)|.$$

Using all of these terms in the numerical scheme then gives

$$\begin{aligned} |\phi^{n+1}_{i,j}| &= |B\phi^n_{i,j} + W^n_{i,j}| \\ &\leq |B\phi^n_{i,j}| + |W^n_{i,j}| \\ &\leq |1 + \mu\Delta t(-\frac{8}{h^2} - \frac{4}{h^2}|\nabla\phi^0|^2)|||\phi^n||_\infty + M \end{aligned}$$

where $M = ||\frac{\triangle\phi^0}{|\nabla\phi^0|} - \vec{P}(x,y)\circ\nabla\phi^0||_\infty + (\lambda_1+\lambda_2)||u_0||^2_\infty$. Let $\alpha = |1+\mu\Delta t(-\frac{8}{h^2} - \frac{4}{h^2}|\nabla\phi^0|^2)|$. Then we may write

$$|\phi^{n+1}_{i,j}| \leq \alpha||\phi^n||_\infty + M.$$

Taking the supremum then gives

$$||\phi^{n+1}||_\infty \leq \alpha||\phi^n||_\infty + M.$$

Now applying the previous expression recursively gives

$$||\phi^{n+1}||_\infty \leq \alpha^{n+1}||\phi^0||_\infty + M(1 + \Sigma^n_{l=1}\alpha^l).$$

If the right hand side of the above inequality is bounded, the solutions will be bounded at all time steps. We could rewrite the above expression as

$$||\phi^n||_\infty \leq \alpha^n||\phi^0||_\infty + M(1 + \Sigma^{n-1}_{l=1}\alpha^l)$$

which gives an explicit bound on ϕ^n. If $\alpha < 1$, we can guarantee that the right hand side of the inequality is bounded even as $n \to \infty$. This, then is the requirement for stability and gives a bound involving the time step and the initial condition. Further, as expected, the value of μ enters into the stability requirement and forces the choice of a smaller time step. Moreover, if we also insert the term $\delta(\phi)$ back into the model, the effect on stability is analagous to that of the parameter μ since in practice we use a regularized version of the delta distribution. This regularized version will be smooth, and, hence, bounded.

2.3.3 Convergence

Now that we have consistency and criteria which guarantees stability, we may determine whether the numerical scheme converges to the true solution. The approach is to consider the numerical scheme involving the estimate $\phi_{i,j}^n$ and subtract the numerical scheme with the true solution $v_{i,j}^n$ including the LTE. This operation will produce a difference equation involving the global error, E defined by

$$E_{i,j}^n = \phi_{i,j}^n - v_{i,j}^n.$$

The major difficulty now is that the "constants" c_1 and c_2 are found as average values of the level set function over the domain of the image. For now, we must assume that these values truly are constant for each time step. This situation would correspond to a stable segmentation of the image (i.e., a segmentation where the level set no longer moves). With this assumption, we will produce a difference equation for the global error given by

$$E_{i,j}^{n+1} = BE_{i,j}^n + \tau_{i,j}^n.$$

From the stability analysis and for a specific time, T, where $N\Delta t = T$, we may write

$$||E^N||_\infty \leq \alpha^N ||E^0||_\infty + \Sigma_{l=0}^{N-1} \alpha^l ||\tau^{N-l}||_\infty.$$

This may be rewritten as

$$||E^N||_\infty \leq \alpha^N ||E^0||_\infty + max_{n \leq N} ||\tau^n||_\infty \Sigma_{l=0}^{N-1} \alpha^l$$

since we may simply take the maximum truncation error within all time steps up to and including the time step for N. We will now take the step sizes to zero; however, we cannot do this in an arbitrary way. In fact, the step sizes all go to zero such that $\alpha < 1$ for all choices of $(h, \Delta t)$. From consistency, we know that the LTE approaches zero. Then, with appropriate initial data, the first term in the above inequality will also approach zero. This proves convergence of the numerical scheme and suggests a theorem for the previous results.

Theorem 2.3.2. *Let $\mu > 0$ be the parameter as defined in the active contours without gradient model. Let h be the step size of a square grid and let Δt be the time step. The linear numerical scheme given by Equation 2.5 is consistent with first order accuracy in time and second order accuracy in spatial dimensions. Moreover, the method is stable for appropriate choices of h, Δt, and initial condition ϕ^0 that satisfy*

$$\alpha = |1 + \mu \Delta t(-\frac{8}{h^2} - \frac{4}{h^2}|\nabla \phi^0|^2)| < 1.$$

Finally, the method converges to a solution of the linear model.

It is worth noting that the typical initial condition for imaging problems of this type is a signed distance function. Under this condition, the bounds given above simplify significantly since for a signed distance function $|\nabla \phi^0| = 1$.

2.4 Numerical Results For the Linearized Model

In this section we present numerical results for the linearized model. The calculations have been completed on a machine running Windows XP with Matlab, a dual core 1.60 Ghz processor, and 2 gigabytes of RAM. We will define the error as the maximum difference between successive iterations or more precisely by

$$Error \;=\; max_{(i,j)}|\phi_{i,j}^{n+1} - \phi_{i,j}^{n}|.$$

The relative error is then the error scaled by the previous iteration, or

$$R.Error \;=\; max_{(i,j)}|\frac{\phi_{i,j}^{n+1} - \phi_{i,j}^{n}}{\phi_{i,j}^{n}}|.$$

Number	Figures	Error	R. Error
1	2.3, 2.4	$1.82 * 10^{-12}$	$4.69 * 10^{-16}$
2	2.5, 2.6	$3.41 * 10^{-13}$	$6.22 * 10^{-16}$
3	2.7, 2.8	$1.82 * 10^{-12}$	$6.61 * 10^{-16}$
4	2.9, 2.10, 2.11	$6.82 * 10^{-13}$	$3.81 * 10^{-15}$
5		0.0185	$7.31 * 10^{-05}$

TABLE 2.2 – Results of Linear Imaging.

Results are shown in Table 2.2. All images use the same initial condition given by the function

$$\phi^0(x,y) = 80 - \sqrt{(x-100.5)^2 + (y-100.5)^2}$$

and shown in Figure 2.2. The first four entries in the table all come from 200 iterations of the linearized model.

Figures 2.3, 2.4 show the image and the segmentation, respectively, and give proof that the linearized model is giving very good segmentation results for simple images. We draw attention to the errors and relative errors in the table. Figures 2.5, 2.6 show the same image, but now we have introduced noise. The segmentation is again complete. The Chan-Vese model is robust in the presence of noise. This shows that the linear model inherits that same robustness.

Figures 2.7, 2.8 show our standard image and the resulting segmentation given by the linear model. Again, we have very low errors and relative errors which imply that the solutions are close to numerical convergence. Further, the images show that we are able to detect objects no matter where the location of the initial zero level set - a particularly important property of the nonlinear model.

Another important property of the nonlinear model is its ability to detect objects which are not defined by a gradient, such as those found in a smooth image. We might not expect the linear model to be able to segment these types of images as they are typically far removed from simple. However, as shown in Figures 2.9, 2.10, 2.11, the method works suprisingly well.

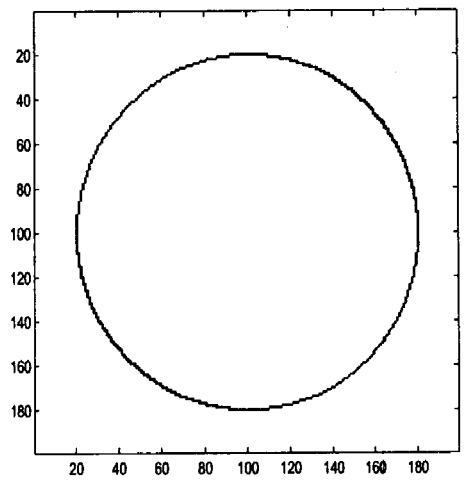

FIGURE 2.2 – The initial condition used with the linearized model.

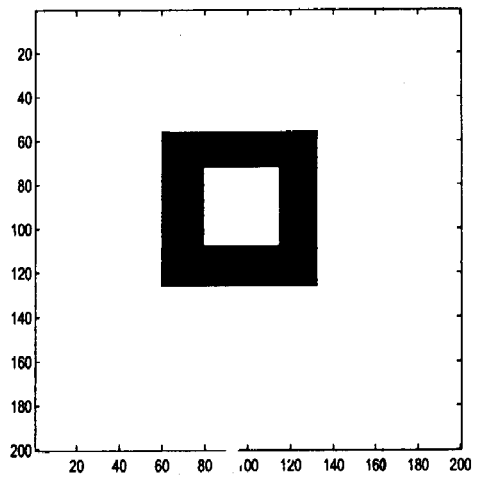

FIGURE 2 3 – Simple image containing no noise and one internal contour used with the linearized model.

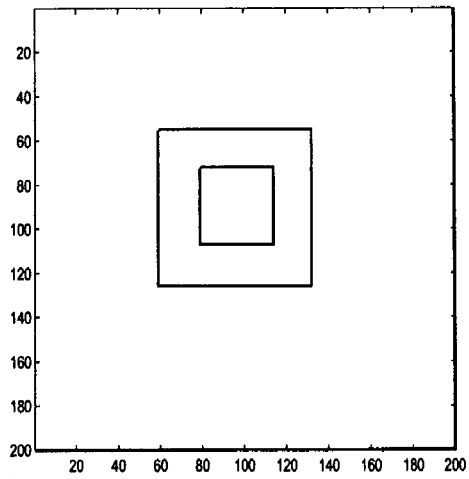

FIGURE 2.4 – This figure shows the segmentation of a simple image with one internal contour.

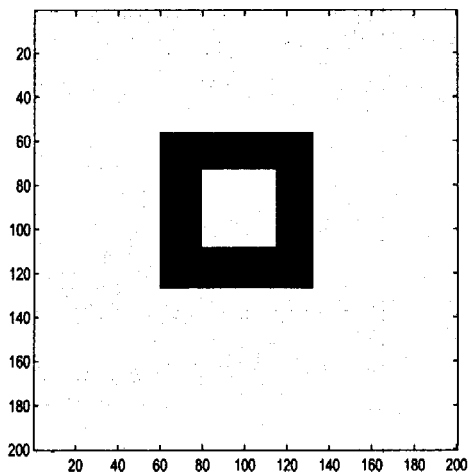

FIGURE 2.5 – Simple image containing noise taken from a uniform distribution.

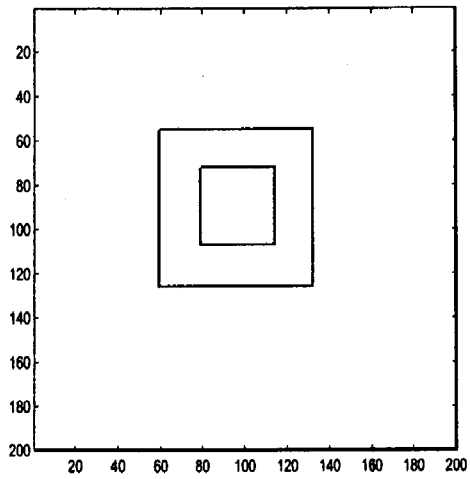

FIGURE 2.6 – The segmentation of an image containing noise. The linear model has inherited the nonlinear model's robustness.

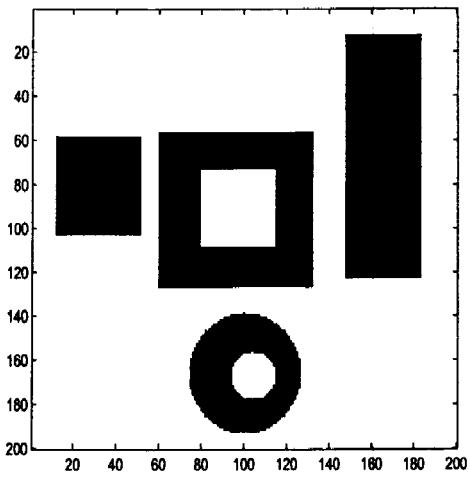

FIGURE 2.7 – Simple image containing more shapes and different types of internal contours.

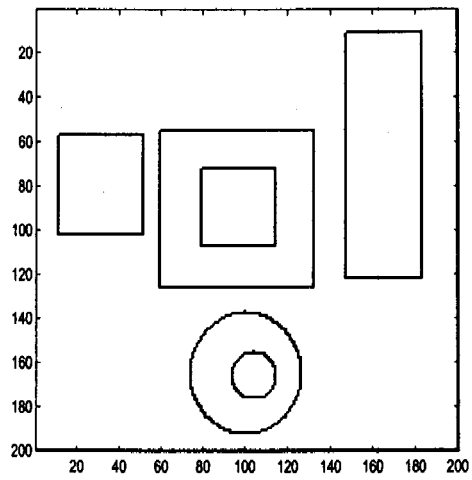

FIGURE 2.8 – Figure shows the segmentation of the simple image with more internal contours. The initial condition started both inside/outside the objects.

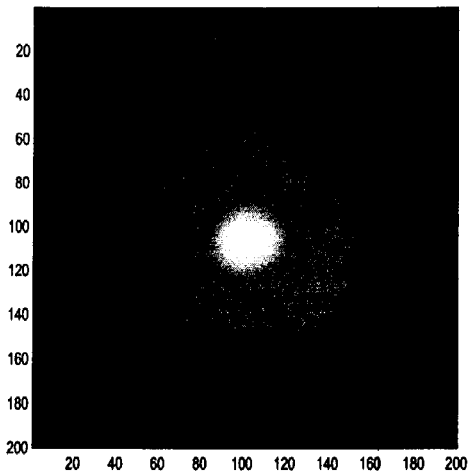

FIGURE 2.9 – Smoothed synthetic image showing distinct intensity regions.

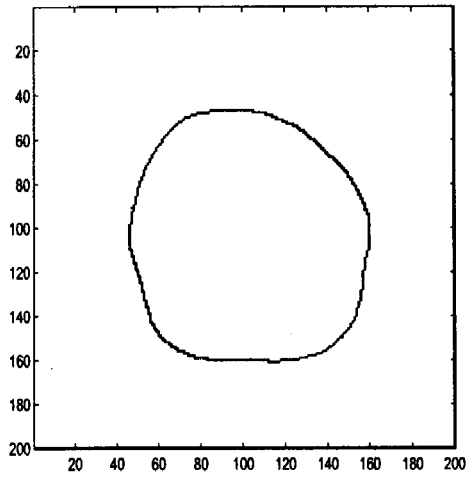

FIGURE 2.10 – This figure illustrates that the ability to detect objects in smooth images has been retained from the nonlinear model.

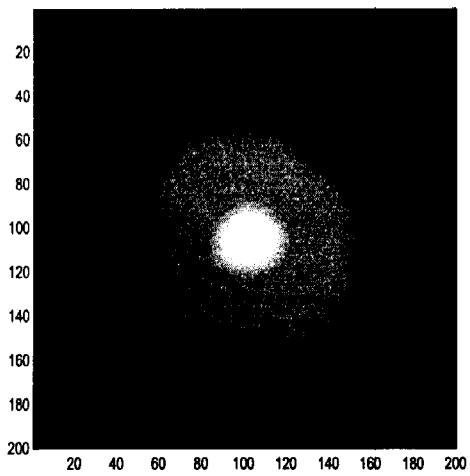

FIGURE 2.11 – Image showing the segmentation overlayed on top of the image.

As stated earlier, we have removed the $\delta(\phi)$ term from the linear model. The reason for this may be seen from the table. The last entry in that table shows the results of the linear model where the distribution has been inserted. The effect of this term is two-fold. First, the term forces the zero level contour to push toward the edges of the object at a much faster rate than any of the other contours. (Incidentally, as this contour moves toward the object boundaries, it pulls away from other contours which are moving in the same direction but at a much slower rate which can cause the function to experience high frequency oscillations in the region where two contours are pulling away - similar to that seen from instability.) Thus, the actual segmentation of the image may be quite rapid. However, the rapid detection ultimately leads to slow convergence since those contours that are "left behind" by the zero level set must now "catch up". In other words, when the zero level set stops moving, the remaining contours continue moving according to the fitting and regularizing terms. The segmentation comes - mostly - from the movement of the zero level set while the convergence comes from the movement of all other level sets. The second effect comes from this movement of all other level sets. As these level sets begin to catch up to the zero level set, they tend to smooth out the high oscillations and ultimately result in a solution which is bounded with regions characterized by small - almost constant - variations. Without that term, we obtain faster convergence since all level sets are capable of detecting the objects; however, the final result tends to be somewhat rough in appearance. Moreover, the model produces regions which have large gradients or zero gradients with more frequency. This makes the use of the distribution term necessary for the nonlinear model as this phenomena leads to significant numerical problems. The linear model, by contrast, does not suffer from these difficulties and just like reconciliation, the distribution term is not necessary unless the application requires a function which needs to have those low variation regions. We complete this chapter by mentioning that results for the linear model have been published in [20].

CHAPTER 3
NEW PROCEDURE USING FOURTH-ORDER FITTING TERMS

The motivation behind this section is to increase the rate of convergence of the level sets. In other words, we wish to detect the edges at a faster rate without losing the benefits of the model. One way to do this is to look at the energy integrals with a slightly different form. Consider

$$\begin{aligned} F(C, c_1, c_2) &= \mu \text{ length(C)} + \nu \text{ area(inside C)} + \\ &\lambda_1 \int_{inside(C)} |u_0 - c_1|^4 dxdy + \lambda_2 \int_{outside(C)} |u_0 - c_2|^4 dxdy \end{aligned} \tag{3.1}$$

where we retain all definitions and ideas previous. This equation will behave in a similar fashion to that proposed by Chan and Vese; however, we expect that objects are detected faster. For instance, if the object to be detected is completely inside the curve C, then $F_1(C) > 0$ and $F_2(C) \approx 0$. However, large deviations from the average intensity will result in larger reponses from the evolution equation than the former model. This will cause the level set curve to shrink rapidly toward the edges of the object to be detected. Similarly, if the object to be detected contains the curve C, then $F_1(C) \approx 0$ and $F_2(C) > 0$. In this case, the large deviation from the average intensity on the outside of the curve will force the level set to expand rapidly. The only expected consequence is that as the curve approaches the edges of the object, the deviation will be smaller than the deviation present in the original model. The fourth order polynomial will make this small deviation even smaller. This causes a smaller response to the deviation than the Chan-Vese model. So, as the curve gets closer to the edges of the object to be detected, we expect convergence to slow. The following will then serve as a proof of concept which will then suggest a new procedure for object detection.

The derivation of the model follows a similar approach to that above. The new

model is then given by

$$\begin{aligned} F(\phi, c_1, c_2) &= \mu \int_\Omega \delta_0(\phi)|\nabla\phi| dxdy + \nu \int_\Omega H(\phi) dxdy + \\ &\lambda_1 \int_\Omega |u_0 - c_1|^4 H(\phi) dxdy + \\ &\lambda_2 \int_\Omega |u_0 - c_2|^4 (1 - H(\phi)) dxdy. \end{aligned} \tag{3.2}$$

Again, we wish to minimize this equation with respect to c_1 and c_2. This minimization problem is equivalent to minimizing G

$$\begin{aligned} G(\phi, c_1, c_2) &= \mu \int_\Omega \delta_0(\phi)|\nabla\phi| dxdy + \nu \int_\Omega H(\phi) dxdy + \\ &\lambda_1 \int_\Omega (u_0 - c_1)^4 H(\phi) dxdy + \\ &\lambda_2 \int_\Omega (u_0 - c_2)^4 (1 - H(\phi)) dxdy. \end{aligned} \tag{3.3}$$

If we hold ϕ constant and minimize F with respect to c_1 and c_2 we have

$$\begin{aligned} \frac{d}{dc_1} G(\phi, c_1, c_2) &= 0 = -4\lambda_1 \int_\Omega (u_0 - c_1)^3 H(\phi) dxdy \\ \frac{d}{dc_2} G(\phi, c_1, c_2) &= 0 = -4\lambda_2 \int_\Omega (u_0 - c_2)^3 (1 - H(\phi)) dxdy. \end{aligned}$$

Expanding and simplifying the above expressions gives

$$\begin{aligned} -c_1^3 \int_\Omega H(\phi) dxdy + c_1^2 \int_\Omega 3u_0 H(\phi) dxdy - \\ c_1 \int_\Omega 3u_0^2 H(\phi) dxdy + \int_\Omega u_0^3 H(\phi) dxdy &= 0 \\ -c_2^3 \int_\Omega (1 - H(\phi)) dxdy + c_2^2 \int_\Omega 3u_0 (1 - H(\phi)) dxdy - \\ c_2 \int_\Omega 3u_0^2 (1 - H(\phi)) dxdy + \int_\Omega u_0^3 (1 - H(\phi)) dxdy &= 0. \end{aligned}$$

In order to solve these equations, we need to review the solution technique for cubic polynomial equations. Of course, numerical algorithms may be used to estimate the values of c_1 and c_2.

3.1 Solution of Cubic Polynomials

Consider the equation

$$ax^3 + bx^2 + cx + d = 0.$$

Let $x = y - \frac{b}{3a}$. This substitution gives

$$\begin{aligned}
a(y - \frac{b}{3a})^3 + b(y - \frac{b}{3a})^2 + c(y - \frac{b}{3a}) + d &= 0 \\
ay^3 - 3a\frac{b}{3a}y^2 + 3a(\frac{b}{3a})^2 y - a(\frac{b}{3a})^3 + by^2 - 2b\frac{b}{3a}y + b(\frac{b}{3a})^2 + cy - c\frac{b}{3a} + d &= 0 \\
y^3 + (\frac{3ac - b^2}{3a^2})y + (\frac{2b^3 - 9a^2bc + 27a^3d}{27a^3}) &= 0 \\
y^3 + py + q &= 0
\end{aligned}$$

where $p = \frac{3ac-b^2}{3a^2}$ and $q = \frac{2b^3-9a^2bc+27a^3d}{27a^3}$. Now let $y = z - \frac{p}{3z}$ for $z \neq 0$. Substituting and multiplying by z^3 gives

$$\begin{aligned}
(z - \frac{p}{3z})^3 + p(z - \frac{p}{3z}) + q &= 0 \\
z^3 - 3\frac{p}{3z}z^2 + 3(\frac{p}{3z})^2 z - (\frac{p}{3z})^3 + pz - p\frac{p}{3z} + q &= 0 \\
z^3(z^3 - 3\frac{p}{3z}z^2 + 3(\frac{p}{3z})^2 z - (\frac{p}{3z})^3 + pz - p\frac{p}{3z} + q) &= 0 \\
z^6 + qz^3 - (\frac{p^3}{27}) &= 0.
\end{aligned}$$

This resulting equation is now quadratic in z^3. Solving gives

$$z^3 = \frac{-q \pm \sqrt{q^2 + \frac{4p^3}{27}}}{2}.$$

Solving this final equation will give six values for z. With these values, we can now back-calculate to find y and then, finally, x.

3.2 Final Model

We may now return to the model derivation. This derivation will be analogous to Chan and Vese's derivation. The only differences will be the final form of the model and of course, the determination of the constants c_1 and c_2. Thus, we have the following

$$\begin{aligned}
\frac{\partial \phi}{\partial t} &= \delta_{2\epsilon}(\phi)(\mu \operatorname{div}(\frac{\nabla \phi}{|\nabla \phi|}) - \nu - \lambda_1(u_0 - c_1)^4 + \lambda_2(u_0 - c_2)^4) \\
\phi(0, x) &= \phi_0(x) \\
\frac{\partial \phi}{\partial \vec{n}} &= 0
\end{aligned} \tag{3.4}$$

where c_1 and c_2 are found from the equations

$$\begin{aligned} -c_1^3 \int_\Omega H(\phi)dxdy + c_1^2 \int_\Omega 3u_0 H(\phi)dxdy - \\ c_1 \int_\Omega 3u_0^2 H(\phi)dxdy + \int_\Omega u_0^3 H(\phi)dxdy &= 0 \\ -c_2^3 \int_\Omega (1 - H(\phi))dxdy + c_2^2 \int_\Omega 3u_0(1 - H(\phi))dxdy - \\ c_2 \int_\Omega 3u_0^2(1 - H(\phi))dxdy + \int_\Omega u_0^3(1 - H(\phi))dxdy &= 0. \end{aligned}$$

3.3 Numerical Examples and Discretization

All numerical simulations have been done on a Windows XP machine, 2 Ghz processor, 1 Gb of RAM, running Matlab. The figures are shown with values of the various parameters, the iterations of the program, and an estimate of the computation time. Unless otherwise noted, the mesh size $\Delta x = \Delta y = h = 1$ and $\Delta t = 0.1$, and all images are size 200×200, grayscale. Note that the following results show "stable" level set curves and not proper mathematical convergence since the idea is to show the rapid response of the fourth order polynomial. Later in this chapter, we treat convergence with more care.

Chan and Vese adopt the discretization of the divergence term found in [45] and use an iterative algorithm proposed in [7]. We use the divergence discretization found in [45] but, for now, we use a non-iterative algorithm derived from an explicit finite difference scheme. Let

$$\begin{aligned} \Delta_-^x \phi &= \phi_{i,j} - \phi_{i-1,j} \\ \Delta_+^x \phi &= \phi_{i+1,j} - \phi_{i,j} \\ \Delta_-^y \phi &= \phi_{i,j} - \phi_{i,j-1} \\ \Delta_+^y \phi &= \phi_{i,j+1} - \phi_{i,j}. \end{aligned}$$

We may then calculate ϕ^{n+1} by using the following

$$\begin{aligned}\frac{\phi_{i,j}^{n+1}-\phi_{i,j}^{n}}{\Delta t} &= \delta_h(\phi_{i,j}^n)\Bigg[\frac{\mu}{h^2}\Delta_-^x\Big(\frac{\Delta_+^x\phi_{i,j}^n}{\sqrt{\frac{(\Delta_+^x\phi_{i,j}^n)^2}{h^2}+\frac{(\phi_{i,j+1}^n-\phi_{i,j-1}^n)^2}{(2h)^2}}}\Big)\\ &+ [\frac{\mu}{h^2}\Delta_-^y\Big(\frac{\Delta_+^y\phi_{i,j}^n}{\sqrt{\frac{(\Delta_+^y\phi_{i,j}^n)^2}{h^2}+\frac{(\phi_{i+1,j}^n-\phi_{i-1,j}^n)^2}{(2h)^2}}}\Big)\\ &- \nu-\lambda_1(u_{0,i,j}-c_1(\phi^n))^2+\lambda_2(u_{0,i,j}-c_2(\phi^n))^2\Bigg].\end{aligned}$$

The following results show the method in action.

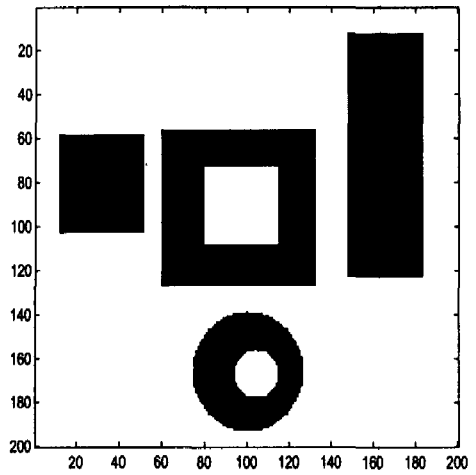

FIGURE 3.1 – The original image with objects to be detected.

Figure 3.1 shows the initial images with some simple shapes we wish to detect while Figure 3.2 shows the initial level set function used. Following Chan and Vese, we use $\lambda_1 = \lambda_2 = 1$ and $\nu = 0$. In Figure 3.3 we show the resulting edges with $\mu = 0.1 * 255^2$. Using (3.4) instead, we need one iteration requiring only 0.53 seconds. Figures 3.4-3.7 show a typical curve evolution for the Chan-Vese model.

Figure 3.9 shows a synthetic image where noise has been introduced using a random number generator. Figure 3.8 shows the detected edges of the image. Figures 3.10-3.12 show the evolution of the level set curves for iterations of 100, 500 and 1000, respectively. Here, we choose $\mu = 10 * 255^2$ to limit the detection to larger groups of objects. The initial level set function is the same as that used for the first sample image. If we run

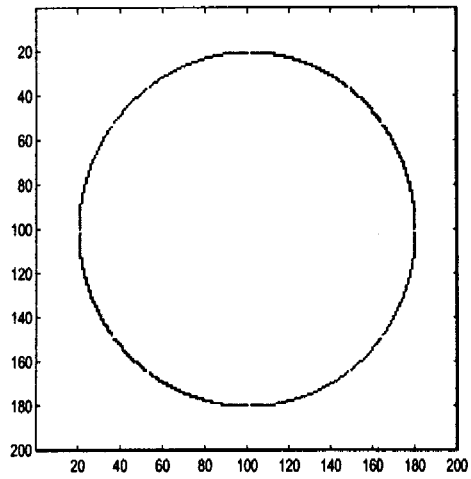

FIGURE 3.2 – Initial level set curve given by $\phi_0(x, y) = 80 - \sqrt{(x - 100)^2 + (y - 100)^2}$.

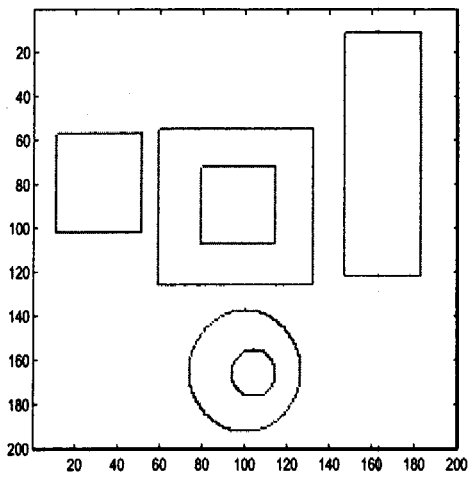

FIGURE 3.3 – The final zero level set showing the detected edges. The parameter values are $\mu = 0.1 * 255^2$, $\nu = 0$, $\lambda_1 = \lambda_2 = 1$. Program ran for 101 iterations, 3.32 seconds.

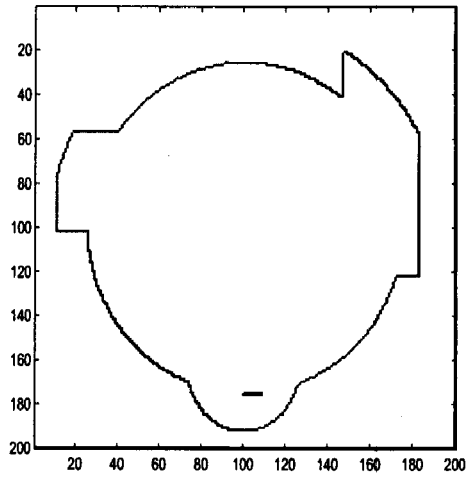

FIGURE 3.4 – The parameter values are $\mu = 0.1 * 255^2$, $\nu = 0$, $\lambda_1 = \lambda_2 = 1$. Program ran for 20 iterations.

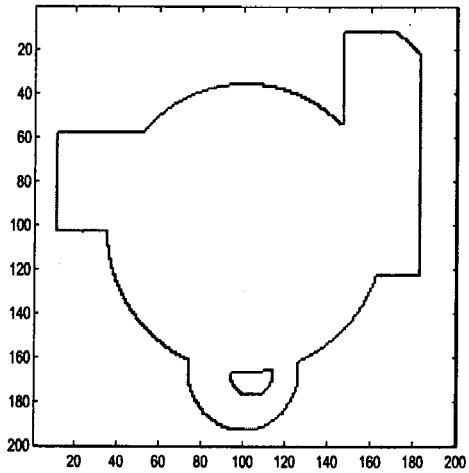

FIGURE 3.5 – The parameter values are $\mu = 0.1 * 255^2$, $\nu = 0$, $\lambda_1 = \lambda_2 = 1$. Program ran for 40 iterations.

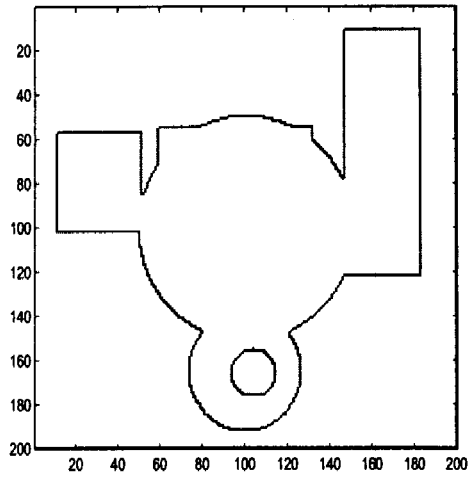

FIGURE 3.6 – The parameter values are $\mu = 0.1 * 255^2$, $\nu = 0$, $\lambda_1 = \lambda_2 = 1$. Program ran for 60 iterations.

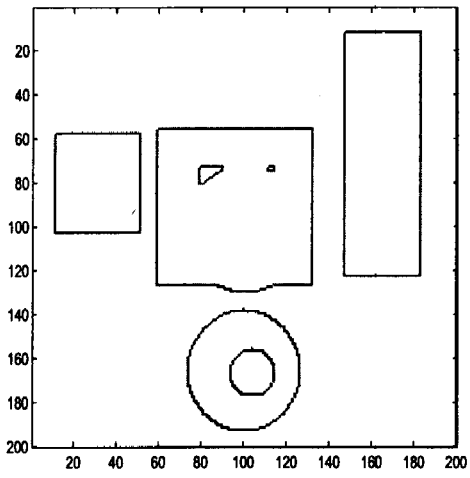

FIGURE 3.7 – The parameter values are $\mu = 0.1 * 255^2$, $\nu = 0$, $\lambda_1 = \lambda_2 = 1$. Program ran for 80 iterations.

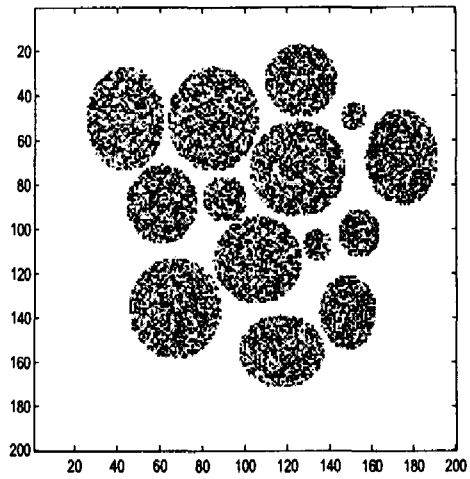

FIGURE 3.8 – The initial image with the noise introduced by a random number generator.

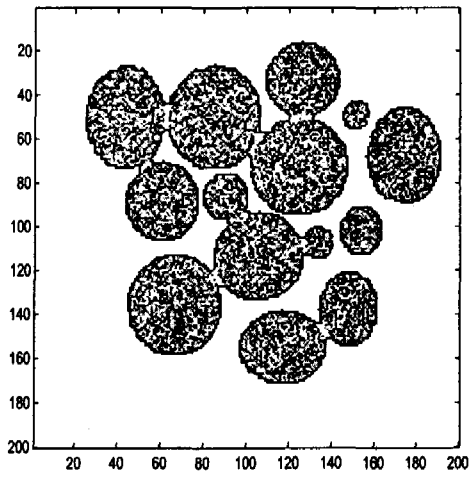

FIGURE 3.9 – The image with the detected edges. The parameter values are $\mu = 10*255^2$, $\nu = 0$, $\lambda_1 = \lambda_2 = 1$. The program ran for 1500 iterations, 44.2 seconds.

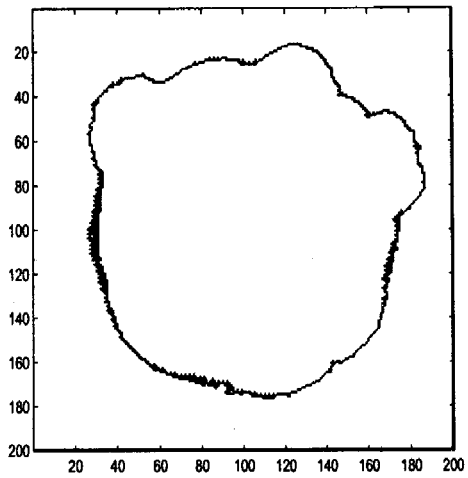

FIGURE 3.10 – The parameter values are $\mu = 10 * 255^2$, $\nu = 0$, $\lambda_1 = \lambda_2 = 1$. The program ran for 100 iterations.

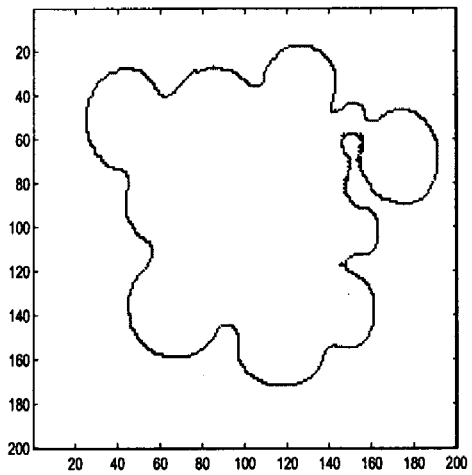

FIGURE 3.11 – The parameter values are $\mu = 10 * 255^2$, $\nu = 0$, $\lambda_1 = \lambda_2 = 1$. The program ran for 500 iterations.

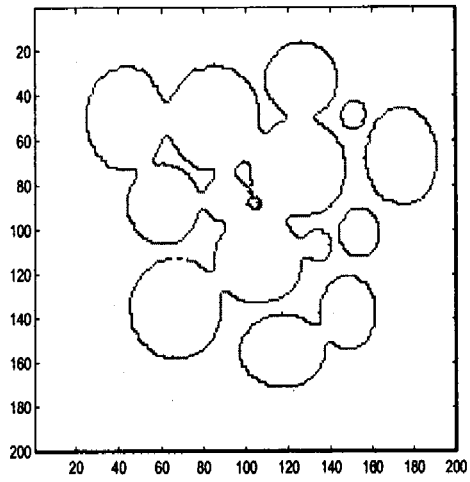

FIGURE 3.12 – The parameter values are $\mu = 10 * 255^2$, $\nu = 0$, $\lambda_1 = \lambda_2 = 1$. The program ran for 1000 iterations.

(3.4) we need only one iteration and 0.54 seconds. The results shown in Figure 3.13 are the result of using the explicit scheme.

One of the strengths of the Chan-Vese model is that it detects edges that are not defined by gradient. Images that are blurred have smooth transitions between regions of constant intensity and, therefore, have edges which are difficult to detect with gradients. Figures 3.14-3.18 show the evolution of the level sets for the Chan-Vese model through 5000 iterations of the program used on a blurred image. Figures 3.19-3.21 show the evolution on the same image but using our model. Note that we have achieved slightly better detection but with only 50 iterations requiring 5.47 seconds compared to the chan-vese model of 5000 iterations and over two minutes.

These results suggest a possible improvement over the Chan-Vese model. The suggested procedure is then given by the following.

1. Initialize ϕ_0 to a signed distance function.

2. Run the fourth order polynomial model for a few iterations. The result of this model will give us a new initial condition for the Chan-Vese model. Note that the this procedure could be used to specify a better initial condition for the original model. We discuss this later in this chapter.

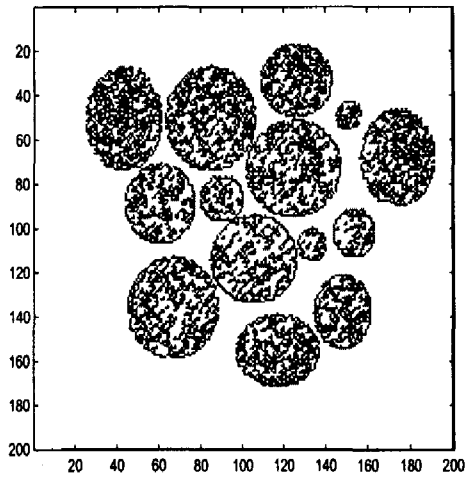

FIGURE 3.13–The level set for a noisy image. The parameter values are $\mu = 10 * 255^2$, $\nu = 0$, $\lambda_1 = \lambda_2 = 1$. The program ran for one iteration, 0.54 seconds.

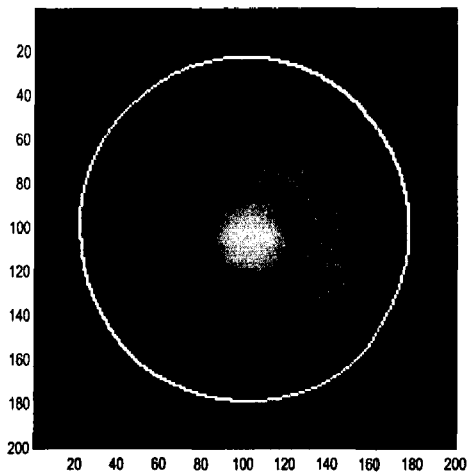

FIGURE 3.14–The parameter values are $\mu = 0.1 * 255^2$, $\nu = 0$, $\lambda_1 = \lambda_2 = 1$. The program ran for 10 iterations.

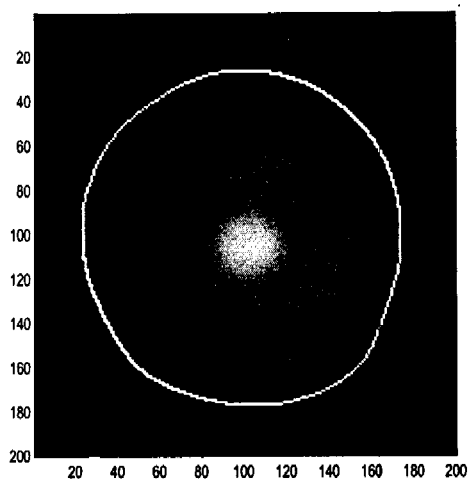

FIGURE 3.15 – The parameter values are $\mu = 0.1*255^2$, $\nu = 0$, $\lambda_1 = \lambda_2 = 1$. The program ran for 50 iterations.

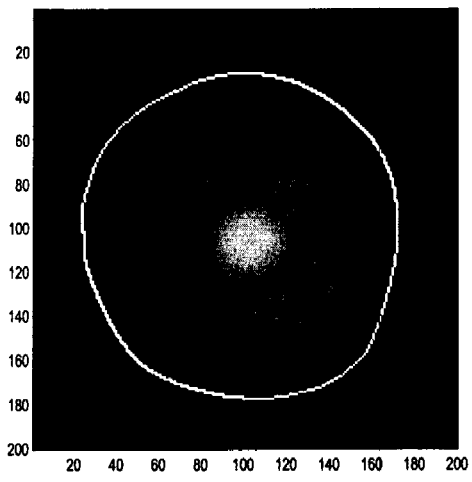

FIGURE 3.16 – The parameter values are $\mu = 0.1*255^2$, $\nu = 0$, $\lambda_1 = \lambda_2 = 1$. The program ran for 100 iterations.

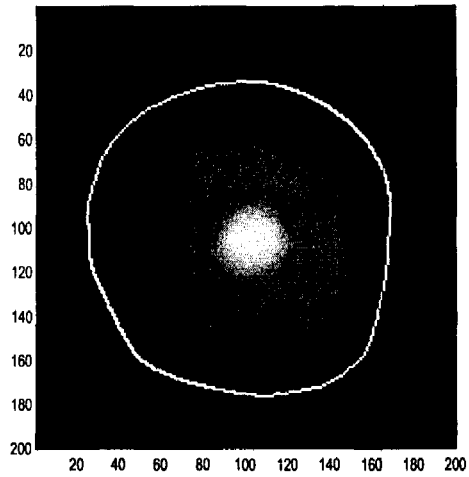

FIGURE 3.17 – The parameter values are $\mu = 0.1*255^2$, $\nu = 0$, $\lambda_1 = \lambda_2 = 1$. The program ran for 1000 iterations.

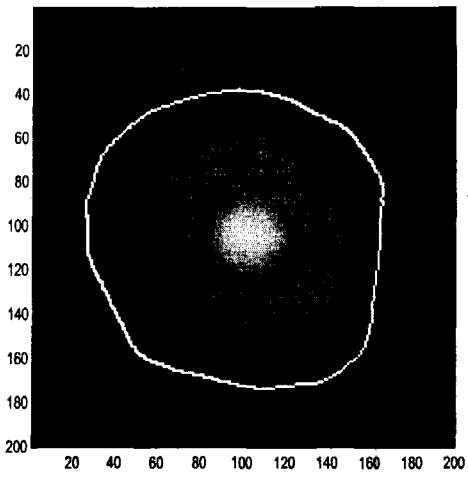

FIGURE 3.18 – The parameter values are $\mu = 0.1*255^2$, $\nu = 0$, $\lambda_1 = \lambda_2 = 1$. The program ran for 5000 iterations, 140.97 seconds.

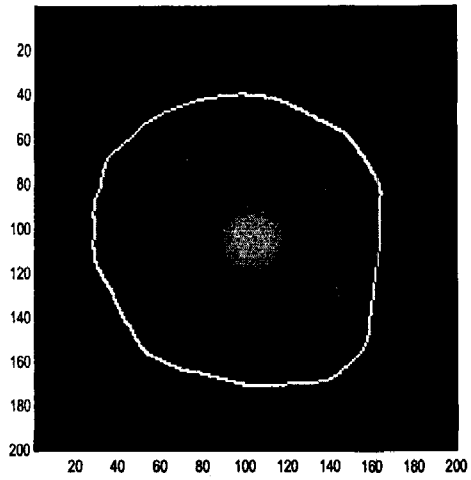

FIGURE 3.19 – The proposed model with parameter values $\mu = 0.1 * 255^2$, $\nu = 0$, $\lambda_1 = \lambda_2 = 1$. The program ran for one iteration.

3. Take the level-set curve from the previous iterations and reconcile to the distance function if desired.

4. Run the Chan-Vese Active Contours model as usual with the iterative numerical scheme.

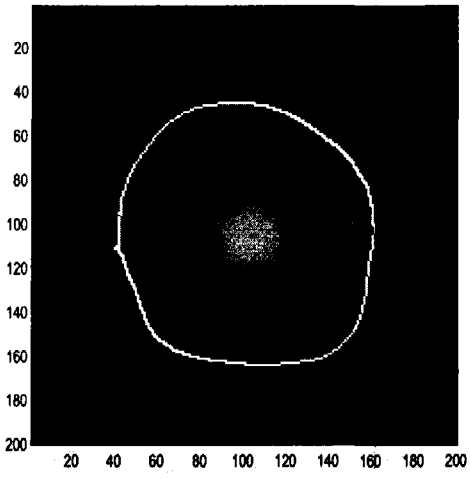

FIGURE 3.20 – The proposed model with parameter values $\mu = 0.1 * 255^2$, $\nu = 0$, $\lambda_1 = \lambda_2 = 1$. The program ran for 10 iterations.

3.4 Comments on Numerical Convergence

This section gives a more careful treatment of numerical convergence. We retain the definitions for error and relative error as given in the numerical section of the linearized model. The idea is that numerical convergence is achieved when the error or relative error falls below some predefined tolerance. The previous section showed that the segmentation occurs quite rapidly for both the Chan-Vese model and the new proposed fourth order model. Now, however, we are more concerned with actual numerical convergence. In this section, we will see that both models converge slowly, although the fourth order does speed convergence.

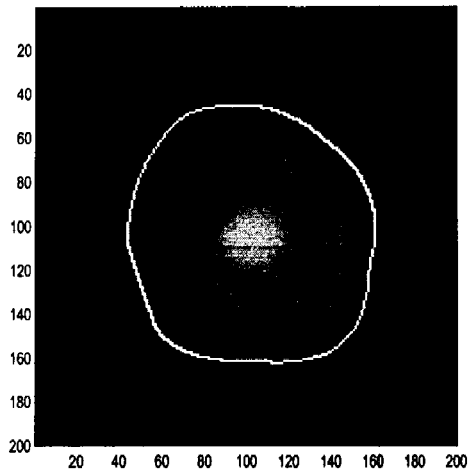

FIGURE 3.21 – The proposed model with parameter values $\mu = 0.1 * 255^2$, $\nu = 0$, $\lambda_1 = \lambda_2 = 1$. The program ran for 50 iteration, 5.47 seconds.

The parameter values are taken to be fixed at $\mu = 100$, $\Delta t = 0.1$, and we are now using an iterative approach as prescribed in [7]. Further, the initial condition has also been fixed and possesses a zero level set consisting of one circle centered at $(101, 101)$ - representing a pixel location - with a radius of 50 pixels. The numerical scheme is then

given by

$$\begin{aligned}\frac{\phi_{i,j}^{n+1}-\phi_{i,j}^{n}}{\Delta t} &= \delta_h(\phi_{i,j}^{n})\Bigg[\frac{\mu}{h^2}\Delta_-^x\Big(\frac{\Delta_+^x\phi_{i,j}^{n+1}}{\sqrt{\frac{(\Delta_+^x\phi_{i,j}^{n})^2}{h^2}+\frac{(\phi_{i,j+1}^{n}-\phi_{i,j-1}^{n})^2}{(2h)^2}}}\Big)\\ &+ [\frac{\mu}{h^2}\Delta_-^y\Big(\frac{\Delta_+^y\phi_{i,j}^{n+1}}{\sqrt{\frac{(\Delta_+^y\phi_{i,j}^{n})^2}{h^2}+\frac{(\phi_{i+1,j}^{n}-\phi_{i-1,j}^{n})^2}{(2h)^2}}}\Big)\\ &- \nu-\lambda_1(u_{0,i,j}-c_1(\phi^n))^2+\lambda_2(u_{0,i,j}-c_2(\phi^n))^2\Bigg]\end{aligned}$$

where all previous finite difference definitions are retained. Since the fourth order model will only be used to specify an initial condition for the second order Chan-Vese model, we will continue to use an explicit procedure for its solution. There is, of course, the option of reconciling the output of the fourth order model before using it as the initial condition for the second order model. The rest of this section is then a discussion of results obtained from testing the procedures. The test image is given by Figure 2.3.

We ran both the second order model and then the new fourth order procedure independently. Both programs ran for many hours to obtain the best possible results. We also note that both methods detect the object in the image accurately and both also detect the internal contour that is present. The second order model ran for 651,000 iterations without reconciliation and ended with an error of $7.447 * 10^{-4}$. It is interesting to note, however, that at iteration 6,591, the energy of the segmentation as defined by Equation 1.16 was only changing by 1.59 with a relative change of 0.0001. This shows that even at this low iteration, the segmentation of the image is almost complete. The rest of the iterations are required for numerical convergence. In contrast, the fourth order procedure ran for 593,112 iterations. However, the results for this procedure include only one iteration of the fourth order code. The objective was to obtain the same error as in the second order model. The new procedure then gave a saving of approximately 60,000 iterations - or about one hour. Interestingly, at iteration 6,591, the fourth order model energy was changing by 0.0006 with a relative energy change of $2.371 * 10^{-6}$. This shows that the new procedure speeds the detection and also aids in numerical convergence.

3.5 Initial Condition Considerations

Since the new procedure may be used to specify a good initial condition for the active contours without gradient model, the choice of initial condition is not as important. Care must still be taken, however, since the new procedure inherits the lack of unique minimizers from the original model. In this section, we follow the ideas set forth in [19] in order to compare the effect of three initial conditions upon five different schemes. That particular analysis is carried out using multiple level set functions, but the ideas may still be applied here. The schemes are defined by

$$\begin{aligned} CV &= \text{200 iterations of the original model by Chan and Vese} \\ NP_1 &= \text{one iteration fourth order, 200 iterations of Chan-Vese} \\ NP_2 &= \text{two iterations fourth order, 200 iterations of Chan-Vese} \\ NP_5 &= \text{five iterations fourth order, 200 iterations of Chan-Vese} \\ L &= \text{200 iterations of the linear model.} \end{aligned}$$

These schemes are compared with three initial conditions given by Figures 2.2, 3.22, and 3.23 and used on the image shown in Figure 2.3. The first initial condition represents one large circle surrounding the objects. The second initial condition represents four circles each with radius of 30 pixels while the last condition represents a set of 1089 seed circles each with radius 2 pixels. The analysis set forth in [19] shows that initial conditions of the type in Figure 3.23 tend to give the best segmentation and lowest energy. Our results agree.

The idea is to compare the energy at each iteration. The new procedure, however, shows such a large decrease in energy in just a few iterations that a graph is not feasible. Thus, we present results in Table 3.1. Parameter values for both the original model and the new procedure are $\mu = 100$ and $\Delta t = 0.1$. The linear model uses the same value for μ but we need $\Delta t = 0.001$ for stability considerations.

We show the energies after iteration five and after iteration 200. As can be seen from the table, for each iteration, the lowest energy is obtained from NP_5. This is expected since the fourth order procedure has completely segmented the image before the second

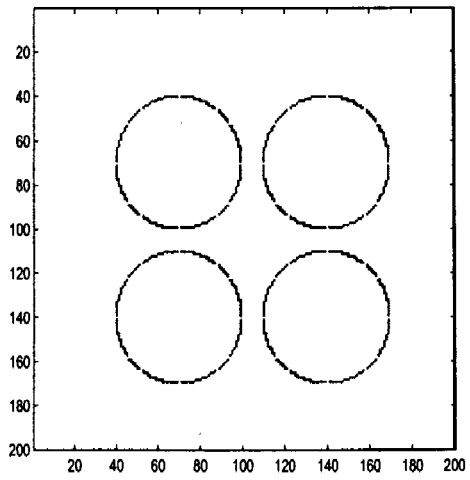

FIGURE 3.22 – Second initial condition used for comparison of methods.

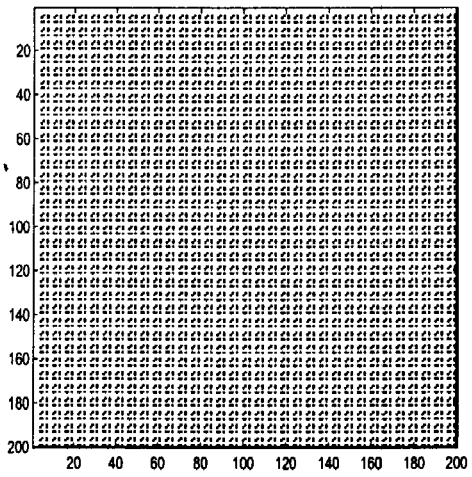

FIGURE 3.23 – Third initial condition used for comparison of methods.

IC	Iterations	CV	NP_1	NP_2	NP_5	L
1	5	$2.12 * 10^8$	87.5	71.1	62.3	$1.36 * 10^5$
	200	$3.75 * 10^4$	87.3	71.1	62.3	$1.36 * 10^5$
2	5	$1.02 * 10^8$	89.6	80.6	69.7	$1.35 * 10^5$
	200	$3.57 * 10^4$	89.5	80.6	69.7	$1.82 * 10^5$
3	5	$9.73 * 10^4$	31.6	31.6	31.5	$1.23 * 10^5$
	200	$3.03 * 10^4$	31.6	31.6	31.6	$1.04 * 10^5$

TABLE 3.1 – Initial condition comparison.

order even begins. In fact, all of the schemes involving the new procedure show very little change in the energy after the first few iterations. This is consistent with results shown previously as the claim is that the segmentation itself is quite fast. Finally, we mention that our results agree completely with those presented by Chan and Vese since we clearly see that initial condition three gives the lowest overall energy for all methods and also gives the fastest response (i.e., the fastest decrease of energy).

Intuitively, this initial condition should be the best choice since we have more "detectors" present in the image. The more detectors present, the more objects we can detect and at a faster rate. Mathematically, the level set is actually starting closer to the edges of any object in the image. The closer the level set is to an object, the less time it should take for the front to reach the object edges.

CHAPTER 4
DATA TRACKING

Let us define an image in a more general way. Suppose that we have a function $u : \Re^2 \times [0, \infty) \rightarrow \Re$ which represents some physical data associated with a desired application. The function u could represent a temperature distribution, concentration of some reacting medium, etc. For a fixed time t, the function will give a set of data $u(x, y; t)$ which will represent the physical quantities in the plane. This data may be interpreted as an "image" of the application.

The Chan-Vese model segments an image into two regions with an average intensity. These intensities are given by c_1 and c_2 where the first gives the intensity "inside" the level set function ϕ and the second gives the average intensity outside ϕ. By treating the data as an "image", we can segment the data into regions of high intensity values and regions of low intensity values. This will allow us to track changes to each region and model the behavior of bulk data. This will be useful in applications where we are not concerned with solution behavior on a small scale but rather we are more concerned with macroscopic details. For instance, we may wish to detect the initial presence of high temperatures associated with some process. These high temperatures may cause material failure over extended periods of time. It is, therefore, essential to model how these regions behave as the model evolves. The region evolution may then be used to lower the risk of material failure and may also give clues as to the reason why such regions occur (e.g. lack of proper ventilation, stagnant fluids, etc.). With this in mind, we begin with two "toy" problems where we explore the feasibility of such an approach.

The Chan-Vese model as written, however, is unsatisfactory for this particular application. The model will segment the data as described; however, we will have little to

no control over the segmentation. Moreover, the segmentation may create regions which have little or no interest. In order to provide more control over the segmentation, we modify the Chan-Vese model in the following way. Suppose that we wish to segment the data into regions which represent known contours. We will then want to segment the regions using some critical value. Let this critical value be represented by c. The value of c is given (i.e., provided by an external source). Let $G(x)$ be given by

$$G(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

Also, we will use a power of four on the fitting terms to speed convergence. Using these modifications, we have the following pde.

$$\begin{aligned} \frac{\partial \phi}{\partial t} &= \delta_{2\epsilon}(\phi)(\mu \ \mathrm{div}(\frac{\nabla \phi}{|\nabla \phi|}) - \nu - \lambda_1(G(u-c)-c_1)^4 + \lambda_2(G(u-c)-c_2)^4) \\ \phi(0,x) &= \phi_0(x) \\ \frac{\partial \phi}{\partial \vec{n}} &= 0 \end{aligned}$$

where the function $u = u(x, y, \tau)$ is the solution of the application for a given time τ. The segmentation, then, is the steady-state solution of this pde. Now we continue with two examples: the transport equation and the heat equation.

4.1 The Transport Equation

Consider the two dimensional transport equation given by

$$\begin{aligned} u_t + (v, \nabla u) &= 0 \\ u(x,0) &= f(x) \end{aligned} \tag{4.1}$$

where it is understood that x is a vector representing all spatial variables, $(\circ, \circ)$ represents the usual dot product, and $v = (v_1, v_2)$ is a fixed vector which represents the direction of transport. For completeness, we review the solution by the method of characteristics [24]. Parameterize a path $(x(s), y(s), t(s))$ with parameter s and initial position given by $(x_0, y_0, 0)$. Define $z(s)$ as

$$z(s) = u(x(s), y(s), t(s)).$$

Now, we differentiate z with respect to s to obtain

$$z'(s) = u_x x'(s) + u_y y'(s) + u_t t'(s).$$

If we define the following equations, $z'(s) = 0$:

$$\begin{aligned} x'(s) &= v_1 \\ y'(s) &= v_2 \\ t'(s) &= 1 \end{aligned}$$

where the initial conditions are $x(0) = x_0$, $y(0) = y_0$, $t(0) = 0$, and $z(0) = u(x_0, y_0, 0) = f(x_0, y_0)$. Solving this system gives the solution as

$$u(x, y, t) = f(x - v_1 t, y - v_2 t).$$

So, as expected, the solution is a translation of the initial condition which implies that the regions of interest are simply moving about the plane.

To illustrate the technique, we consider an initial condition given by

$$f(x) = 200e^{-\frac{1}{10}((x-20)^2+(y-20)^2)} \tag{4.2}$$

and let $v = (1, 1)$. The solution then is given by

$$u(x, y, t) = 200e^{-\frac{1}{10}((x-t-20)^2+(y-t-20)^2)}. \tag{4.3}$$

For all numerical results , we are using the following parameter values: $\mu = 10, \lambda_1 = 1, \lambda_2 = 1, \nu = 0$. The numerical computations for segmentation use a time step of 0.1 and the pixel width for the spatial variables (i.e., $h = \Delta x = \Delta y = 1$). For the initial condition above, suppose we want to track regions that have concentrations of 100 or more (i.e., the critical value is $c = 100$). Then we produce the following numerical results.

The initial contour used for detection is shown in Figure 4.1 while the final contour and the detected image are shown in Figures 4.2 and 4.3, respectively. Figures 4.5-4.9 give the detected regions and detected images for times of $t = 1, t = 3, t = 7$. As can be seen from the figures, the region is being tracked as it moves in the direction of v. (Note that the images are inverted and that the direction of v is toward the lower right.)

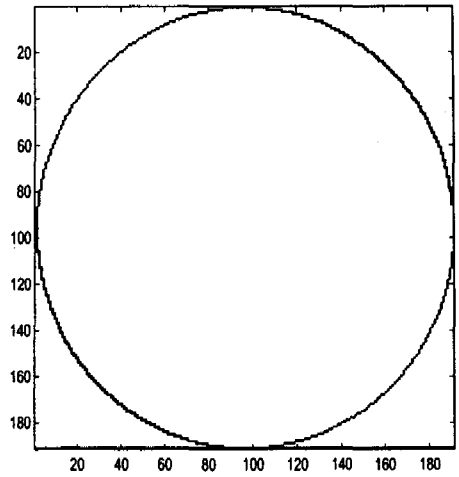

FIGURE 4.1 – The initial contour for the transport equation.

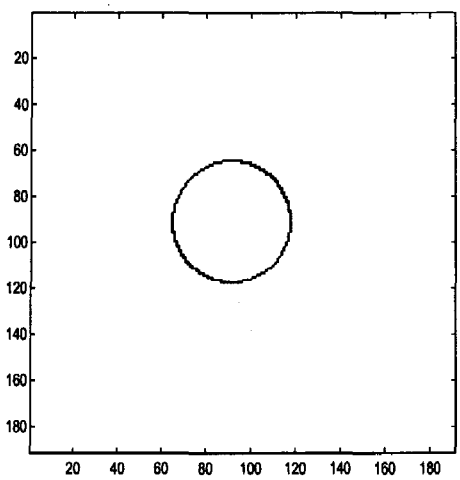

FIGURE 4.2 – The level set showing the region of interest for the initial condition of the transport equation.

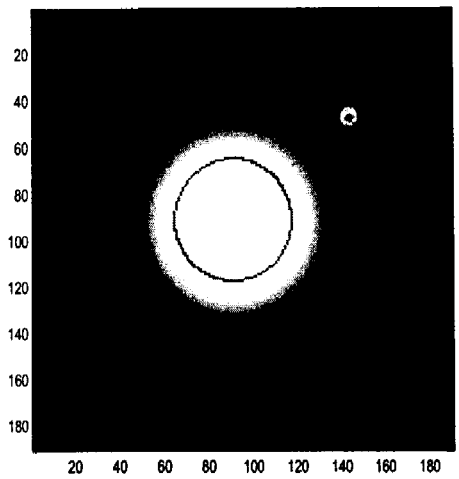

FIGURE 4.3 – The actual solution with the region overlayed. This is the initial condition.

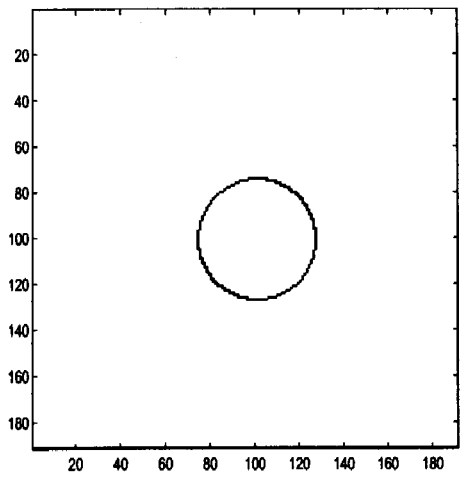

FIGURE 4.4 – The level set showing the region of interest for the transport equation with time of $t = 1$.

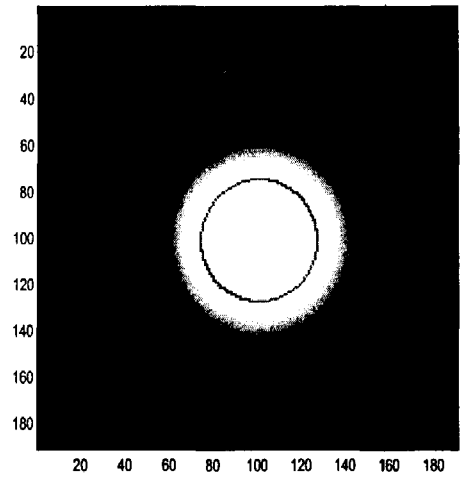

FIGURE 4.5 – The actual solution with the region overlayed for time $t = 1$.

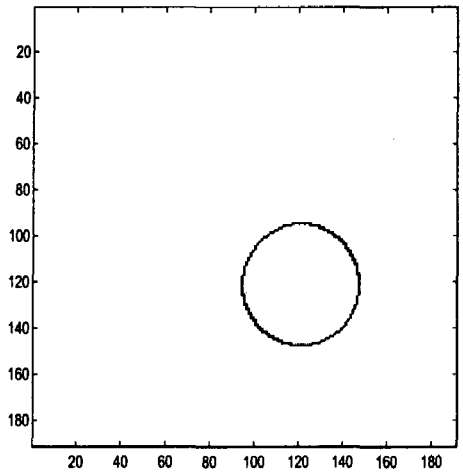

FIGURE 4.6 – The level set showing the region of interest for the transport equation with time of $t = 3$.

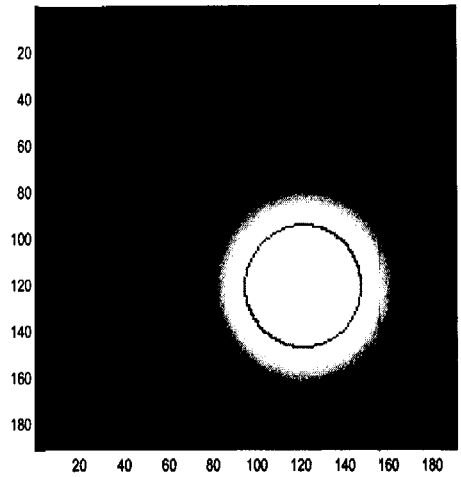

FIGURE 4.7 – The actual solution with the region overlayed for time $t = 3$.

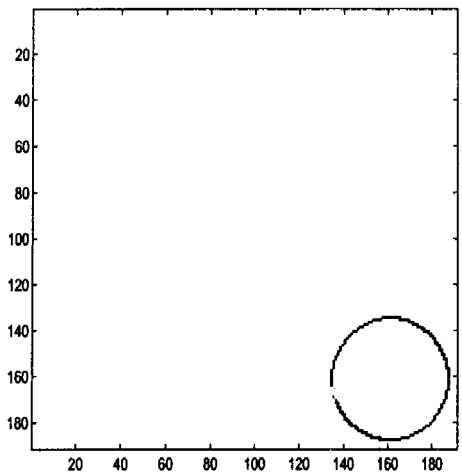

FIGURE 4.8 – The level set showing the region of interest for the transport equation with time of $t = 7$.

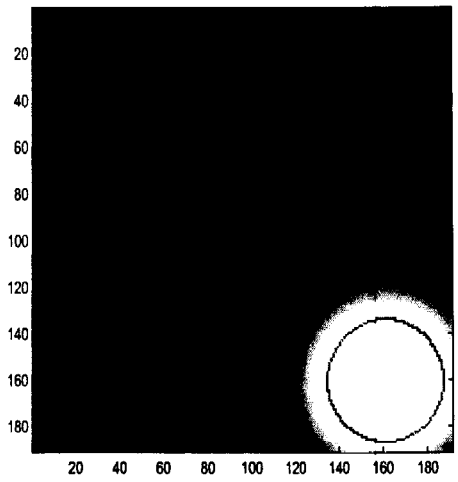

FIGURE 4.9–The actual solution with the region overlayed for time $t = 7$.

4.2 The Diffusion Equation

Now consider the two-dimensional heat equation.

$$u_t = \alpha \Delta u$$

with appropriate boundary and initial conditions. Again, we treat $u(x,t)$ as defined above. For purposes of this section, we let $\alpha = 1$. To solve the equation numerically, we discretize as follows.

$$u_{i,j}^{k+1} = u_{i,j}^{k} + \Delta t\left(\frac{u_{i+1,j}^{k} - 2u_{i,j}^{k} + u_{i-1,j}^{k}}{h^2} + \frac{u_{i,j+1}^{k} - 2u_{i,j}^{k} + u_{i,j-1}^{k}}{h^2}\right). \tag{4.4}$$

We have chosen to use $h = 1, \Delta t = 0.01$. The parameters values are the same as was used for the transport equation including the initial contour in Figure 4.1. We will use an initial condition of

$$u(x,y,0) = 200e^{-\frac{1}{1000}((x-200)^2+(y-200)^2)} \tag{4.5}$$

and track regions of intensity 90 and above. Further, we assume adiabatic conditions at the boundary of the domain

$$\frac{\partial u}{\partial t} = 0 \text{ on } \partial U \tag{4.6}$$

where U is the complete domain of the equation.

The dynamics of this equation are also well-known. The steady-state solution of the equation will be a constant function over the entire domain U. As time increases, we will see two effects depending upon the values of u. If u is greater than the steady-state solution, then u will decrease. If u is less than the steady-state solution, then u will increase. Thus, if we choose a critical value to be above the steady-state solution, the region we are tracking should decrease in size as time increases and approach a single point. Alternatively, if we choose a critical value less than the steady-state, the region we are tracking will increase until the entire domain is contained inside the region. Choosing the critical value to be the steady-state value will result in no movement of the region initially. At the limit, however, the region will be the entire domain. For now, with the above conditions and a critical value of $c = 90$, we obtain the following results.

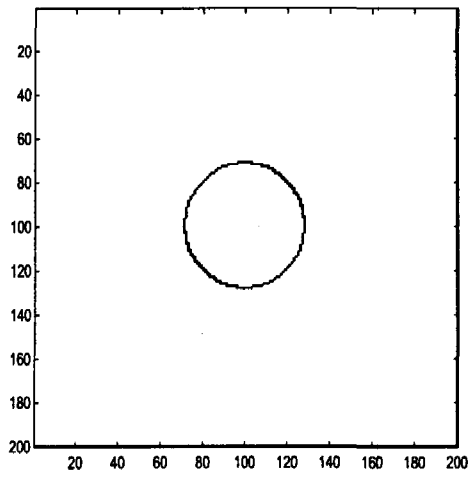

FIGURE 4.10 – The final contour for the heat equation at time $t = 0$.

Again, the figures show both the final contour of the chan-vese segmentation and the final image which gives the solution to the heat equation with region tracking overlayed. As before, the solutions are inverted with the y-axis increasing as we go down the graph. From the figures, we see the region is decreasing.

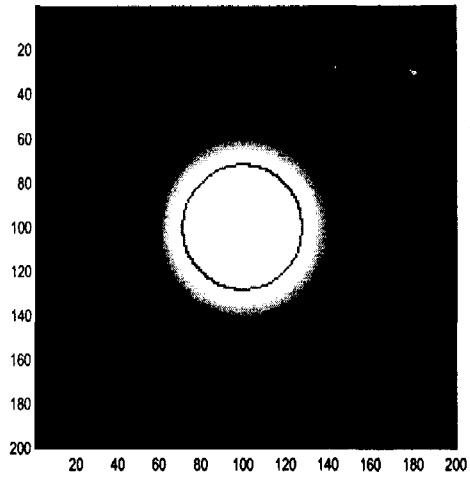

FIGURE 4.11 – The final contour for the heat equation at time $t = 0$.

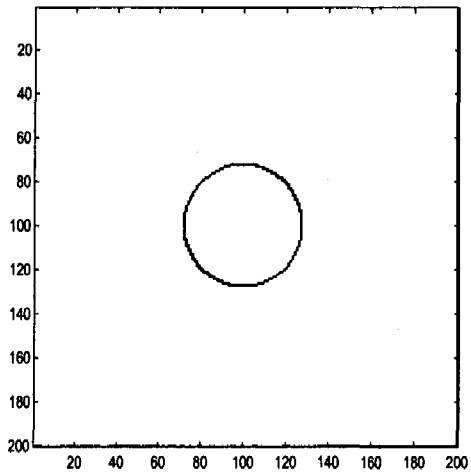

FIGURE 4.12 – The final contour for the heat equation at time $t = 30$.

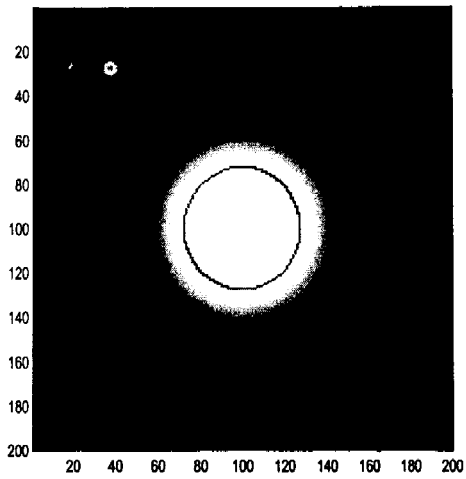

FIGURE 4.13 – The final contour for the heat equation at time $t = 30$.

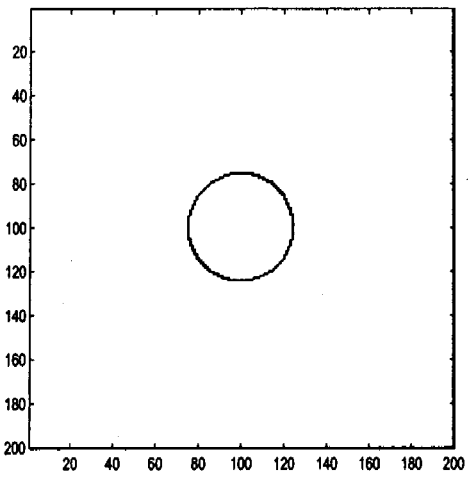

FIGURE 4.14 – The final contour for the heat equation at time $t = 120$.

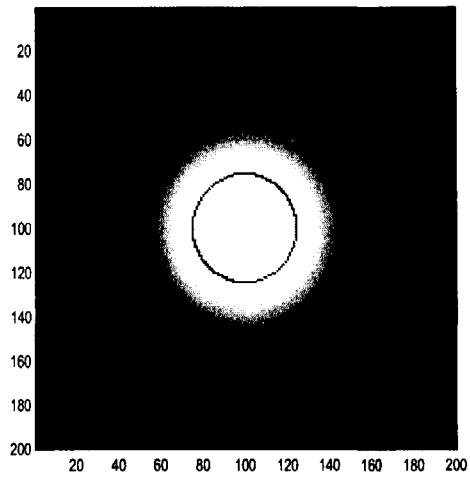

FIGURE 4.15 – The final contour for the heat equation at time $t = 120$.

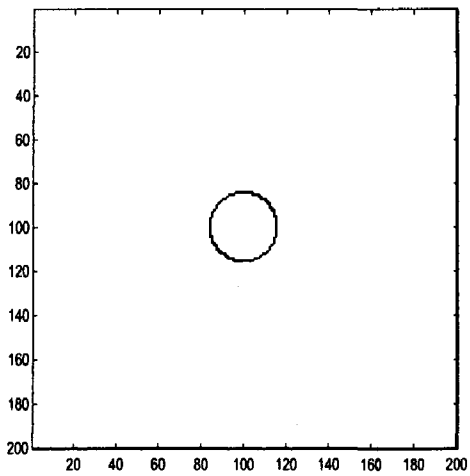

FIGURE 4.16 – The final contour for the heat equation at time $t = 240$.

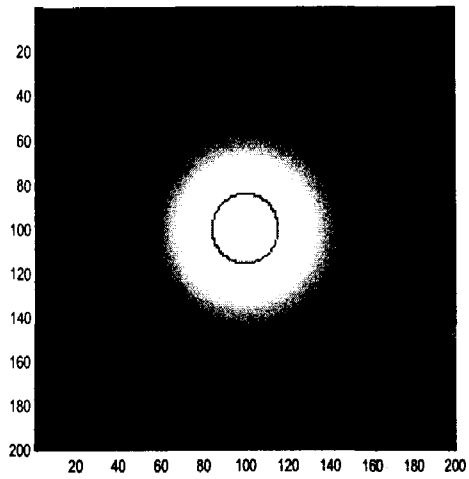

FIGURE 4.17 – The final contour for the heat equation at time $t = 240$.

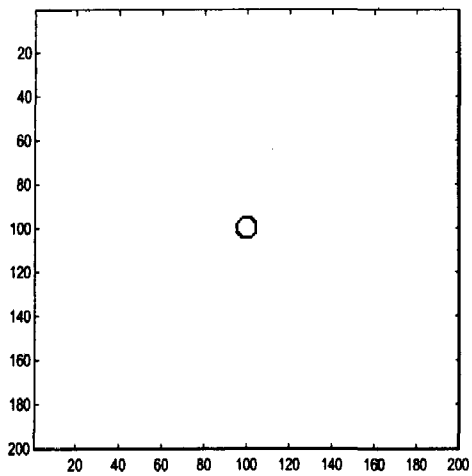

FIGURE 4.18 – The final contour for the heat equation at time $t = 300$.

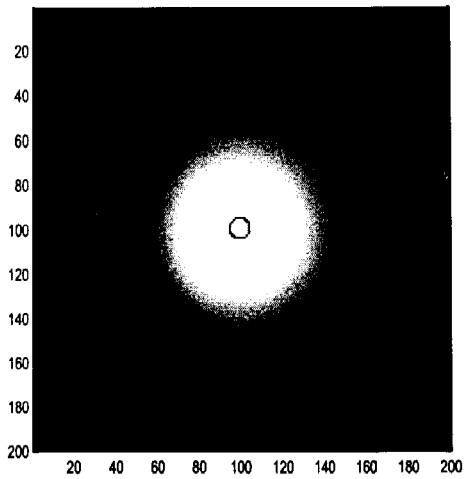

FIGURE 4.19 – The final contour for the heat equation at time $t = 300$.

4.3 Reaction-Diffusion Equation

In order to further substantiate the claims made above, we now turn our attention to a more complicated example: the reaction-diffusion equation. This type of equation may be written as

$$u_t = D\Delta u - R(u,t)$$

where $R(u,t)$ represents the reaction term and D is the diffusion coefficient. The notation above assumes that the equation is meant to represent the concentration of a reactant. We will choose a simple reaction, use an iterative scheme to solve the resulting pde, and then use mathematical imaging to track regions of interest. The difference in this section is that we will now use the linearized model for the segmentation. Since the linearized model gives fast segmentation and convergence for simple images, it is ideally suited for this particular application. Of course, we could also use the fourth order procedure as discussed in the previous sections.

Consider a first-order ideal reaction given by

$$A \rightarrow R$$

where k is the reaction constant, A is the reactant, and R is the only product. We will use notation found in [27]; however, both Fogler [27] and Levenspiel [34] are classic books

on chemical reaction engineering. Further, Murray's classic books [40] [41] are also good references on the reaction-diffusion equation and give a complete analysis.

Assume that the reaction is carried out on a square surface of dimensions $L \times L$ but with negligible thickness (i.e., on a table top). So $\Omega = \{(x,y) \in \Re^2 | 0 \leq x \leq L, 0 \leq y \leq L\}$ is the domain of the concentration function. Define the concentration of reactant A as $C_A = C_A(x,y,t)$. Also assume Neumann boundary conditions

$$\frac{\partial C_A}{\partial \vec{n}} = 0 \text{ on } \partial\Omega$$

and initial condition of $C_A(x,y,0) = C_{A0}(x,y)$. We then define the rate of disappearance of the reactant A as

$$(-r_A) = kC_A.$$

Substituting into the equation gives our model

$$\begin{aligned} (C_A)_t &= D\Delta(C_A) - kC_A \text{ in } \Omega \\ \frac{\partial C_A}{\partial \vec{n}} &= 0 \text{ on } \partial\Omega \\ C_A(x,y,0) &= C_{A0}(x,y). \end{aligned} \tag{4.7}$$

Next, we wish to nondimensionalize the pde. This may be done by using the following substitutions.

$$\begin{aligned} \bar{C}_A &= max_{(x,y)} C_{A0}(x,y) \\ \tau &= kt \\ \beta &= \frac{D}{kL} \\ \bar{x} &= \frac{x}{L} \\ \bar{y} &= \frac{y}{L} \\ \hat{C}_A &= \frac{C_A}{\bar{C}_A}. \end{aligned}$$

With these subsitutions and after some simplification, Equation (4.7) becomes

$$\begin{aligned} (\hat{C}_A)_t &= D\Delta(\hat{C}_A) - k\hat{C}_A \text{ in } \Omega \\ \frac{\partial \hat{C}_A}{\partial \vec{n}} &= 0 \text{ on } \partial\Omega \\ \hat{C}_A(x,y,0) &= \hat{C}_{A0}(x,y). \end{aligned} \tag{4.8}$$

4.3.1 Numerical Results

We solve Equation (4.8) for 50 time steps with $L = 10$, $\Delta\bar{x} = \Delta\bar{y} = 0.1$, $\Delta\tau = 0.01$ and $\beta = 0.10$. The results are shown for tracking concentration values of 0.8 or higher. The initial condition is given by

$$\hat{C}_{A0}(x, y) = e^{-\frac{1}{8}((\bar{x}-5)^2+(\bar{y}-5)^2)}$$

which means that most of the concentration is found at the center of the domain. The following figures display the results.

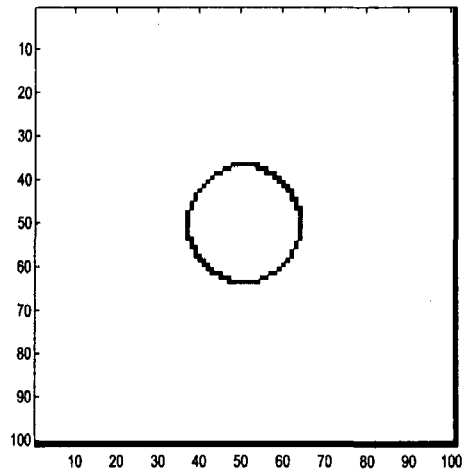

FIGURE 4.20 – Reaction diffusion tracking results for a critical value of 0.8 for $\tau = 0$.

Figure 4.20 gives the tracking result for the initial condition. Inside the region, the concentrations are at least 80% of the maximum concentration. We should expect that the concentrations will be decreasing as in the results for the heat equation. The difference in this example, however, is that the concentration may not decrease in the strict sense. For instance, consider an isolated grid point. At this point, the reaction forces the concentration to decrease in a manner proportional to the concentration itself. This may cause a situation where the concentration at this point is lower than surrounding concentrations. Thus, we have a local diffusion which takes place and the concentration at our grid point may actually increase during the next time step. This exact situation

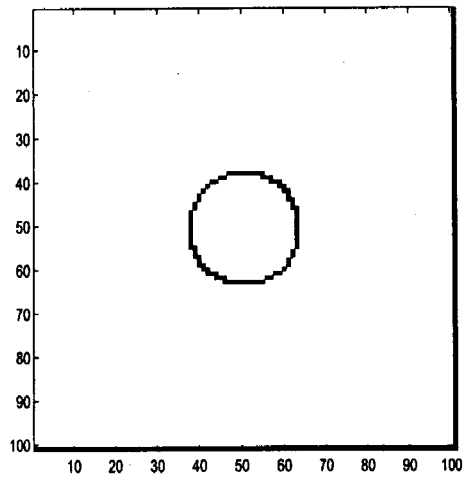

FIGURE 4.21 – Reaction diffusion tracking results for a critical value of 0.8 for $\tau = 0.1$.

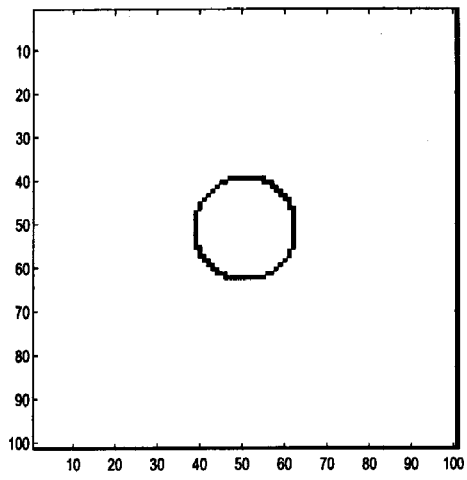

FIGURE 4.22 – Reaction diffusion tracking results for a critical value of 0.8 for $\tau = 0.2$.

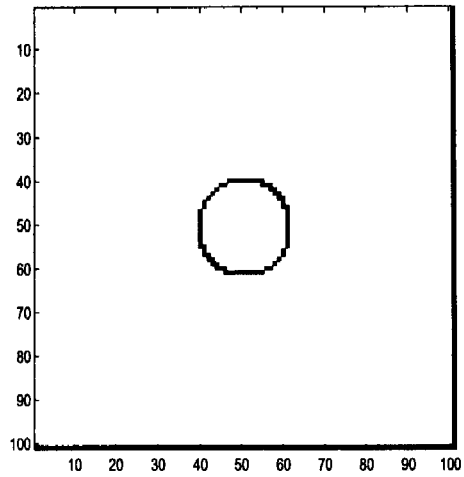

FIGURE 4.23 – Reaction diffusion tracking results for a critical value of 0.8 for $\tau = 0.3$.

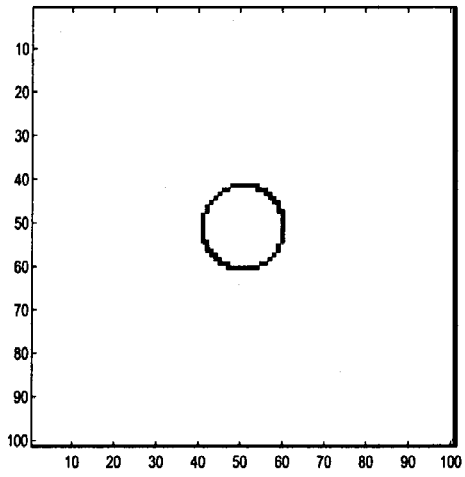

FIGURE 4.24 – Reaction diffusion tracking results for a critical value of 0.8 for $\tau = 0.4$.

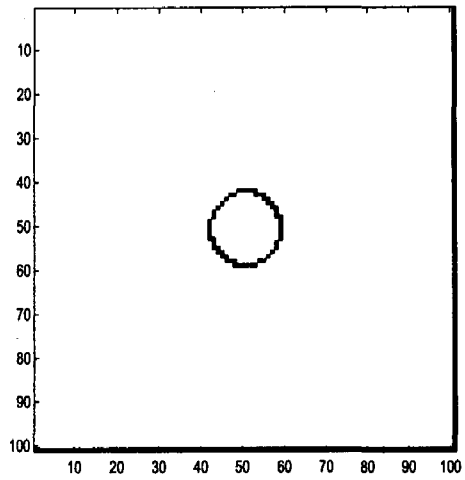

FIGURE 4.25 – Reaction diffusion tracking results for a critical value of 0.8 for $\tau = 0.5$.

may be seen from the remaining figures, although the tendency for the region to shrink is obvious. For Figures 4.21-4.25, we have shown the results of the tracking every 0.1 units of time.

CHAPTER 5
CONCLUSION AND DISCUSSION OF FUTURE WORK

Surprisingly, the linearized form of the active contours without gradient model gave very good results - at least for simple images. This model also does not suffer from the numerical difficulties of most other imaging models in that reconciliation is not required. Further, the convergence rate is very fast. This warrants further investigation with more complicated images and extension of the model into both color images and multi-dimensional "images". We have tested the linear model for images which are not simple - including the results in Figures 2.9, 2.10, 2.11 - and find results comparable to the nonlinear model.

After showing that the fourth order procedure does offer significant savings on computation time and resources, the next step would be to extend the procedure to color and multi-dimensional images and investigate the improvement over the second order model. We note here that using this procedure blurs the relative difference between initial conditions. The reason for this is that the fourth order model may be used to quickly obtain a reasonable segmentation of the image from any initial condition. This result may be smoothed by reconciliation. The overall result is that the Chan-Vese second order model is given a much better initial condition than may otherwise have been used.

Finally, the examples given for data tracking show that the method does have some utility in visualizing regions of interest. Of course, the obvious step would be to look at much more sophisticated examples where visualizing data may be problematic. Three dimensional spatial data could be easily visualized using such an approach. Further, the extension of the procedure to track multiple regions would allow bands of data to be tracked easily. We believe this type of visualization can be quite useful for more

complicated applications.

REFERENCES

[1] D. Adelsteinsson and J.A. Sethian, *The fast construction of extension velocities in level set methods*, Journal of Computational Physics **148** (1999), no. 1, 2–22.

[2] L. Ambrosio and V.M. Tortorelli, *Approximation of functionals depending on jumps by elliptic functionals via γ-convergence*, Communications on Pure and Applied Mathematics **XLIII** (1990), 999–1036.

[3] A. Amini, T. Weymouth, and R. Jain, *Using dynamic programming for solving variational problems in vision*, IEEE Trans. Pattern Analysis Machine Intelligence **12** (1990), 855–867.

[4] V.I. Arnold, *Geometrical methods in the theory of ordinary differential equations*, first ed., Springer-Verlag, New York, 1983.

[5] B. Aubert and P. Kornprobst, *Mathematical problems in image processing: Partial differential equations and the calculus of variations*, 2nd ed., Springer, New York, 2006.

[6] G. Aubert and L. Blanc-Feraud, *Some remarks on the equivalence between 2d and 3d classical snakes and geodesic active contours*, The International Journal of Computer Vision **34** (1999), no. 1, 19–28.

[7] G. Aubert and L. Vese, *A variational method in image recovery*, SIAM J. of Numer. Anal. **34** (1997), no. 5, 1948–1979.

[8] H.M. Soner Barles, G. and P.E. Souganidis, *Front propagation and phase field theory*, SIAM J. Control and Optimization **31** (March 1993), no. 2, 439–469.

[9] G. Bellitini and A. Coscia, *Approximation of functional depending on jumps and corners*, Bolletino della Unione Matematica Italiana **8** (1994), no. 1, 151–181.

[10] M. Boue and P. Dupuis, *Markov chain approximations for deterministic control problems with affine dynamics and quadratic cost in the control*, SIAM J. Numer. Anal. **3** (1999), no. 36, 667–695.

[11] A. Braides, *Mathematical problems in image processing: Partial differential equations and the calculus of variations*, Lecture Notes in Mathematics, vol. 1694 (New York), Springer-Verlag, 1998, pp. 215–232.

[12] A. Braides and G. Dal Maso, *Non-local approximation of the mumford-shah functional*, Calc. Var. Partial Differ. Eq. **5** (1997), no. 4, 293–322.

[13] M. Carriero, A. Leaci, and F. Tomarelli, *A second order model in image segmentation: Blake & zisserman functional*, Nonlinear differ. Eq. Appl. **25** (1996), 57–72.

[14] V. Caselles, F. Catte, T. coll, and F. Dibos, *A geometric model for active contours*, Numerische Mathematik **66** (1993), 1–31.

[15] V. Caselles, R. Kimmel, and G. Sapiro, *On geodesic active contours*, Proceedings of the 5th International Conference on Computer Vision (Boston), IEEE Computer Society Press, 1995, pp. 694–699.

[16] ———, *On geodesic active contours*, Int. J. Comput. Vis. **22** (1997), no. 1, 61–70.

[17] A. Chambolle, *Image segmentation by varational methods: Mumford and shah functional and the discrete approximation*, SIAM Journal of Applied Mathematics **55** (1995), no. 3, 827–863.

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

[19] ———, *A multiphase level set framework for image segmentation using the mumford shah model*, International Journal of Computer Vision **50** (2002), no. 3, 271–293.

[20] John M. Cochran and Yongzhi Xu, *Analysis of linearized version of active contours without gradient*, Applicable Analysis **8** (2008), no. 87, 967–985.

[21] L.D. Cohen, *On active contour models and balloons*, CVGIP: Image Understand. **53** (1991), 211–218.

[22] P. Covello and G. Rodrigue, *A generalized front marching algorithm for the solution of the eikonal equation*, Journal of Computation and Applied Mathematics **156** (2003), 371–388.

[23] B. Curwen and eds. A. Blake, *Dynamic contours: Real-time active splines*, Active Vision.

[24] Lawrence Evans, *partial differential equations*, first ed., American Mathematical Society, Providence, RI, 2002.

[25] M. Falcone and R. Ferretti, *Discrete time high-order schemes for viscosity solutions of hamilton-jacobi-bellman equations*, Numer. Math. **7** (1994), no. 3, 315–344.

[26] ———, *Semi-lagrangian schemes for hamilton-jacobi equations, discrete representation formulae and godunov methods*, J. Comput. Phys. **2** (2002), no. 175, 559–575.

[27] H. Scott Fogler, *Elements of chemical reaction engineering*, 3rd ed., Prentice Hall, New Jersey, 1999.

[28] M. Gobbino, *Finite difference approximation of the mumford-shah functional*, Communications on Pure Applied Mathematics **51** (1998), no. 2, 197–228.

[29] Jose Gomes and Olivier Faugeras, *Reconciling distance functions and level sets*, Journal of Visual Communication and Visual Representation **11** (2000), 209–223.

[30] J. Helmsen, E. Puckett, P. Colella, and M. Dorr, *Two new methods for simulating photolithography development in 3d*, Proc. SPIE **2726** (1996), 253–261.

[31] M. Kass, A. Witkin, and D. Terzopoulos, *Snakes: Active contour models*, First International Conference on Computer Vision (London), IEEE Computer Society Press, 1987, pp. 259–268.

[32] S. Kichenassamy, A. Kumar, P. Olver, A. Tannenbaum, and A. Yezzy, *Gradient flows and geometric active contour models*, Proc. ICCV.

[33] ———, *Conformal curvature flows: from phase transitions to active vision*, Archive for Rational Mechanics and Analysis **134** (1996), 275–301.

[34] Octave Levenspiel, *Chemical reaction engineering*, 3rd ed., John Wiley and Sons, New York, 1999.

[35] Randall J. Leveque, *Finite difference methods for ordinary and partial differential equations*, first ed., SIAM, Philadelphia, 2007.

[36] R. Malladi, J.A. Sethian, and B.C. Vemur, *Evolutionary fronts for topology-independent shape modeling and recovery*, Proceedings of the 3rd European Conference on Computer Vision (Stockholm) (J.O. Eklundh, ed.), Springer-Verlage, 1994, pp. 3–13.

[37] ———, *Shape modeling with front propagation: A level sete approach*, IEEE Transactions on Pattern Analysis and Machine Intelligence **17** (1995), no. 2, 158–175.

[38] T. McInerney and D. Terzopoulos, *Topologically adaptable snakes*, Proc. ICCV (1995), 840–845.

[39] D. Mumford and J. Shah, *Optimal approximations by piecewise smooth functions and associated variational problems*, Communications on Pure and Applied Mathematics **42** (1989), 577–684.

[40] J.D. Murray, *Mathematical biology i: An introduction*, 3rd ed., Springer, New York, 2002.

[41] ———, *Mathematical biology ii: Spatial models and biomedical applications*, 3rd ed., Springer, New York, 2002.

[42] S. Osher and J.A. Sethian, *Fronts propagating with curvature dependent speed*, Journal of Computational Physics **79** (1988), 12–49.

[43] L.C. Polymenakos, D.P. Bertsekas, and J.N. Tsitsiklis, *Implementation of efficient algorithms for globally optimal trajectories*, IEEE Transactions on Automatic Control **42** (1998), no. 2.

[44] E. Rouy and A. Tourin, *A viscosity solution approach to shape-from-shading*, SIAM J. Num. Anal. **29** (1992), no. 3, 867–884.

[45] L. Rudin, S. Osher, and E. Fatemi, *Nonlinear total variation based noise algorithms*, Phys. D. **60** (1992), 250–268.

[46] J.A. Sethian, *Recent numerical algorithms for hypersurfaces moving with curvature-dependent speed: Hamilton-jacobi equations and conservation laws*, J. Differential Geometry **31** (1990), 131–136.

[47] ———, *A fast marching level set method for monotonically advancing fronts*, Proc. Nat. Acad. Sci. **93** (1996), no. 4, 1592–1595.

[48] K. Siddiqi, Y. Lauziere, A. Tannenbaum, and S. Zucker, *Area and length minimizing flows for shape segmentation*, IEEE Transactions on Image Processing **7** (1998), 433–443.

[49] J.N. Tsitsiklis, *Efficientt algorithms for globally optimal trajectories*, IEEE Transactions on Automatic Control **40** (1995), no. 9, 1528–1538.

[50] C. Xu and J.L. Prince, *Snakes, shapes, and gradient vector flow*, IEEE Transactions on Image Processing **7** (1998), no. 3, 359–369.

[51] H.K. Zhao, S. Osher, B. Merriman, and M.Kang, *Implicit and non-parametric shape reconstruction from unorganized points using variational level set method*, Computer Vision and Image Understanding **80** (2000), no. 3, 295–319.

[52] Hongkai Zhao, *A fast sweeping method for eikonal equations*, Math. Comp. **74** (2004), no. 250, 603–627.

APPENDIX I
ACTIVE CONTOURS WITHOUT GRADIENT PROGRAM CODE

This program calculates the results for the Chan-Vese Active Contours Without Gradient Model. The code uses an iterative approach.

I.1 Main Code

```
function [time,energy_vector,loop_error_vector,iterate_vector]=

ac_jmc_second_order(mu,dt,U,lambda1,lambda2,nu,color,energy_tol,

tol,overall_tol,use_special_ic,special_ic,stop_iterations
```

We want to time the calculations, so we use the commands tic-toc.

```
tic
```

We need to know the size of the image U. We will also set many of the variables which will be used based upon that size.

```
[Y,X] = size(U);

Energy = zeros(10,1);

loop_error = zeros(10,1);

loop_iterates = zeros(10,1);

phi = zeros( size(U) );

phi_rec = zeros( size(U) );
```

```
D1 = zeros( Y+2,X+2 );

D2 = zeros( Y+2,X+2 );

Gradient = zeros( size(U) );
```

The variable h will be the spatial step size. Since we are working on images, the step size will be one.

```
h = 1;
```

The variable DD represents the Dirac Delta function evaluations - recall that we are using a regularized version of the Dirac Delta function.

```
DD = zeros( size(U) );
```

The following variables are used for the integration involved in finding the two constants c_1 and c_2.

```
Energy_integrand = zeros( size(U) );

prod = zeros( size(U) );

prod1 = zeros( size(U) );

heavy = zeros( size(U) );

heavym1 = zeros( size(U) );
```

The next code allows us to specify either internally or externally defined initial conditions. This allows for flexibility in the choice of the initial condition.

```
if use_special_ic == 1

    phi = special_ic;

else
```

```
    for i=1:Y
        for j=1:X
            phi(i,j) = initial_condition((i-1)*h,(j-1)*h);
        end
    end
end
```

We want to locate the *approximate* location of the level set curve. The following code does this by checking for sign changes between successive pixels.

```
Zerolevelinitial = phi;
for j=1:(X-1)
    for i=1:(Y-1)
        if or( or(and( phi(i,j)>0,phi(i+1,j)<0),
and(phi(i,j)<0,phi(i+1,j)>0)),or(and( phi(i,j)>0,phi(i,j+1)<0),
and(phi(i,j)<0,phi(i,j+1)>0)))
            Zerolevelinitial(i,j)=0;
        else
            Zerolevelinitial(i,j)=255;
        end
    end
end
```

Next, display the results of the level set location.

```
figure

imagesc(Zerolevelinitial), colormap(gray)
```

Next, we initialize several variables which keep track of various program code. The error is the overall stopping criteria and is eventually compared to the overall tolerance. The

```
relative_error
```

represents the error scaled by the previous iteration and k keeps track of the number of iterations the global program executes.

```
error = 1000;

k=0;
```

If we reach the desired tolerance, we want to immediately stop the program. The following variable is then initialized here and checked further in the code.

```
stop_program = 0;
```

We now create the reflected boundaries by increasing the size of the initial condition, shifting the initial condition to the lower right, and reassigning the boundaries accordingly.

```
phi_non_reflected = phi;

phi(Y+1,:)=0;phi(Y+2,:)=0;phi(:,X+1)=0;phi(:,X+2)=0;

phi = circshift(phi,[1,1]);

newX = X + 2;newY = Y + 2;

phi(1,:) = phi(2,:);

phi(newY,:) = phi(newY-1,:);
```

```
phi(:,1) = phi(:,2);

phi(:,newX) = phi(:,newX-1);
```

At this point, the original matrix is now of size $(Y+2, X+2)$ with the original initial condition found in $(2,2)$ to $(Y+1, X+1)$. To speed the program execution, we can vectorize the formulas. We only update values in the center of the matrix (i.e., for rows $2 \ldots Y-1$ and columns $2 \ldots X-1$.

```
I = 2:(newY-1);

J = 2:(newX-1);

a = mu/h^2;
```

Now we begin the global iteration which will calculate the steady-state of the pde. The variables $D1$ and $D2$ represent the denominators of the discretization for the divergence term.

```
while (error >= overall_tol)&&(stop_program ~= 1)

    D1(I,J) = sqrt( (phi(I,J+1) - phi(I,J)).^2/h^2 +

                              (phi(I+1,J) - phi(I-1,J)).^2/(2*h)^2);

    D2(I,J) = sqrt( (phi(I,J+1) - phi(I,J-1)).^2/(2*h)^2 +

                             (phi(I+1,J) - phi(I,J)).^2/h^2);
```

Again, we wish to keep the previous iteration for comparison in order to calculate the error at the current iteration.

```
    phi_temp = phi_non_reflected;
```

The functions $H(\phi)$ and $(1 - H(\phi))$ are then calculated using the following code.

```
    heavy = 1/2*(1+(2/pi).*atan(phi_temp./h));
```

```
heavym1 = 1-heavy;
```

To find the products $u_0 H(\phi)$ and $u_0(1 - H(\phi))$, we need to convert the image to double precision which requires adding one due to the way Matlab handles 8-bit and 16-bit images.

```
prod = (double(U)+1) .* heavy;

prod1 = (double(U)+1) .* heavym1;
```

The values for c_1 and c_2 are then given by

```
c1 = jmc_trapezoid(prod,h)/jmc_trapezoid(heavy,h);

c2 = jmc_trapezoid(prod1,h)/jmc_trapezoid(heavym1,h);
```

We next make certain that the denominators have been reflected as well since the above calculation will not produce proper values for the boundaries.

```
D1(1,:) = D1(2,:);

D1(newY,:) = D1(newY-1,:);

D1(:,1) = D1(:,2);

D1(:,newX) = D1(:,newX-1);

D2(1,:) = D2(2,:);

D2(newY,:) = D2(newY-1,:);

D2(:,1) = D2(:,2);

D2(:,newX) = D2(:,newX-1);
```

The Dirac Delta and fitting terms are then calculated using the next code.

```
DD(I,J) = 1/(pi*h) * 1./(1+ (phi(I,J)./h).^2);
```

```
    Q(I-1,J-1) = -nu-lambda1.*(double(U(I-1,J-1))+1-c1).^2+
 lambda2.*(double(U(I-1,J-1))+1-c2).^2;
```

If this is the first time through the code, calculate the energy associated with the initial condition.

```
    if k==0
        Gradient(I-1,J-1)=sqrt((phi(I,J+1)-phi(I,J-1)).^2/(2*h)^2+
(phi(I+1,J) - phi(I-1,J)).^2/(2*h)^2);
        Energy_integrand(I-1,J-1)=mu*DD(I-1,J-1).*Gradient(I-1,J-1)+
lambda1*( double(U(I-1,J-1))+1-c1).^2 .*
jmc_heaviside(phi_non_reflected(I-1,J-1))+
lambda2*( double(U(I-1,J-1))+1-c2).^2 .*
(1-jmc_heaviside(phi_non_reflected(I-1,J-1)));
```

Now assign that energy to the first entry in the variable Energy.

```
        Energy(1,1) = jmc_trapezoid(Energy_integrand,h);
    end
```

Now we call the iterative program code - shown in the next section. This code calculates the grid function for the next time step.

```
    [phi,loop_error(k+1),loop_iterates(k+1)] =
ac_jmc_cv_iterate(phi, newX, newY, I, J, DD, D1, D2, Q, dt, a, tol);
```

The variable phi_non_reflected is the version of ϕ without reflections. This is now found and used to calculate the energy associated with this new estimate for ϕ.

```
    phi_non_reflected = phi;

    phi_non_reflected(:,newX)=[];

    phi_non_reflected(newY,:)=[];

    phi_non_reflected(:,1) = [];

    phi_non_reflected(1,:) = [];

    Gradient(I-1,J-1) = sqrt( (phi(I,J+1)-phi(I,J-1)).^2/(2*h)^2 +

(phi(I+1,J) - phi(I-1,J)).^2/(2*h)^2);

    Energy_integrand(I-1,J-1) = mu*DD(I-1,J-1).*Gradient(I-1,J-1) +

lambda1*( double(U(I-1,J-1))+1-c1).^2 .*

heaviside(phi_non_reflected(I-1,J-1)) +

lambda2*( double(U(I-1,J-1))+1-c2).^2 .*

(1-heaviside(phi_non_reflected(I-1,J-1)));

    Energy(k+2,1) = jmc_trapezoid(Energy_integrand,h);
```

We may wish to base the stopping criteria on energy rather than the actual values of the grid function for ϕ. If the flag `energy_tol` is zero, we let the error be defined as the maximum change in the grid function between the two successive iterations.

```
    if energy_tol == 0

        error = max( max( abs(phi_non_reflected-phi_temp)));

        min_error = min( min( abs(phi_non_reflected-phi_temp)));
```

Otherwise, we will use the `energy_difference` as the stopping criteria.

```
else

    energy_difference = abs( Energy(k+2,1) - Energy(k+1,1) )/

            max(Energy(k+2,1),Energy(k+1,1));

    if energy_difference <= energy_tol

        error = overall_tol - 1;

        actual_energy_difference =

                    abs( Energy(k+2,1) - Energy(k+1,1) )

    else

        error = overall_tol + 1;

    end

end
```

As we have seen, the code may require many thousand iterations to converge. We want to save the results at different times so that we can obtain some results if something unforeseen causes the code to stop abruptly. The code as shown saves every ten iterations.

```
if (mod(k,10) == 0)&&(k ~= 0)

    k

    save 'phi_output.out' phi_non_reflected -ASCII -double

    save 'energy_output.out' Energy -ASCII -double

    save 'error.out' error -ASCII -double

end
```

Now, we can increment the global iterations by one and check to see if we have reached the maximum iterations desired by the user. If so, we stop the program.

```
    k=k+1;

    if k==stop_iterations

        stop_program = 1;

    end

end
```

Next, we show the user the final errors and set the function ϕ to be the last calculated result.

```
min_error

error

relative_error = max( max( abs(

  (phi_non_reflected-phi_temp)./ phi_temp) ))

if energy_tol ~= 0

    energy_difference

end

phi = phi_non_reflected;
```

The rest of the code is used to locate the level set and show various results to the user.

```
Zerolevel = phi;

for j=1:(X-1)
```

```
    for i=1:(Y-1)
        if or( or(and( phi(i,j)>0,phi(i+1,j)<0),
                        and(phi(i,j)<0,phi(i+1,j)>0)),
or(and( phi(i,j)>0,phi(i,j+1)<0),
                            and(phi(i,j)<0,phi(i,j+1)>0)))
            Zerolevel(i,j)=0;
        else
            if phi(i,j)==0
                Zerolevel(i,j)=0;
            else
                Zerolevel(i,j)=255;
            end
        end
    end
end
```

We may also want to show the level set overlayed on top of the original image. This is accomplished using the following.

```
if color ~= (-1)
    B=U;
    for j=1:X
```

```
        for i=1:Y
            if Zerolevel(i,j)==0
                B(i,j)=color;
            end
        end
    end
    figure
    imagesc(B),colormap(gray)
end
```

Now, show the results including the level set and ϕ. Further, assign the output values for the level set function ϕ, the energy, and errors.

```
figure
imagesc(Zerolevel), colormap(gray)
figure
imagesc(phi),colormap(gray)
figure
mesh(phi)
energy_vector = Energy;
loop_error_vector = loop_error;
iterate_vector = loop_iterates;
```

Stop the internal timing.

```
time=toc;
```

End the program.

```
end
```

I.2 Iteration Code

```
 function [result,end_error,number_of_iterates] =
ac_jmc_cv_iterate(phi, newX, newY, i, j, DD, D1, D2, Q, dt, a, tol)
```

We begin with a guess for the solution at this iteration.

```
guess = phi;
```

The following terms are ratios of the denominators. In order to prevent division by zero - and the need to check every entry in the matrices - we can add a very small number in the form of ϵ^2. Here, ϵ is the machine tolerance - or the difference between two successive real numbers.

```
D1r = 1./(eps^2+D1);
```

```
D2r = 1./(eps^2+D2);
```

We want the code to execute at least once, so initialize the check variable to something larger than the stopping tolerance of the code.

```
check = tol + 1;
```

To make the code easier to read, we store the coefficients in a new matrix, and in order to keep track of the number of iterations for each loop of the iterative code, we initialize a variable l.

```
coefficient(i,j) = 1./(1+a*dt*DD(i,j).*(D1r(i,j)+D1r(i,j-1)+
```

```
                                    D2r(i,j)+D2r(i-1,j)));

l = 1;
```

The iterative portion of this program now begins by saving the original guess. The reason for this is so that we can compare the old guess to the new calculated guess in order to give an error estimate (i.e., a check for the stopping tolerance).

```
while (check >= tol)

    guessold = guess;
```

Now calculate the new values for the function guess.

```
    guess(i,j) = coefficient(i,j).*( phi(i,j)+dt*DD(i,j).*

       (a*(guess(i,j+1).*D1r(i,j) + guess(i,j-1).*D1r(i,j-1) +

        guess(i+1,j).*D2r(i,j) + guess(i-1,j).*D2r(i-1,j)) +

        Q(i-1,j-1) ));
```

We are using reflections to handle boundary terms, so now reflect the given function around its boundary.

```
    guess(1,:)=guess(2,:);

    guess(newY,:)=guess(newY-1,:);

    guess(:,1)=guess(:,2);

    guess(:,newX)=guess(:,newX-1);
```

Next, we want to calculate the difference in the two functions - the old guess and the new one.

```
    guess_diff = abs(guess-guessold);
```

Of course, we do not want to count the reflected values in the error calculation, so we need to eliminate the boundary.

```
guess_diff(newY,:) = [];

guess_diff(:,newX) = [];

guess_diff(1,:) = [];

guess_diff(:,1) = [];
```

Now, we can calculate the error and increment the number of iterations.

```
check = max( max( guess_diff ) );

l = l + 1;
```

End the while loop.

```
end
```

The returned function ϕ is then the final guess, and we may also want to return the total iterations that were required and the final error.

```
result = guess;

end_error = check;

number_of_iterates = l;
```

End the program.

```
end
```

APPENDIX II
LINEARIZED MODEL CODE

The program presented in this chapter details the calculation used for the linearized model. The code is analagous to the code in the previous section with minor differences.

II.1 Main Code

```
function [phi_out,zerolevel_out] =

   linear_model(U,dt,N,mu,color,ireconcile,reconcile_iterations)
```

Usually, we assign λ_1 and λ_2 to be one and as usual, the spatial step size is one.

```
lambda1 = 1;

lambda2 = 1;

h = 1;
```

Find the size of the input image and define various functions.

```
[Y,X] = size(U);

phi = zeros( size(U) );

prod = zeros( size(U) );

prod1 = zeros( size(U) );

heavy = zeros( size(U) );
```

```
heavym1 = zeros( size(U) );

Zerolevel = zeros( size(U) );
```

We begin timing the calculations here and show the image.

```
tic

figure

imagesc(U),colormap(gray)
```

Next, define the initial condition. The code shows the function to be used. This could be modified or changed as needed, but this particular initial condition gives good results.

```
[Yvals,Xvals] = meshgrid(1:1:Y,1:1:X);

phi = 80 - sqrt( (Xvals-100.5).^2 + (Yvals-100.5).^2 );
```

Again, we wish to locate the approximate level set curve.

```
for j=1:(X-1)

    for i=1:(Y-1)

        if or( or(and( phi(i,j)>0,phi(i+1,j)<0),

                and(phi(i,j)<0,phi(i+1,j)>0)),

                or(and( phi(i,j)>0,phi(i,j+1)<0),

                and(phi(i,j)<0,phi(i,j+1)>0)))

            Zerolevelinitial(i,j)=0;

        else

            Zerolevelinitial(i,j)=255;
```

```
        end
    end
end
figure
imagesc(Zerolevelinitial), colormap(gray)
```

Define the error, relative error, and initialize the iteration step. Further, we will save the current function ϕ for use later in the error calculations.

```
error = 1000;
relative_error = 0;
k=1;
phi_non_reflected = phi;
```

As before, we will use reflections to handle the boundary terms. The reflections are found in a similar way

```
phi(Y+1,:)=0;phi(Y+2,:)=0;phi(:,X+1)=0;phi(:,X+2)=0;
phi = circshift(phi,[1,1]);
newX = X + 2;newY = Y + 2;
phi(1,:) = phi(2,:);
phi(newY,:) = phi(newY-1,:);
phi(:,1) = phi(:,2);
phi(:,newX) = phi(:,newX-1);
```

Again, we wish to use vectorization to help speed program execution.

```
I = 2:(newY-1);

J = 2:(newX-1);
```

Now, we call the function which will calculate the coefficients as discussed in the text.

```
[A00,A01,A02,A10,A11,A12,A20,A21,A22,first_terms_in_W] =

 linear_coefficients(X,Y,h);
```

Begin the loop by checking to see if the user wishes to reconcile the result at this iteration.

```
while (k<=N)

    if ireconcile == 1

        if ( mod(reconcile_iterations,k)==0 )
```

Reconcile the function to a signed distance function.

```
            phi = twoD_eikonal(phi, 0.1, 1000);
```

And update the reflections as necessary.

```
            phi_non_reflected = phi;

            phi_non_reflected(:,newX) = [];

            phi_non_reflected(newY,:) = [];

            phi_non_reflected(:,1) = [];

            phi_non_reflected(1,:) = [];
```

If we reconcile, we want to take that into account when we calculate the error; hence, the previous iteration becomes this reconciled version.

```
        phi_temp = phi_non_reflected;

    end

else

    phi_temp = phi_non_reflected;

end
```

Next, we calculate the values for c_1 and c_2 using the same methods as above.

```
heavy = 1/2*(1+(2/pi).*atan(phi_temp./h));

heavym1 = 1-heavy;

prod = (double(U)+1) .* heavy;

prod1 = (double(U)+1) .* heavym1;

c1 = jmc_trapezoid(prod,h)/jmc_trapezoid(heavy,h);

c2 = jmc_trapezoid(prod1,h)/jmc_trapezoid(heavym1,h);
```

The values for the function W may now be found.

```
W(I-1,J-1) = first_terms_in_W(I-1,J-1)-

      lambda1.*(double(U(I-1,J-1))+1-c1).^2+

      lambda2.*(double(U(I-1,J-1))+1-c2).^2;
```

Update the values of the function using the numerical scheme and the linear coefficients found above.

```
phi(I,J) = phi(I,J) + dt*( mu*(

        A00(I-1,J-1).*phi(I-1,J-1)+A01(I-1,J).*phi(I-1,J)+
```

```
A02(I-1,J+1).*phi(I-1,J+1)+A10(I,J-1).*phi(I,J-1)+

A11(I,J).*phi(I,J)+A12(I,J+1).*phi(I,J+1)+

A20(I+1,J-1).*phi(I+1,J-1)+A21(I+1,J).*phi(I+1,J)+

A22(I+1,J+1).*phi(I+1,J+1) )+W(I-1,J-1) );
```

Next, reflect the function as needed to assign the boundary values and define a non-reflected version for use in the error calculation.

```
phi(1,:) = phi(3,:);

phi(newY,:) = phi(newY-2,:);

phi(:,1) = phi(:,3);

phi(:,newX) = phi(:,newX-2);

phi_non_reflected = phi;

phi_non_reflected(:,newX)=[];

phi_non_reflected(newY,:)=[];

phi_non_reflected(:,1) = [];

phi_non_reflected(1,:) = [];
```

Now calculate the error and relative error.

```
error = max( max( abs(phi_non_reflected-phi_temp)));

relative_error = max( max( abs(

    (phi_non_reflected-phi_temp)./phi_temp) ) );
```

Increase the iteration number by one.

```
    k=k+1;
end
```

If we have exited the loop, show the user the final error and relative error. And, as above, show the user results including the final level set and ϕ.

```
error
relative_error
phi = phi_non_reflected;
Zerolevel = phi;
for j=1:(X-1)
    for i=1:(Y-1)
        if or( or(and( phi(i,j)>0,phi(i+1,j)<0),
                  and(phi(i,j)<0,phi(i+1,j)>0)),
                  or(and( phi(i,j)>0,phi(i,j+1)<0),
                  and(phi(i,j)<0,phi(i,j+1)>0)))
            Zerolevel(i,j)=0;
        else
            if phi(i,j)==0
                Zerolevel(i,j)=0;
            else
                Zerolevel(i,j)=255;
```

```
            end
        end
    end
end
if color ~= (-1)
    B=U;
    for j=1:X
        for i=1:Y
            if Zerolevel(i,j)==0
                B(i,j)=color;
            end
        end
    end
    figure
    imagesc(B),colormap(gray)
end
figure
image(Zerolevel),colormap(gray)
figure
imagesc(phi),colormap(gray)
```

```
figure

surf(Xvals,Yvals,phi)
```

Finally, assign the output function ϕ and the output zero level curve.

```
phi_out = phi;

zerolevel_out = Zerolevel;
```

End the code timer.

```
toc
```

End the program.

```
end
```

II.2 Determination of Coefficients Code

The code presented in this section is self-explanatory. The entire purpose of this function is to calculate the various coefficients used in the linearized model. One consequence of using an actual function for the initial condition is that the various derivatives may be found *exactly* without the introduction of various numerical error created by approximations. The code also simplifies significantly, as can be seen below.

```
function [A00,A01,A02,A10,A11,A12,A20,A21,A22,first_terms_in_W] =

 linear_coefficients(X,Y,h)
```

We calculate the various derivatives as needed.

```
[Yval, Xval] = meshgrid(1:1:Y,1:1:X);

Denom = sqrt( (Xval-100.5).^2+(Yval-100.5).^2 );

tphi_x = -(Xval-100.5)./Denom;
```

```
tphi_y = -(Yval-100.5)./Denom;

tphi_xx = ( (Xval-100.5)-Denom.^2 )./( Denom.^3 );

tphi_yy = ( (Yval-100.5)-Denom.^2 )./( Denom.^3 );

tphi_xy = ( (Xval-100.5).*(Yval-100.5) )./( Denom.^3 );
```

The Hessian and laplacian can now be computed accurately along with various other parameters as seen in the text.

```
tgrad_hessian_grad = 2.*tphi_x.*tphi_y.*tphi_xy + tphi_x.^2 .*
                     tphi_xx + tphi_y.^2 .* tphi_yy;

tlaplacian = tphi_xx + tphi_yy;

tP1 = -tlaplacian.*tphi_x + 3*tphi_x.*tgrad_hessian_grad -
                     2*(tphi_x.*tphi_xx+tphi_y.*tphi_xy);

tP2 = -tlaplacian.*tphi_y + 3*tphi_y.*tgrad_hessian_grad -
                     2*(tphi_x.*tphi_xy+tphi_y.*tphi_yy);

tfirst_terms_in_W = tlaplacian - (tP1.*tphi_x+tP2.*tphi_y);
```

We also need reflected versions of these functions, however.

```
P1 = reflected_function(tP1,X,Y);

P2 = reflected_function(tP2,X,Y);

phi_x = reflected_function(tphi_x,X,Y);

phi_y = reflected_function(tphi_y,X,Y);

first_terms_in_W = tfirst_terms_in_W;
```

The linear coefficients may now be calculated according to their associated definitions.

```
A00 = -phi_x.*phi_y/(2*h^2);

A01 = -(P2/(2*h)+(1+phi_y.^2)/h^2);

A02 = -A00;

A10 = -(P1/(2*h)+(1+phi_x.^2)/h^2);

A11 = -(2/h^2)*(2+phi_x.^2+phi_y.^2);

A12 = -A10;

A20 = A02;

A21 = A01;

A22 = A00;
```

End the program.

```
end
```

CURRICULUM VITAE

John M. Cochran

Department of Mathematics

University of Louisville

Louisville KY

cochrajm@yahoo.com

Education

PhD Mathematics

University of Louisville (May 2009)

Louisville, Kentucky

MA Mathematics

University of Louisville (May 2008)

Louisville, Kentucky

MS Chemical Engineering

Rose-Hulman Inst. of Tech. (May 1999)

Terre Haute, Indiana

BS Chemical Engineering and Mathematics

Rose-Hulman Inst. of Tech. (May 1997)

Terre Haute, Indiana

Experience

Thermal Design Engineer

Vogt-NeM, Inc.

Louisville, Kentucky

1999-2001

- Responsible for design of heat recovery steam generators including heat transfer surface, piping, drums, and insulation.

Summer Intern

Vogt-NeM, Inc.

Louisville, Kentucky

1998

- Gave support for Sales Price Estimation Programs.

Skills

Extensive knowledge of Maple, Mathematica, Microsoft Office, Matlab

Working knowledge of Pascal, C++, Java, Latex

Publications

John M. Cochran and Yongzhi Xu. *Analysis of Linearized Version of Active Contours Without Gradient. Applicable Analysis*,87:8, 967-985

John M. Cochran and Yongzhi Xu. *Multiphase Image Segmentation Using Linearized Active Contours Without Gradient Model. In preparation.*

Presentations

John M. Cochran, *Linearized Active Contours*, Annual Meeting of Indiana Section of MAA, Terre Haute, IN, October 2008

John M. Cochran, *Data Tracking Using Mathematical Imaging*, Annual Meeting of Kentucky Section of MAA, Bowling Green, Kentucky, March 2008

John M. Cochran, *Norm Estimates for Navier-Stokes Equations*, Course: Problems in Applied Analysis, Louisville, Kentucky, October 2006

Affiliations

American Institute of Chemical Engineers (AIChE), American Mathematical Society (AMS), Mathematical Association of America (MAA)

Teaching Experience

University of Louisville

Primary Instructor for:

MA111 College Algebra, MA112 Trigonometry, MA190 Precalculus, MA180 Elements of Calculus

Recitation Instructor for:

MA111 College Algebra, MA107 Finite Mathematics

Awards and Honors	Ken F. and Sandra S. Hohman Graduate Fellowship (University of Louisville 2007), Pi Mu Epsilon (Rose-Hulman 1994-1997), Omega Chi Epsilon (Rose-Hulman 1996-1997)
Activities	President of Pi Mu Epsilon (1996-1997), Treasurer of Pi Mu Epsilon (1995-1996), Counselor/Tutor for Fast Track Calculus summer program (1994-1997), Tutored Mathematics/Chemistry in Rose-Hulman Learning Center (1994-1997), Treasurer of Rose-Hulman Fencing Club (1994-1995)