ELEC 639B Selected Topics in Image Processing

Variational PDE Models and Applications in Image Processing

LECTURE NOTES

Wu-Sheng Lu

Department of Electrical and Computer Engineering
University of Victoria

Office: EOW 427
Voice: 721-8692
E-mail: wslu@ece.uvic.ca
Web: http://www.ece.uvic.ca/~wslu/639B/index.html
December 2007

CONTENTS

References

The references listed below were utilized to prepare the notes. Individual chapters cite additional references which will be given at the end of the chapters.

[1] G. Aubert and P. Kornprobst, *Mathematical Problems in Image Processing: Partial Differential Equations and Calculus of Variations*, Springer, 2002.

[2] T. F. Chan and J. Shen, *Image Processing and Analysis: Variational, PDE, Wavelet, and Stochastic Methods*, SIAM, 2005.

[3] G. Sapiro, *Geometric Partial Differential Equations and Image Analysis*, Cambridge University Press, 2001.

[4] R. Kimmel, *Numerical Geometry of Images: Theory, Algorithms, and Applications*, Springer, 2004.

[5] S. J. Osher amd R. P. Fedkiw, *Level Set Methods and Dynamic Implicit Surfaces*, Springer 2002.

[6] I. M. Gelfand and S. V. Fomin, *Calculus of Variations*, Dover, 2000.

[7] J. W. Thomas, *Numerical Partial Differential Equations: Finite Difference Methods*, Springer, 1995.

[8] L. Rudin, S. Osher, and E. Fetami, "Nonlinear total variation based noise removal algorithm," *Physica D*, vol. 60, pp. 259-268, 1992.

[9] L. Rudin and S. Osher, "Total variation based image restoration with free local constraints," Proc. ICIP, vol. 1, pp. 31-35, 1994.

[10] T. F. Chan and S. Esedoglu, "Aspects of total variation regularized L^1 function approximation," *SIAM J. Applied Math.*, vol. 65, no. 5, pp. 1817-1837, 2005.

[11] P. Perona and J. Malik, "Scale-space and edge detection using anisotropic diffusion," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 12, pp. 629-639, July 1990.

[12] F. Catte, P.-L. Lions, J.-M. Morel, and T. Coll, "Image selective smoothing and edge detection by nonlinear diffusion," *SIAM J. Numerical Analysis*, vol. 29, no. 1, pp. 182-193, Feb. 1992.

[13] J. Weickert, B. M. ter H. Romeny, and M. A. Viergever, "Efficient and reliable schemes for nonlinear diffusion filtering," *IEEE Trans. Image Processing*, vol. 7, no. 3, pp. 398-410, March 1998.

[14] M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: Active contour models," *Int. J. Computational Vision*, vol. 1, pp. 321-331, 1988.

[15] V. Caselles, R. Kimmel, and G. Sapiro, "On geodesic active contours," *Int. J. Computational Vision*, vol. 22, no. 1, pp. 61-79, 1997.

[16] D. Mumford and J. Shah, "Optimal approximation by piecewise smooth functions and associated variational problems," *Communications on Pure and Applied Mathematics*, vol. 42, pp. 577-685, 1989.

[17] T. F. Chan and L. Vese, "Active contours without edges," *IEEE Trans. Image Processing*, vol. 10, pp. 266-277, Feb. 2001.

[18] S. Osher and J. A. Sethian, "Fronts propagating with curvature –dependent speed: Algorithms based on Hamilton-Jacobi formulation," *J. Computational Physics*, vol. 79, pp. 12-49, 1988.

[19] T. F. Chan, J. Shen, and L. Vese, "Variational PDE models in image processing," *Notices of AMS*, vol. 50, no. 1, pp. 14 – 26, Jan. 2003.

[20] UCLA Imagers homepage, http://www.math.ucla.edu/~imagers/.

Chapter 1 INTRODUCTION

1.1 Digital Images

In this course we shall adopt the view point that a digital image comes from a continuous world and is a sampled-and-quantized version of an analog image. The basic idea in the acquisition of a digital image is to superimpose a regular grid on an analog image and to assign a discrete (i.e. finite precision) number to each square of the grid, called a pixel, in accordance with, e.g., the average brightness (which is often called gray level) in that square.

The most common pixel representation of digital images uses unsigned bytes from 0 to 255. In the case of color digital images, a pixel is described typically by three channels (bands) for red, green, and blue components. Throughout we only consider gray-scale images.

The resolution of still digital images in each dimension has reached the range of 4000 to 5000 these days, but typical resolution for simulation studies in academic research remains in the range of 256 to 1024. Image sizes in video data vary from 720×576 for standard format to 1920×1440, or higher for high definition. For medical imagery, functional MRI images can be as low as 128×128.

1.2 Image Models

1.2.1 *Image Processing as an Input-Output System*

An image from a real world scene is usually composed of a wide variety of structures that possibly include textures, progressive or sharp contours, and fine objects. Directly connected to image processing are two dual fields in contemporary computer science: computer vision and computer graphics. Vision (whether machine or human) tries to reconstruct the 3-D world from observed 2-D images, while graphics pursues the opposite direction by designing suitable 2-D scene images to simulate our 3-D world. Image processing is the crucial middle way connecting the two. Abstractly, image processing can be considered as an input-output system.

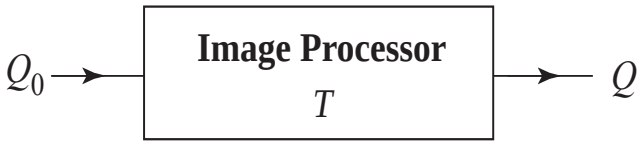


Figure 1.1

Here T denotes a typical image processor: for example, denoising, deblurring, segmentation, compression, or inpainting. The input data Q_0 can represent an observed or measured single image or image sequence, and the output $Q = (q_1, q_2, \cdots)$ contains all the targeted image *features*. For example, the human visual system can be considered as a highly involved multilevel image processor T. The input Q_0 represents the image sequence that is constantly projected onto the retina. The output vector Q contains all the major features that are important to our daily life, from the low level ones such as relative orders, shapes, and grouping rules to high-level feature parameters that help classify or identify various patterns and objects.

Typical image processing problems include [19]:

T	Q_0	Q
denoising + deblurring	$u_0 = Hu + w$	clear and sharp u
inpainting	$u_0 \|_{\Omega \setminus D}$	entire image $u \|_{\Omega}$
segmentation	u_0	"objects" $[u_k, \Omega_k]$, $k = 1, 2, \ldots$
scale-space	u_0	multiscale images $\{u_{\lambda_1}, u_{\lambda_2}, \cdots\}$
motion estimation	$\{u_0^{(1)}, u_0^{(2)}, \cdots\}$	optical flows $\{\vec{v}^{(1)}, \vec{v}^{(2)}, \cdots\}$

where the word *inpainting* is an artistic synonym for image interpolation, originated among museum restoration artists who manually restore cracked ancient paintings. The figure below shows an example of Mumford-Shah inpainting for text removal [19].

The two main ingredients of image processing are the input Q_0 and the processor T. As a result, the two key issues that have been driving mainstream mathematical research on image processing are
(a) the modeling and representation of the input visual data Q_0, and
(b) the modeling of the processing operators T. Although the two are independent, they are closely connected to each other by the universal rule in mathematics: the structure and performance of an operator T is greatly influenced by how the input functions are modeled or represented.

1.2.2 Image Modeling and Representation

To efficiently handle and process images, we need first to understand what images really are mathematically and how to represent them. For example, is it adequate to treat them as general L^2 functions or as a subset of L^2 with suitable regularity constraints? Here we briefly outline three major classes of image modeling and representation.

- *Random Fields Modeling*

An observed image u_0 is modeled as the sampling of a random field. For example, the Ising spin model in statistical mechanics can be used to model binary images. More generally, images are modeled by some Gibbs/Markovian random fields [R1]. The statistical properties of fields are often established through a filtering technique and learning theory. Random field modeling is the ideal approach for describing natural images with rich texture patterns such as trees and mountains.

• *Wavelet Representation*

An image is often acquired from the responses of a collection of microsensors (or photo receptors), either digital or biological. During the past two decades, it has been gradually realized (and experimentally supported) that such local responses can be well approximated by wavelets. This new representation tool has revolutionized our notion of images and their multiscale structures [R2]. The new JPEG2000 protocol for image coding and the successful compression of the FBI database of fingerprints are its two most influential applications. The theory is still being actively pushed forward by a new generation of geometric wavelets such as *curvelets* (Candés and Donoho) and *beamlets* (Pennec and Mallat).

• *Regularity Spaces*

In the linear filtering theory of conventional digital image processing, an image u is considered to be in the Sobolev space $W^{1,1}(\Omega)$. The Sobolev model works well for homogeneous regions, but it is insufficient as a global image model, since it "smears" the most important visual cue, namely, edges. Two well-known models have been introduced to recognize the existence of edges. One is the "object-edge" model of Mumford and Shah [16], and the other is the bounded-variation (BV) image model of Rudin, Osher, and Fatemi [8]. The object-edge model assumes that an ideal image u consists of disjoint homogeneous object patches $[u_k, \Omega_k]$ with $u_k \in W^{1,1}(\Omega_k)$ and regular boundaries $\partial\Omega_k$ (characterized by one-dimensional Hausdorff measure). The BV image model assumes that an ideal image has bounded total variation $\int_\Omega |Du|\,dx$. Regularity-based image models are generally applicable to images with low texture patterns and without rapidly oscillatory components.

1.2.3 Modeling of Image Processors

How images are modeled and represented very much determines the way we model image processors. We shall illustrate this viewpoint through the example of denoising $u = Tu_0$: $u_0 = u + n$, assuming for simplicity that the white noise n is additive and homogeneous, and there is no blurring involved. When images are represented by wavelets, the denoising processor T is in some sense "diagonalized" and is equivalent to a simple engineering on the individual wavelet components. This is a celebrated result of Donoho and Johnstone on threshold-based denoising schemes. Under the statistical/random field modeling of images, the denoising processor T becomes MAP (*Maximum A Posteriori*) estimation. By Bayes's formula, the posterior probability given an observation u_0 is

$$p(u|u_0) = p(u_0|u)p(u)/p(u_0)$$

The denoising processor T is achieved by solving the MAP problem

$$\max_u p(u|u_0)$$

Therefore, it is important to know not only the random field image model $p(u)$ but also the mechanism by which u_0 is generated from the ideal image u (the so-called *generative* data model). The two are crucial for successfully carrying out Bayesian denoising. Finally, if the ideal image u is modeled as an element in a regular function space such as $W^{1,1}(\Omega)$ or $\mathrm{BV}(\Omega)$, then the denoising processor T can be realized by a variational optimization. For instance, in the BV image model, T is achieved by

$$\min_u \int_\Omega |Du|\,dx$$

$$\text{subject to}: \frac{1}{|\Omega|}\int_\Omega (u-u_0)^2\,dx \le \sigma^2$$

where the white noise is assumed to be well approximated by the standard Gaussian $N(0,\sigma^2)$. This well-known denoising model, first proposed by Rudin, Osher, and Fatemi, belongs to the more general class of *regularized* data-fitting models. Just as different coordinate systems that describe a single physical object are related, different formulations of the same image processor are closely interconnected. Again, take denoising for example. It has been shown that the wavelet technique is equivalent to an approximate optimal regularization in certain Besov spaces (Cohen, Dahmen, Daubechies, and DeVore). On the other hand, Bayesian processing and the regularity-based variational approach can also be connected (at least formally) by Gibbs's formula in statistical mechanics, see [19].

1.3 Variational PDE Methods

Having briefly introduced the general picture of mathematical image processing, in the rest of the course we will focus on the variational PDE methods through two processors: restoration and segmentation. For the history and a detailed description of current developments of the variational and PDE method in image and vision analysis, see two special issues in *IEEE Trans. Image Processing* [**7** (3), 1998] and *J. Visual Comm. Image Rep.* [**13** (1/2), 2002] and also two recent monographs [1], [3]. In the variational or "energy"-based models, nonlinear PDEs emerge as one derives formal Euler-Lagrange equations or tries to locate local or global minima by the gradient descent method. Some PDEs can be studied by the viscosity solution approach [R3], while many others still remain open to further theoretical investigation. Compared with other approaches, the variational PDE method has remarkable advantages in both theory and computation. First, it allows one to directly handle and process visually important geometric features such as gradients, tangents, curvatures, and level sets. It can also effectively simulate several visually meaningful dynamic processes, such as linear and nonlinear diffusions and the information transport mechanism. Second, in terms of computation, it can profoundly benefit from the existing wealth of literature on numerical analysis and computational PDEs. For example, various well-designed shock-capturing schemes in computational fluid dynamics (CFD) can be conveniently adapted to edge computation in images.

Additional References

[R1] S. Geman and D. Geman, Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images, *IEEE Trans. Pattern Anal. Machine Intell.* **6** (1984), 721–741.
[R2] S. Mallat, *A Wavelet Tour of Signal Processing*, Academic Press, 1998.
[R3] M. G. Crandall, H. Ishii, and P. L. Lions, User's guide to viscosity solutions of second order partial linear differential equations, *Bull. Amer. Math. Soc. (N.S.)* **27** (1992), 1–67.

Chapter 2 BACKGROUND MATHEMATICS

2.1 Functionals and Euler-Lagrange Equations

2.1.1 *Functionals*

Roughly speaking, a functional is a "function of function". We are all familiar with the notion of function, say $f(x)$, which is taken to mean a correspondence that assigns a definite (real) number for each variable value x in a certain domain. Similarly, a *functional* is a correspondence that assigns a real number to each *function* (such as a curve, or a surface, etc.) that belongs to some class of functions.

For example, for every smooth curve that connects two fixed points $\boldsymbol{a}$ and $\boldsymbol{b}$, the length of the curve is a functional. As another example, the time required for a mass point to travel from an upper end-point down along a smooth and monotonically decreasing plane curve to the lower end-point is a functional. From these examples, we see that a functional is defined over a certain class of functions with certain degree of smoothness and satisfying certain boundary conditions.

2.1.2 *Variational Problems: Examples*

Example 2.1 A smooth plane curve that joins points $\boldsymbol{p}_a = (a, A)$ and $\boldsymbol{p}_b = (b, B)$ can be described by a differentiable function $y = y(x)$ with $A = y(a)$ and $B = y(b)$, and its length is known to be

$$J[y] = \int_a^b \sqrt{1 + y'^2}\, dx \tag{2.1}$$

Here $J[y]$ is obviously a functional for the class of smooth functions $y(x)$ that satisfy the boundary conditions $A = y(a)$ and $B = y(b)$. A simple variational problem associated with the functional in (2.1) is to find the shortest curve joining points $\boldsymbol{p}_a$ and $\boldsymbol{p}_b$. ■

Typically, for the one-variable case a functional of interest assumes the form

$$J[y] = \int_a^b f(x, y, y')\, dx \tag{2.2}$$

and for the two-variable case

$$J[u] = \iint_\Omega f(x, y, u, u_x, u_y)\, dxdy \tag{2.3}$$

where $u = u(x, y)$, $u_x = \partial u / \partial x$ and $u_y = \partial u / \partial y$.

Example 2.2 Let $u(x, y)$ denote a two-variable smooth function that models the light intensity of an (analog) image over region Ω. The total variation (TV) of the image is defined as

$$J[u] = \iint_\Omega \sqrt{u_x^2 + u_y^2}\, dx\, dy \tag{2.4}$$

which is obviously a functional of type (2.3) for the class of smooth functions defined over region Ω. It has been argued that the noise removal problem for images can be cast as minimizing the TV subject to some statistical constraints. ■

2.1.3 *Several Lemmas* [6]

The key analytical tool for investigating various image processing problems in a variational optimization setting is the Euler-Lagrange (EL) equation. The lemmas presented below are instrumental in deriving the EL equation.

In what follows the set of continuous functions over interval $[a, b]$ is denoted by $C(a, b)$ and the set of functions that are continuous and have up to p-th order continuous derivatives on $[a, b]$ is denoted by $D_p(a, b)$.

Lemma 2.1 If $\alpha(x)$ is continuous on $[a, b]$, and if

$$\int_a^b \alpha(x)h(x)dx = 0$$

for every function $h(x) \in C(a,b)$ such that $h(a) = h(b) = 0$, then $\alpha(x) = 0$ for all x in $[a, b]$.

Proof. The lemma will be proved by the method of contradiction. Suppose the function $\alpha(x)$ is nonzero, say positive, at some point in $[a, b]$. Then $\alpha(x)$ is also positive in some interval $[x_1, x_2]$ contained in $[a, b]$. If we set

$$h(x) = (x - x_1)(x_2 - x)$$

for x in $[x_1, x_2]$ and $h(x) = 0$ otherwise, then $h(x)$ obviously satisfies the condition of the lemma. However,

$$\int_a^b \alpha(x)h(x)\,dx = \int_{x_1}^{x_2} \alpha(x)(x - x_1)(x_2 - x)dx > 0$$

because the integrand is positive (except at x_1 and x_2). This contradiction proves the lemma. ■

Lemma 2.2 If $\alpha(x)$ is continuous in $[a, b]$, and if

$$\int_a^b \alpha(x)h'(x)dx = 0$$

for every $h(x) \in D_1(a,b)$ such that $h(a) = h(b) = 0$, then $\alpha(x) = c$ for all x in $[a, b]$, where c is a constant.

Proof. Let c be a constant given by

$$c = \frac{1}{b-a}\int_a^b \alpha(x)dx$$

which implies that

$$\int_a^b [\alpha(x) - c]dx = 0\,,$$

and define function $h(x)$ as

$$h(x) = \int_a^x [\alpha(\xi) - c]d\xi\,.$$

Evidently, $h(x)$ belongs to $D_1(a, b)$ and satisfies $h(a) = 0$ and $h(b) = 0$. Then on the one hand,

$$\int_a^b [\alpha(x) - c]h'(x)dx = \int_a^b \alpha(x)h'(x)dx - c[h(b) - h(a)] = 0 .$$

while on the other hand,

$$\int_a^b [\alpha(x) - c]h'(x)dx = \int_a^b [\alpha(x) - c]^2 dx .$$

Therefore, $\alpha(x) - c = 0$ i.e. $\alpha(x) = c$ for all x in $[a, b]$. ■

Lemma 2.3 If $\alpha(x)$ is continuous in $[a, b]$, and if

$$\int_a^b \alpha(x)h''(x)dx = 0$$

for every $h(x) \in D_2(a,b)$ such that $h(a) = h(b) = 0$ and $h'(a) = h'(b) = 0$, then $\alpha(x) = c_0 + c_1x$ for all x in $[a, b]$, where c_0 and c_1 are constants.

Proof. Let c_0 and c_1 be constants defined by the conditions

$$\int_a^b [\alpha(x) - c_0 - c_1x]dx = 0$$

$$\int_a^b dx \int_a^x [\alpha(\xi) - c_0 - c_1\xi]d\xi = 0$$

and let

$$h(x) = \int_a^x d\xi \int_a^\xi [\alpha(t) - c_0 - c_1t]dt$$

It is easy to verify that $h(x) \in D_2(a,b)$ and satisfies $h(a) = h(b) = 0$ and $h'(a) = h'(b) = 0$. Then on the one end,

$$\int_a^b [\alpha(x) - c_0 - c_1x]h''(x)dx$$

$$= \int_a^b \alpha(x)h''(x)dx - c_0[h'(b) - h'(a)] - c_1\int_a^b xh''(x)dx$$

$$= -c_1[bh'(b) - ah'(a)] - c_1[h(b) - h(a)] = 0$$

while on the other hand,

$$\int_a^b [\alpha(x) - c_0 - c_1x]h''(x)dx = \int_a^b [\alpha(x) - c_0 - c_1x]^2 dx = 0$$

It follows that $\alpha(x) - c_0 - c_1x = 0$, hence $\alpha(x) = c_0 + c_1x$ for all x in $[a, b]$. ■

Lemma 2.4 If $\alpha(x)$ and $\beta(x)$ are continuous in [a, b], and if

$$\int_a^b [\alpha(x)h(x) + \beta(x)h'(x)]dx = 0 \tag{2.5}$$

for every function $h(x) \in D_1(a,b)$ such that $h(a) = h(b) = 0$, then $\beta(x)$ is differentiable, and $\beta'(x) = \alpha(x)$ for all x in [a, b].

Proof. Let $A(x)$ be the function defined by

$$A(x) = \int_a^x \alpha(\xi)\, d\xi$$

Using integration by parts, we can verify that

$$\int_a^b \alpha(x)h(x)\, dx = -\int_a^b A(x)h'(x)\, dx$$

hence (2.5) can be written as

$$\int_a^b [-A(x) + \beta(x)]h'(x)\, dx = 0 \tag{2.6}$$

By Lemma 2.2, (2.6) implies that $\beta(x) - A(x) = \text{const},$ i.e., $\beta(x) = A(x) + \text{const},$ thus function $\beta(x)$ is differentiable and by definition of $A(x)$, $\beta'(x) = \alpha(x)$. ■

2.1.4 *Variation of a Functional*

Let $J[y]$ be a functional defined on some normed linear space and let

$$\Delta J[h] = J[y + h] - J[y]$$

be its increment corresponding to the increment $h = h(x)$ of the "independent variable" $y = y(x)$. If y is fixed, then $\Delta J[h]$ is a functional of h, in general a nonlinear functional. Suppose that

$$\Delta J[h] = \varphi[h] + \varepsilon \|h\|$$

where $\varphi[h]$ is a linear functional and $\varepsilon \to 0$ as $\|h\| \to 0$. Then the functional $J[y]$ is said to be *differentiable*, and the principal linear part of the increment $\Delta J[h]$, i.e., the linear functional $\varphi[h]$ which differs from $\Delta J[h]$ by an infinitesimal of order higher than 1 relative to $\|h\|$, is called the variation of $J[y]$ and is denoted by $\delta J[h]$. It can be shown that the variation of a differentiable functional is unique [6]. We now use the concept of the variation of a functional to establish a necessary condition for a functional to have an extremum.

Similar to the concept of the extremum of a function from analysis, we say that the functional $J[y]$ has a relative extremum for $y = \hat{y}$ if $J[y] - J[\hat{y}]$ does not change its sign in some neighborhood of the curve $y = \hat{y}(x)$.

At this point, it is important that the meaning of the term "neighborhood" is explicitly clarified. To this

end we need to define two norms, one for space $C(a, b)$ and the other for $D_1(a, b)$:

♦ For a function $y(x)$ in space $C(a, b)$, we define

$$\|y\|_{0,\infty} = \max_{a\le x\le b} |\, y(x)\,|$$

♦ For a function $y(x)$ in space $D_1(a, b)$, we define

$$\|y\|_{1,\infty} = \max_{a\le x\le b} |\, y(x)\,| + \max_{a\le x\le b} |\, y'(x)\,|$$

Depending on which of these two norms (topology) is employed, a *neighborhood* of the curve $y = \hat{y}(x)$ may be referred to those y such that $\|y - \hat{y}\|_{0,\infty} < \varepsilon$, or to those y such that $\|y - \hat{y}\|_{1,\infty} < \varepsilon$. Accordingly, we shall say that the functional $J[y]$ has a weak extremum for $y = \hat{y}$, if there exist an $\varepsilon > 0$, such that $J[y] - J[\hat{y}]$ has the same sign for all y in the neighborhood of $\hat{y}(x)$ characterized by $\|y - \hat{y}\|_{1,\infty} < \varepsilon$. On the other hand, we shall say that the functional $J[y]$ has a strong extremum for $y = \hat{y}$, if there exist an $\varepsilon > 0,$ such that $J[y] - J[\hat{y}]$ has the same sign for all y in the neighborhood of $\hat{y}(x)$ characterized by $\|y - \hat{y}\|_{0,\infty} < \varepsilon$.

Theorem 2.1 [6] A necessary condition for the differentiable functional $J[y]$ to have an extremum for $y = \hat{y}$ is that its variation vanish for $y = \hat{y}$, namely,

$$\delta J[h] = 0$$

for $y = \hat{y}$ and all admissible h.

Proof. To be explicit, suppose $J[y]$ has a minimum for $y = \hat{y}$. By the definition of the variation $\delta J[h]$, we have

$$\Delta J[h] = \varphi[h] + \varepsilon \|h\| \qquad (2.7)$$

where $\varepsilon \to 0$ as $\|h\| \to 0$. Hence, for sufficiently small $\|h\|$, the sign of $\Delta J[h]$ is the same as the sign of $\delta J[h]$. Now, suppose that $\delta J[h_0] \neq 0$ for some admissible h_0. Then for any $\alpha > 0$, no matter how small, we have

$$\delta J[-\alpha h_0] = -\delta J[\alpha h_0]$$

because $\delta J[h]$ is a linear functional of h. Hence (2.7) can be made to have either sign for arbitrary small $\|h\|$, which is not possible according to the hypothesis $J[y]$ has a minimum for $y = \hat{y}$. This contradiction proves the theorem. ■

2.1.5 *Euler-Lagrange Equation: The One-Variable Case*

We now study the simplest variational problem with one variable: *Let f(x, y, z) be a function with continuous first and second (partial) derivatives with respect to all its variables. Then, among all functions in $D_1(a, b)$ that satisfy the boundary conditions*

$$y(a) = A, \qquad y(b) = B \tag{2.8}$$

find the function for which the functional

$$J[y] = \int_a^b f(x, y, y')\,dx \tag{2.9}$$

has a weak extremum.

Suppose we give $y(x)$ an increment $h(x)$. In order for the function $y(x) + h(x)$ to continue to satisfy the boundary condition (2.8), we must impose

$$h(a) = h(b) = 0 \tag{2.10}$$

The corresponding increment of the functional (2.9) is given by

$$\Delta J = J[y+h] - J[y] = \int_a^b f(x, y+h, y'+h')dx - \int_a^b f(x, y, y')dx$$

$$= \int_a^b \left[f(x, y+h, y'+h') - f(x, y, y') \right] dx$$

Using Taylor's theorem, it follows that

$$\Delta J = \int_a^b [\, f_y(x, y, y')h + f_{y'}(x, y, y')h']\,dx + \cdots\cdots \tag{2.11}$$

where the subscripts denote partial derivatives with respect to the corresponding arguments, and the dots denote terms of order higher than 1 relative to h and h'. The integral in the right-hand side of (2.11) represents the principal linear part of the increment ΔJ, hence the variation of $J[y]$ is given by

$$\delta J = \int_a^b \left[f_y(x, y, y')h + f_{y'}(x, y, y')h' \right] dx$$

By Theorem 2.1, a necessary condition for $J[y]$ to have an extreme for $y = y(x)$ is that

$$\delta J = \int_a^b \left(f_y h + f_{y'} h' \right) dx = 0 \tag{2.12}$$

for all admissible h. According to Lemma 2.4, (12) implies that

$$f_y - \frac{d}{dx} f_{y'} = 0 \tag{2.13}$$

The equation in (2.13) is known as the *Euler-Lagrange equation*. Thus we have proved

Theorem 2.2 Let $J[y]$ be a functional of the form

$$J[y] = \int_a^b f(x, y, y')\,dx$$

defined on the set of functions which have continuous first derivatives in $[a, b]$ and satisfy the boundary conditions $y(a) = A$, $y(b) = B$. Then a necessary condition for $J[y]$ to have an extremum for a given $y(x)$ is that $y(x)$ satisfies the Euler-Lagrange equation

$$f_y - \frac{d}{dx} f_{y'} = 0$$

Example 2.3 Find a plane curve connecting two points $(a, y(a))$ and $(b, y(b))$ that possesses the shortest length. Evedently, the problem amounts to finding $y(x)$ that solves the variational minimization problem in Example 2.1, i.e.,

$$\begin{aligned} &\underset{y(x)}{\text{minimize}} \quad J[y] = \int_a^b \sqrt{1 + y(x)'^2}\, dx \\ &\text{subject to:} \quad y(a) = A,\ y(b) = B \end{aligned}$$

If we apply the EL equation (2.13) to the problem and note that function f in this case does not explicitly depend on y i.e. $f_y = 0$, we have

$$\frac{df_{y'}}{dx} = \frac{d}{dx}\left[\frac{y'}{\sqrt{1 + y'^2}}\right] = 0$$

which implies that

$$\frac{y'}{\sqrt{1 + y'^2}} = c$$

which in turn gives y' = const., hence $y = c_1 x + c_0$ for some constants c_0 and c_1 (to be determined by the two end-point constraints). As expected, we conclude that the unique straight segment connecting the two end points is the solution of the problem. ■

2.1.6 *Euler-Lagrange Equation: The Two-Variable Case*

In this section we consider functionals depending on functions of *two variables* that assume the form of

$$J[u] = \iint_\Omega f(x, y, u, u_x, u_y)\, dx\, dy \tag{2.14}$$

where Ω is some closed region and u_x, u_y are partial derivatives of $u = u(x, y)$.

Similar to Lemma 2.1 (for the one-variable case), we have

Lemma 2.5 If $\alpha(x, y)$ is a fixed function which is continuous in a closed region Ω, and if the integral

$$\iint_\Omega \alpha(x, y) h(x, y)\, dx\, dy$$

vanishes for every function $h(x, y)$ which has continuous first and second derivatives in Ω and equals zero on the boundary Γ of region Ω, then $\alpha(x, y) = 0$ everywhere in Ω.

The proof of the above lemma is similar in spirit to that of Lemma 2.1 and the interested reader is referred to [6].

In order to apply the necessary condition for an extremum of the functional (14), i.e., $\delta J = 0$, we must first calculate the variation δJ. Let $h(x, y)$ be an arbitrary function which has continuous first and second derivative and vanishes in the boundary Γ of Ω. Then if $u(x, y)$ belongs to the domain of definition, i.e., $u(x, y)$ and its first and second derivatives are continuous in R and $u(x, y)$ takes given values on boundary Γ, so does $u(x, y) + h(x, y)$. It follows that

$$\Delta J = J[u+h] - J[u]$$

$$= \iint_{\Omega} \left[f(x, y, u+h, u_x + h_x, u_y + h_y) - f(x, y, u, u_x, u_y) \right] dx\, dy$$

$$= \iint_{\Omega} \left[f_u h + f_{u_x} h_x + f_{u_y} h_y \right] dx\, dy + \cdots\cdots$$

The integral in the last equality represents the principal linear part of the increment ΔJ, hence the variation of $J[u]$ is given by

$$\delta J = \iint_{\Omega} \left[f_u h + f_{u_x} h_x + f_{u_y} h_y \right] dx\, dy$$

By using Green's theorem

$$\iint_{\Omega} \left(\frac{\partial Q}{\partial x} - \frac{\partial P}{\partial y} \right) dx\, dy = \int_{\Gamma} \left(P\, dx + Q\, dy \right)$$

we can write

$$\iint_{\Omega} \left(f_{u_x} h_x + f_{u_y} h_y \right) dx\, dy$$

$$= \iint_{\Omega} \left[\frac{\partial}{\partial x} \left(f_{u_x} h \right) + \frac{\partial}{\partial y} \left(f_{u_y} h \right) \right] dx\, dy - \iint_{\Omega} \left(\frac{\partial}{\partial x} f_{u_x} + \frac{\partial}{\partial y} f_{u_y} \right) h\, dx\, dy$$

$$= \int_{\Gamma} \left(f_{u_x} h\, dy - f_{u_y} h\, dx \right) - \iint_{\Omega} \left(\frac{\partial}{\partial x} f_{u_x} + \frac{\partial}{\partial y} f_{u_y} \right) h\, dx\, dy$$

$$= - \iint_{\Omega} \left(\frac{\partial}{\partial x} f_{u_x} + \frac{\partial}{\partial y} f_{u_y} \right) h\, dx\, dy$$

where the last equality is obtained by the fact that $h(x, y)$ assumes zero value on boundary Γ. Consequently, the variation of the functional $J[u]$ becomes

$$\delta J = \iint_{\Omega} \left(f_u - \frac{\partial}{\partial x} f_{u_x} - \frac{\partial}{\partial y} f_{u_y} \right) h(x, y)\, dx\, dy \qquad (2.15)$$

Thus the condition $\delta J = 0$ implies that the integral in (2.15) vanishes for any $h(x, y)$ satisfying the zero boundary condition. By Lemma 2.5, this leads to the second-order partial differential equation, again known as *the Euler-Lagrange equation*:

$$f_u - \frac{\partial}{\partial x} f_{u_x} - \frac{\partial}{\partial y} f_{u_y} = 0 \tag{2.16}$$

2.1.7 *Variational Derivatives*

Variational derivatives are for functionals and the concept is a direct and natural generalization of partial derivatives for multivariate functions. For illustration purposes consider a functional

$$J[y] = \int_a^b f(x, y, y')\, dx, \quad y(a) = A, \quad y(b) = B \tag{2.17}$$

We divide the interval into $n + 1$ equal subintervals by placing the grid points as

$$x_0 = a, x_1, \ldots, x_n, x_{n+1} = b, \quad \Delta x = x_{i+1} - x_i$$

and replace the smooth function $y(x)$ by the polygonal line with vertices

$$(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n), (x_{n+1}, y_{n+1}) \quad \text{with} \quad y_i = y(x_i)$$

We then approximate the functional in (2.17) by a *multivariate function* as

$$J(y_1, y_2, \ldots, y_n) = \sum_{i=0}^{n} f\left(x_i, y_i, \frac{y_{i+1} - y_i}{\Delta x}\right) \Delta x \tag{2.18}$$

If we compute the partial derivative $\partial J(y_1, \ldots, y_n) / \partial y_k$ and notice that y_k appears in (2.18) in only two terms, we obtain

$$\frac{\partial J}{\partial y_k \Delta x} = f_y\left(x_k, y_k, \frac{y_{k+1} - y_k}{\Delta x}\right) - \frac{1}{\Delta x}\left[f_{y'}\left(x_k, y_k, \frac{y_{k+1} - y_k}{\Delta x}\right) - f_{y'}\left(x_{k-1}, y_{k-1}, \frac{y_k - y_{k-1}}{\Delta x}\right)\right] \tag{2.19}$$

Note the term $\partial y_k \Delta x$ in the denominator on the left of (2.19) has a geometric meaning of being the area of the region between the curves of $y(x)$ and $y(x) + h(x)$ in the neighborhood of x_k. As the number of points of subdivision increases without limit, i.e., $\Delta x \to 0$, (2.19) converges to a limit that we call the variational derivative of the functional in (2.17) and we shall denote it as $\delta J / \delta y$. From (2.19) it follows that

$$\frac{\delta J}{\delta y} = f_y(x, y, y') - \frac{d}{dx} f_{y'}(x, y, y') \tag{2.20}$$

On comparing (2.20) with (2.13), we conclude that the meaning of the Euler-Lagrange equation is that *the variational derivative of the functional at an extremum $y(x)$ must vanish at every point x.*

In the general case, the variational derivative is defined as follows. Let $J[y]$ be a functional depending on

function $y(x)$ and suppose we give $y(x)$ an increment $h(x)$ which is nonzero only in the neighborhood of a point x_0. We denote $\Delta\sigma$ the area between the curve $h(x)$ and the x-axis and consider the ratio

$$\frac{J[y+h]-J[y]}{\Delta\sigma} \tag{2.21}$$

If we let increment $h(x)$ approach the zero function in such a way that the area $\Delta\sigma$ goes to zero while both the maximum of $|h(x)|$ and the length of the interval where $h(x)$ is nonvanishing go to zero, and the expression (2.21) has a limit, then the limit is called the variational derivative of $J[y]$ at point x_0 and is denoted as

$$\left.\frac{\delta J}{\delta y}\right|_{x=x0} \tag{2.22}$$

It can be shown that the analogs of all the familiar rules obeyed by the ordinary derivatives such as the formulas for differentiating sums and products of functions and composite functions etc. are valid for variational derivatives. From (2.21) and (2.22) we can write

$$\Delta J \equiv J[y+h]-J[y]=\left\{\left.\frac{\delta J}{\delta y}\right|_{x=x0}+\varepsilon\right\}\Delta\sigma \tag{2.23}$$

where $\varepsilon \to 0$ as $\Delta\sigma \to 0$. It follows that the differential or variation of the functional $J[y]$ at point x_0 can be expressed in terms of the variational derivative as

$$\delta J=\left.\frac{\delta J}{\delta y}\right|_{x=x0}\Delta\sigma \tag{2.24}$$

2.1.8 *Variational Problems with Constraints: One-Variable Case*

Many applications of the calculus of variations lead to problems in which constraints other than boundary conditions are imposed. Here we illustrate an elementary treatment of such problems by considering the minimization of the functional

$$J[y]=\int_a^b f(x,y,y')\,dx \tag{2.25}$$

subject to the end-point conditions

$$y(a)=A, \qquad y(b)=B \tag{2.26a}$$

and functional constraint

$$K[y]=\int_a^b g(x,y,y')\,dx=0 \tag{2.26b}$$

Theorem 2.3 [6] Suppose function $y = y(x)$ is an extremum for the variational problem (2.25), (2.26) and $y(x)$ is not an extremum of functional $K[y]$, then there exists a constant λ such that $y = y(x)$ is an extremum of the functional

$$\int_a^b (f+\lambda g)dx \tag{2.27}$$

i.e., $y = y(x)$ satisfies the differential equation

$$f_y - \frac{d}{dx} f_{y'} + \lambda\left(g_y - \frac{d}{dx} g_{y'} \right) = 0 \tag{2.28}$$

Proof. Let $J[y]$ have an extremum for the curve $y = y(x)$ subject to the constraints in (2.26). We choose two points x_1 and x_2 in $[a, b]$, where x_1 is arbitrary and x_2 satisfies a condition to be stated below. We then construct an increment $\delta_1 y(x) + \delta_2 y(x)$ for function $y(x)$, where $\delta_1 y(x)$ is nonzero only in a neighborhood of x_1, and $\delta_2 y(x)$ is nonzero only in a neighborhood of x_2. Using variational derivatives, we can express the corresponding increment ΔJ of the functional as

$$\Delta J = \left\{ \left. \frac{\delta f}{\delta y} \right|_{x=x1} + \varepsilon_1 \right\} \Delta\sigma_1 + \left\{ \left. \frac{\delta f}{\delta y} \right|_{x=x2} + \varepsilon_2 \right\} \Delta\sigma_2 \tag{2.29}$$

(see (2.23)), where the "small areas" $\Delta\sigma_1$ and $\Delta\sigma_2$ in this case are given by

$$\Delta\sigma_1 = \int_a^b \delta_1 y(x)\, dx\,, \quad \Delta\sigma_2 = \int_a^b \delta_2 y(x)\, dx$$

and $\varepsilon_1, \varepsilon_2 \to 0$ as $\Delta\sigma_1, \Delta\sigma_2 \to 0$. We now require that the varied curve

$$y^*(x) = y(x) + \delta_1 y(x) + \delta_2 y(x)$$

satisfy the condition $K[y^*] = K[y]$. If we write ΔK in a form similar to (2.29), then the requirement that $K[y^*] = K[y]$ becomes

$$\Delta K = K[y^*] - K[y] = \left\{ \left. \frac{\delta g}{\delta y} \right|_{x=x1} + \varepsilon_1' \right\} \Delta\sigma_1 + \left\{ \left. \frac{\delta g}{\delta y} \right|_{x=x2} + \varepsilon_2' \right\} \Delta\sigma_2 = 0 \tag{2.30}$$

By hypothesis that $y = y(x)$ is not an extremal of $K[y]$, we can choose a point x_2 at which

$$\left. \frac{\delta g}{\delta y} \right|_{x=x2} \neq 0$$

thus (2.30) leads to

$$\Delta\sigma_2 = -\left\{ \frac{\left. \frac{\delta g}{\delta y} \right|_{x=x1}}{\left. \frac{\delta g}{\delta y} \right|_{x=x2}} + \varepsilon' \right\} \Delta\sigma_1 \tag{2.31}$$

where $\varepsilon' \to 0$ as $\Delta\sigma_1 \to 0$. By setting

$$\lambda = -\frac{\left.\frac{\delta f}{\delta y}\right|_{x=x2}}{\left.\frac{\delta g}{\delta y}\right|_{x=x2}}$$

and substituting (2.31) into (2.29), we obtain

$$\Delta J = \left\{ \left.\frac{\delta f}{\delta y}\right|_{x=x1} + \lambda \left.\frac{\delta g}{\delta y}\right|_{x=x1} \right\} \Delta\sigma_1 + \varepsilon\, \Delta\sigma_2 \tag{2.32}$$

where $\varepsilon \to 0$ as $\Delta\sigma_1 \to 0$. The above expression for ΔJ explicitly involves variational derivatives only at $x = x_1$ and the increment $h(x)$ is now just $\delta_1 y(x)$, because the "compensating increment" $\delta_2 y(x)$ has been taken into account by using the condition $\Delta K = 0$. Therefore, the first term on the right-hand side of (2.32) represents the principal part of ΔJ, i.e. the variation of the functional J at point x_1 is given by

$$\delta J = \left\{ \left.\frac{\delta f}{\delta y}\right|_{x=x1} + \lambda \left.\frac{\delta g}{\delta y}\right|_{x=x1} \right\} \Delta\sigma_1$$

Since the necessary condition for $y(x)$ to be an extreme of $J[y]$ is $\delta J = 0$ and since $\Delta\sigma_1$ is nonzero while x_1 is arbitrary, we have $\frac{\delta f}{\delta y} + \lambda \frac{\delta g}{\delta y} = 0$ which by (2.20) leads to (2.28), and the proof is complete. ■

Remark: If we define the integrand of the functional in (2.27) as the Lagrangian of the problem in (2.25), (2.26) and denote it as

$$L(x, y, y', \lambda) = f(x, y, y') + \lambda g(x, y, y') \tag{2.33}$$

then the equation in (2.28) can be expressed as the Euler-Lagrange equation of the Lagrangian:

$$L_y(x, y, y', \lambda) - \frac{d}{dx} L_{y'}(x, y, y', \lambda) = 0 \tag{2.34}$$

We now present a theorem, which is the counterpart of Theorem 2.3 for two-variable functions, without proof.

Theorem 2.4 Suppose function $u = u(x, y)$ is an extremum for the variational problem

$$\underset{u(x,y)}{\text{minimize}} \quad J[u] = \iint_\Omega f(x, y, u, u_x, u_y)\, dx\, dy \tag{2.35a}$$

$$\text{subject to:} \quad \iint_\Omega g_1(x, y, u, u_x, u_y)\, dx\, dy = 0 \tag{2.35b}$$

$$\iint_\Omega g_2(x, y, u, u_x, u_y)\, dx\, dy = 0 \tag{2.35c}$$

then there exist constants λ_1 and λ_2 such that $u = u(x, y)$ is an extremum of the functional

$$\iint_\Omega (f + \lambda_1 g_1 + \lambda_2 g_2)\, dx\, dy \tag{2.36}$$

i.e., $u = u(x, y)$ satisfies the Euler-Lagrange equation

$$L_u - \frac{\partial}{\partial x} L_{u_x} - \frac{\partial}{\partial y} L_{u_y} = 0 \tag{2.37}$$

where $L = f + \lambda_1 g_1 + \lambda_2 g_2$ is the Lagrangian of problem (2.35).

2.2 Function Spaces L^p and $W^{1,p}$

2.2.1 *Function Space* $L^p(\Omega)$

Let Ω be an open set of R^N. For each finite $p \geq 1$, space $L^p(\Omega)$ is defined as the set of all Lebesque measurable functions $f(x)$ such that $\int_\Omega |f(x)|^p\, dx < \infty$. It is well known that $L^p(\Omega)$ is a Banach space, i.e. a complete and normed function space whose norm is defined by

$$\|f(x)\|_p = \left[\int_\Omega |f(x)|^p\, dx\right]^{1/p} \tag{2.38}$$

For $p = \infty$, $L^\infty(\Omega)$ is the space of Lebesque measurable functions $f(x)$, each is bounded by a constant in the entire region Ω except possibly in a set of zero measure.

2.2.2 *Sobolev Space* $W^{1,p}$

With $1 \leq p \leq \infty$, the Sobolev space $W^{1,p}(\Omega)$ is characterized as the set of functions $f(x) \in L^p(\Omega)$ whose first-order partial derivatives also belong to $L^p(\Omega)$.

Notation $W_0^{1,p}(\Omega)$ is referred to the set $W_0^{1,p}(\Omega) = \{u(x) : u \in W^{1,p}(\Omega), u|_{\partial\Omega} = 0\}$.

Sobolev spaces play an important role in the theory of partial differential equations.

2.3 Functions of Bounded Variation

2.3.1 *Bounded Variation Functions of One Variable*

Definition Let $u(x)$ be a function defined over $[a, b]$ on which it always assumes finite values. For a set grid points $a = x_0 < x_1 < \ldots < x_n = b$, we construct the sum

$$V_u(x_0, \cdots, x_n) = \sum_{i=1}^{n} |u(x_i) - u(x_{i-1})|$$

which is called *the variation of u(x) with respect to grid set* (x_0, …, x_n). If there a constant M such that

$$\sup_{x_0,\cdots,x_n} V_u(x_0,\cdots,x_n) \le M < \infty$$

regardless of the number of grid points, then $u(x)$ is called a function of *bounded variation* (BV), and

$$\bigvee_a^b(u) = \sup_{x_0,\cdots,x_n} V_u(x_0,\cdots,x_n)$$

is called the total variation (TV) of $u(x)$ on $[a, b]$. In what follows, "$u(x)$ is BV" is taken to mean that $u(x)$ is a function of BV.

Example 2.4 Let $u(x)$ be a function defined on $[a, b]$ that satisfies the Lipschitz condition, namely, there exists a constant M such that, for any x and x' on $[a, b]$, $|u(x)-u(x')| \le M\,|x-x'|$ holds. It follows that

$$V_u(x_0,\cdots,x_n) = \sum_{i=1}^{n} |u(x_i)-u(x_{i-1})| \le \sum_{i=1}^{n} M\,|x_i - x_{i-1}| = M(b-a)$$

which implies that $\bigvee_a^b(u) \le M(b-a)$, hence $u(x)$ is BV. In addition, if we use Lagrange's formula $u(x_1)-u(x_2) = u'(\xi)\cdot(x_1-x_2)$ for some $\xi \in [x_1, x_2]$, then we conclude that if $u(x)$ has bounded first-order derivative on $[a, b]$, then $u(x)$ is a BV function. ■

Example 2.5 A continuous function is not necessarily BV function. For example, function $u(x)$ defined by

$$u(x) = \begin{cases} x\sin\dfrac{1}{x} & \text{for } 0 < x \le 1 \\ 0 & \text{for } x = 0 \end{cases}$$

is continuous on [0, 1] (see the figure below), but it can be shown that $u(x)$ in not a BV function (Problem 2.?). ■

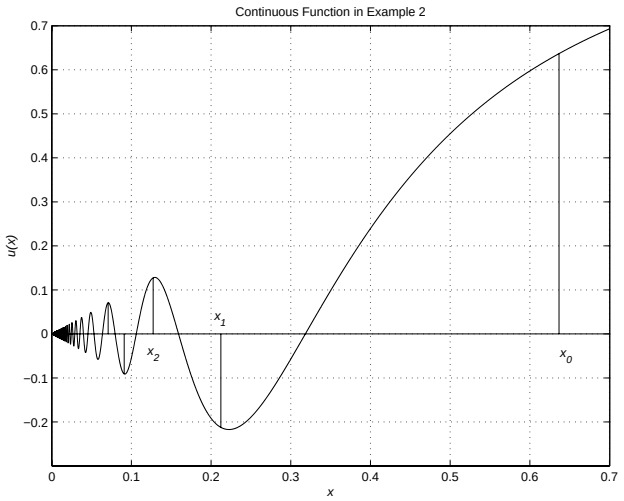


Figure: Function $u(x)$ for Example 2.5

2.3.2 *Properties of Bounded Variation Functions*

Some properties and characterization of BV functions are listed below without proof.

(P1) If $u(x)$ is monotonic on $[a, b]$, then $u(x)$ is BV and $\overset{b}{\underset{a}{\vee}}(u) = |u(a) - u(b)|$.

(P2) If $u(x)$ is BV, then it is a bounded function, i.e., there exists a constant M such that $|u(x)| \le M$ for $x \in [a,b]$.

(P3) If $u(x)$ is BV and $\overset{b}{\underset{a}{\vee}}(u) = 0$, then $u(x)$ on $[a, b]$ is a constant.

(P4) If $u(x)$ and $v(x)$ are BV on $[a, b]$ and α and β are constants, then $\alpha u(x) + \beta v(x)$ is BV.

(P5) If $u(x)$ and $v(x)$ are BV on $[a, b]$, then $u(x) \cdot v(x)$ is BV.

(P6) If $u(x)$ is BV on $[a, b]$ and c is any real value with $a < c < b$, then

$$\overset{b}{\underset{a}{\vee}}(u) = \overset{c}{\underset{a}{\vee}}(u) + \overset{b}{\underset{c}{\vee}}(u)$$

Conversely, if $u(x)$ is BV on $[a, c]$ and $[c, b]$, then $u(x)$ is also BV on $[a, b]$.

(P7) If $u(x)$ is BV on $[a, b]$, then $\overset{x}{\underset{a}{\vee}}(u)$ is a monotonically increasing function of x on $[a, b]$.

(P8) **Jordan's Decomposition Theorem** If $u(x)$ is defined on $[a, b]$ and is BV, then $u(x)$ can be expressed is the difference of two monotonically increasing functions, e.g.,

$$u(x) = \varphi(x) - \psi(x) \quad \text{with} \quad \varphi(x) = \frac{1}{2}\left[\overset{x}{\underset{a}{\vee}}(u) + u(x)\right], \psi(x) = \frac{1}{2}\left[\overset{x}{\underset{a}{\vee}}(u) - u(x)\right]$$

(P9) If $u(x)$ is BV on $[a, b]$, then it has at most a countable infinite number of points on $[a, b]$, where $u(x)$ is discontinuous and all discontinuous points are of first kind; $u(x)$ is Riemann-integralable; $u(x)$ has finite derivative almost everywhere on $[a, b]$; and its derivative $u'(x)$ is Lesbeque-integralable on $[a, b]$.

(P10) If $u(x)$ is BV on $[a, b]$, then

$$\frac{d}{dx}\overset{x}{\underset{a}{\vee}}(u) = |u'(x)| \tag{2.39}$$

holds almost everywhere on $[a, b]$.

2.3.3 *Bounded Variation Functions of Two Variables*

From (2.38), we can write $d\overset{x}{\underset{a}{\vee}}(u) = |u'(x)|dx$, hence the total variation (TV) of $u(x)$ can be computed as

$$\overset{b}{\underset{a}{\vee}}(u) = \int_a^b d\overset{x}{\underset{a}{\vee}}(u) = \int_a^b |u'(x)|dx \tag{2.40}$$

The expression of TV in (2.40) provides a natural way to extend the concept of BV from the one-variable case to the two-variable case.

Definition A function $u(x, y)$ defined in a compact region Ω in R^2 is said to be a function of bounded variation (BV) if it has a finite total variation (TV) which is defined by

$$\vee_{\Omega}(u) = \iint_{\Omega} \sqrt{u_x^2 + u_y^2}\, dx\, dy \tag{2.41}$$

where $u_x = \partial u(x, y) / \partial x, u_y = \partial u(x, y) / \partial y$. Note that the integrand in (2.41), i.e. $\sqrt{u_x^2 + u_y^2}$ is just the magnitude of the gradient of $u(x, y)$, so the TV can also be written as

$$\vee_{\Omega}(u) = \iint_{\Omega} \left|\nabla u(x, y)\right| dx\, dy \tag{2.42}$$

which is the L^1 norm of vector function ∇u . We shall present a more rigorous setting for BV functions in the next section after the introduction of Banach space and several other concepts.

2.4 Topologies on Banach Spaces

2.4.1 *Vector Spaces, Normed Linear Spaces, and Banach Spaces*

Definition A *vector space* over the real field is a set V whose elements are called vectors and in which two operations called addition and scalar multiplication are defined with the following familiar algebraic properties:

(P1) $x + y = y + x$
(P2) $x + (y + z) = (x + y) + z$
(P3) V contains a unique vector 0 (called the zero vector) such that $x + 0 = x$
(P4) to each x in V there exists a unique vector $-x$ such that $x + (-x) = 0$
(P5) For real scalars α and β, $1x = x$, $\alpha(\beta x) = (\alpha\beta)x$, and
$\alpha(x + y) = \alpha x + \beta x, \quad (\alpha + \beta)x = \alpha x + \beta x$

Definition A *linear transformation* (*operator*) of a vector space V into a vector space V_1 is a mapping A of V into V_1 such that

$$A(\alpha x + \beta y) = \alpha Ax + \beta Ay$$

for all x, y in V and all scalars α and β. In the special case in which V_1 is the field of real scalars, A is called a *linear functional*.

Definition A vector space X is said to be a normed linear space if to each x in X, there is associated a nonnegative real number $\|x\|$, called the norm of x, such that

(a) $\|x + y\| \le \|x\| + \|y\|$ for all x and $y \in X$
(b) $\|\alpha x\| = |\alpha| \|x\|$
(c) $\|x\| = 0$ implies $x = 0$

By (a), the triangle inequality

$$\|x - y\| \le \|x - z\| + \|z - y\|$$

holds for any x, y, z in X.

Definition A *Banach space* is a normed linear space which is complete in the metric defined by its norm. Recall that a space is said to be complete is every Cauchy sequence is a convergent sequence. (A sequence $\{x_n\}$ is said to be a Cauchy sequence if for any given $\varepsilon > 0$, there exists an integer $N > 0$ such that $\|x_n - x_m\| < \varepsilon$ as long as n and m are greater than N.)

Example 2.6 For $1 \le p < \infty$, each $L^p(\Omega)$ is a Banach space normed by $\|f\|_p$ in (2.38). For $p = \infty$, $L^\infty(\Omega)$ is the set of functions that are essentially bounded over Ω .If we define the norm of a function f in $L^\infty(\Omega)$ as its essential supremum, then $L^\infty(\Omega)$ is also a Banach space. ■

Example 2.7 For $1 \le p < \infty$, each Sobolev space $W^{1,p}(\Omega)$ is a Banach space normed by

$$\|f\|_{1,p} = \|f\|_p + \sum_{i=1}^{n} \left\| \frac{\partial f}{\partial x_i} \right\|_p \tag{2.43}$$

2.4.2 *Topologies on Banach Spaces*

Definition Let A be a linear operator from a normed linear space X into a normed linear space Y, the norm of operator A is defined by

$$\|A\| = \sup_{x \ne 0, x \in X} \left\{ \frac{\|Ax\|}{\|x\|} \right\} \tag{2.44}$$

The set of all linear functionals in Banach X is called the dual of X and is denoted by X', namely

$$X' = \left\{ l : X \to R \text{ linear such that } \|l\|_{X'} = \sup_{x \ne 0} \frac{|l(x)|}{\|x\|_X} < \infty \right\} \tag{2.45}$$

Definition Two topologies can be endowed in a Banach space:

(i) The strong topology, denoted by $x_n \xrightarrow[X]{} x$, is defined by $\|x_n - x\|_X \to 0$ as $n \to \infty$.

(ii) The weak topology, denoted by $x_n \xrightarrow[X]{} x$, is defined by $l(x_n) \to l(x)$ for every $l \in X'$.

Obviously, strong convergence implies weak convergence, but the converse is false in general.

Definition The dual space X' can also be endowed with strong and weak topologies:

(i) The strong topology, denoted by $l_n \xrightarrow[X']{} l$, is defined by

$$\|l_n - l\|_{X'} \to 0, \quad \text{i.e.} \quad \sup_{x \ne 0} \frac{|l_n(x) - l(x)|}{\|x\|_X} \to 0 \text{ as } n \to \infty$$

(ii) The weak topology, denoted by $l_n \xrightarrow[X']{} l$, is defined by

$$z(l_n) \to z(l) \text{ for every } z \in (X')' \text{ the bidual space of } X$$

It is often more convenient and useful to equip space X' with a third topology:

(iii) The weak* topology, denoted by $l_n \xrightarrow[X']{*} l$, is defined by

$$l_n(x) \to l(x) \text{ for every } x \in X$$

Definition Space X is said to be reflexive if $(X')' = X$. Space X is called separable if is contains a countable dense subset.

Example 2.8 $L^p(\Omega)$ is reflexive for $1 < p < \infty$, and separable for $1 \le p < \infty$. The dual space of $L^p(\Omega)$ for $1 \le p < \infty$ is $L^{p'}(\Omega)$ with $1/p + 1/p' = 1$. ■

Example 2.9 $X = L^1(\Omega)$ is nonreflexive and its dual space is $X' = L^\infty(\Omega)$. ■

The main properties associated with these topologies are summarized in the following theorem.
Theorem 2.5 (Weak sequential compactness)

(i) Let X be a reflexive Banach space and $\{x_n\}$ be a bounded sequence in X, i.e., $\|x_n\| \le K$ for some constant $K > 0$. Then there exist an x in X and a subsequence $\{x_{nk}\}$ of $\{x_n\}$ such that $x_{n_k} \xrightarrow[X]{} x$ as $k \to \infty$.

(ii) Let X be a separable Banach space and $l_n \in X'$ such that $\|l_n\|_{X'} \le K$. Then there exist $l \in X'$ and a subsequence l_{n_k} of l_n such that $l_{n_k} \xrightarrow[X']{*} l$ as $k \to \infty$.

2.4.3 *Convexity and Lower Semicontinuity*

Here we begin with the concepts of lower and upper limits of sequences of real numbers in calculus, and lower semicontinuity of functionals for weak topology.

Definition Let $\{s_n\}$ be a sequence of real numbers, and E be the set of numbers s such that $s_{n_k} \to s$ for some subsequence $\{s_{n_k}\}$. In other words, set E contains all subsequence limits , plus possibly the numbers $+\infty, -\infty$. The *upper* and *lower limits* of $\{s_n\}$ are defined by $s^* = \sup E$ and $s_* = \inf E$ and are denoted by $\overline{\lim}\, s_n = s^*$ and $\underline{\lim}\, s_n = s_*$, respectively.

Example 2.10 Let $s_n = (-1)^n/[1 + (1/n)]$. It can readily verified that

$$\overline{\lim}\, s_n = 1 \text{ and } \underline{\lim}\, s_n = -1 \qquad ■$$

Definition Functional F is called *lower semicontinuous* (l.s.c.) at x_0 for the weak topology if for any sequence $x_n \rightharpoonup x_0$, we have

$$\underline{\lim}_{x_n \rightharpoonup x_0} F(x_n) \ge F(x_0) \tag{2.46}$$

Similarly, F is called *upper semicontinuous* (u.s.c.) at x_0 for the weak topology if for any sequence $x_n \rightleftharpoons x_0$, we have

$$\overline{\lim_{x_n \rightleftharpoons x_0}} F(x_n) \le F(x_0)$$

Example 2.11 Consider functions by modifying $\cos(x)$ at point $x_0 = \pi/2$ as

$$f_1(x) = \begin{cases} \cos(x) & \text{for } x \neq \pi/2 \\ -0.5 & \text{for } x = \pi/2 \end{cases} \quad \text{and} \quad f_2(x) = \begin{cases} \cos(x) & \text{for } x \neq \pi/2 \\ 0.5 & \text{for } x = \pi/2 \end{cases}$$

Because

$$\underline{\lim_{x_n \rightleftharpoons x_0}} f_1(x_n) = 0 \ge -0.5 = f_1(x_0) \quad \text{and} \quad \overline{\lim_{x_n \rightleftharpoons x_0}} f_2(x_n) = 0 \le 0.5 = f_2(x_0)$$

$f_1(x)$ is lower semicontinuous at x_0 and $f_2(x)$ is upper semicontinuous at x_0. ■

Definition A functional F is called convex in X if for all x, y in X and $\lambda \in [0, 1]$

$$F\left(\lambda x + (1-\lambda) y\right) \le \lambda F(x) + (1-\lambda) F(y)$$

There are several results that relate the concept of l.s.c to convexity.

Theorem 2.6 If F is a convex functional, then F is weakly l.s.c. if and only if it is strongly l.s.c.

The theorem is useful because strong l.s.c. is usually easier to prove.

For integral functionals of the type

$$F(u) = \int_\Omega f(x, u(x), \nabla u(x))\, dx$$

where $0 \le f(x, u, \xi) \le a(x, |u|, |\xi|)$ and a is increasing with respect to $|u|$ and $|\xi|$, we have

Theorem 2.7 $F(u)$ is sequentially weakly l.s.c. in $W^{1,p}(\Omega)$ (weakly * l.s.c. if $p = \infty$) if and only if f is convex with respect to ξ.

Now let us consider a functional F in Banach space X and the the minimization problem

$$\inf_{x \in X} F(x) \tag{2.47}$$

Concerning the existence of a solution for problem (2. 47), a typical treatment of the problem involves three steps as follows.

(A) Construct a minimizing sequence $x_n \in X$, i.e., a sequence satisfying

$$\lim_{n\to\infty} F(x_n) = \inf_{x\in X} F(x)$$

(B) If F is coercive, i.e. $\lim_{|x|\to+\infty} F(x) = +\infty$, then the sequence x_n is uniformly bounded, i.e., $\|x_n\|_X \le C$. Now if X is reflexive, then by Theorem 2.5 there exist an x_0 in X and a subsequence $\{x_{nk}\}$ of $\{x_n\}$ such that $x_{n_k} \xrightarrow[X]{} x_0$.

(C) To prove that x_0 is a minimum point of F, one only needs to show that F is l.s.c. at x_0, i.e., $\underline{\lim}_{x_{n_k}\rightharpoonup x_0} F(x_{n_k}) \ge F(x_0)$ as it implies that $F(x_0) = \min_{x\in X} F(x)$.

2.4.4 *The Space of Bounded Variation Functions*

In Sec. 2.3, elementary studies of bounded variation functions were carried out for both the one-variable and two-variable cases. In what follows study the function space and its basic properties in a more rigorous manner.

Definition Let Ω be an open subset of R^N and let $u(x)$ be a function in $L^1(\Omega)$. We define the total variation (TV) of function $u(x)$ as

$$\int_\Omega |Du| = \sup\left\{\int_\Omega u \operatorname{div}\varphi\, dx;\ \varphi = (\varphi_1, \varphi_2, \cdots \varphi_N) \in C_0^1(\Omega)^N, \|\varphi\|_{L^\infty(\Omega)} \le 1\right\} \tag{2.48}$$

where $\operatorname{div}\varphi = \sum_{i=1}^N \frac{\partial \varphi_i}{\partial x_i}$ is the divergence of the vector-valued function $\varphi(x) = \left(\varphi_1(x), \varphi_2(x)\cdots\varphi_N(x)\right)$, dx is the Lebesque measure, $C_0^1(\Omega)^N$ is the space of continuously differentiable functions with compact support in Ω (i.e., $\varphi|_{\partial\Omega} = 0$), and $\|\varphi\|_{L^\infty(\Omega)} = \sup_{x\in\Omega} \|\varphi(x)\|_2 = \sup_{x\in\Omega} \sqrt{\sum_{i=1}^N \varphi_i^2(x)}$.

Example 2.12 If $u \in C^1(\Omega)$ with $N = 2$, then we can write

$$\int_\Omega u \operatorname{div}\varphi\, dx = -\int_\Omega \nabla u \cdot \varphi\, dx$$

hence the supremum is achieved when $\varphi = -\nabla u / \|\nabla u\|_2 = \dfrac{-\nabla u}{\sqrt{\left(\partial u/\partial x_1\right)^2 + \left(\partial u/\partial x_2\right)^2}}$ which leads the expression in (2.48) to

$$\int_\Omega |Du| = \sup\left\{\int_\Omega u \operatorname{div}\varphi\, dx;\ \varphi = (\varphi_1, \varphi_2) \in C_0^1(\Omega)^2, \|\varphi\|_{L^\infty(\Omega)} \le 1\right\} \tag{2.48'}$$

$$= \int_\Omega \nabla u \cdot \frac{\nabla u}{\|\nabla u\|_2} dx = \int_\Omega \|\nabla u\|_2\, dx = \int_\Omega \sqrt{\left(\frac{\partial u}{\partial x_1}\right)^2 + \left(\frac{\partial u}{\partial x_2}\right)^2}\, dx \equiv \int_\Omega |\nabla u|\, dx$$

We see that the definition of TV is identical to that in (2.41) (and (2.42)) as long as $u(x)$ is a smooth function. However, the present definition of TV allows the concept be applied to a much broader function class than the Sobolev space $W^{1,1}$. ■

Example 2.13 Consider the one-variable function $u(x)$ defined by

$$u(x) = \begin{cases} -1 \text{ if } -1 \le x < 0 \\ 1 \text{ if } 0 < x \le 1 \end{cases}$$

Note that $u(x)$ does not belong to $W^{1,1}$ because at $x = 0$ the derivative of $u(x)$ does not exist. However we can compute for $\varphi \in C_0^1(-1, 1)$

$$\int_{-1}^{1} u(x) \cdot \varphi'(x)\, dx = \int_{-1}^{0} -\varphi'(x)\, dx + \int_{0}^{1} \varphi'(x)\, dx = -2\varphi(0)$$

hence the supremum in (2.48) gives $\int_{-1}^{1} |Du| = 2$, which makes sense as the value coincides with the total variation of the function on [–1, 1] that in this case equals to the jump magnitude of $u(x)$ at point $x = 0$. ■

Definition The space of *bounded variation functions* is a set of functions $u(x)$ with bounded total variation $\int_\Omega |Du|$, and the space is denoted by $BV(\Omega)$.

Example 2.14 Let A be a subset of R^N and $u(x) = \chi_A(x)$ be the characteristic function of A, i.e., $\chi_A(x) = 1$ if $x \in A$ and $\chi_A(x) = 0$ elsewhere. The expression in (2.48) then becomes

$$\int_\Omega |Du| = \sup \left\{ \int_A \operatorname{div} \varphi \, dx; \varphi \in C_0^1(\Omega)^N, |\varphi|_{L^\infty(\Omega)} \le 1 \right\} \tag{2.49}$$

where $|\varphi| = \sqrt{\sum_{i=1}^{N} \varphi_i^2(x)}$. If the supremum in (2.49) is finite, A is said to be a set with finite perimeter in Ω and we write

$$\int_\Omega |Du| = \mathrm{Per}_\Omega(A) \tag{2.50}$$

If ∂A (the boundary of A) is smooth, then $\mathrm{Per}_\Omega(A)$ is the classical perimeter for $N = 2$ and surface area for $N = 3$. ■

Several useful properties of $BV(\Omega)$ are summarized below, where we assume Ω is bounded and has a Lipschitz boundary.

(P1) (*Lower semicontinuity*) TV is lower semicontinuous with respect to L^1 topology. Namely, if $u_j \in BV(\Omega)$ and $u_j \xrightarrow[j\to\infty]{L^1(\Omega)} u$, then $\int_\Omega |Du| \le \varliminf_{j\to\infty} \int_\Omega |Du_j|$.

(P2) (*A weak* topology*) When endowed with the norm $\|u\|_{BV(\Omega)} = \|u\|_{L^1(\Omega)} + \int_\Omega |Du|$, $BV(\Omega)$ is a Banach space. However, the space does not possess good compactness properties this way. Typically, one works in $BV(\Omega)$ with the BV-w* topology which is defined as

$$u_j \xrightarrow[BV-w^*]{*} u \quad \Leftrightarrow \quad u_j \xrightarrow[L^1(\Omega)]{} u \text{ and } Du_j \xrightarrow[M]{*} Du$$

where $Du_j \xrightarrow[M]{*} Du$ means $\int_\Omega \varphi Du_j \to \int_\Omega \varphi Du$ for all φ in $C_0(\Omega)^N$. Equipped with this topology, $BV(\Omega)$ has the following compactness property.

(P3) (*Compactness*) Every uniformly bounded sequence u_j in $BV(\Omega)$ is relatively compact in $L^p(\Omega)$ for $1 \le p < N/(N-1)$. Moreover, there exist a subsequence u_{jk} and u in $BV(\Omega)$ such that $u_{j_k} \xrightarrow[BV-w^*]{*} u$.

See [1, Sec.2.2.3] for more details.

2.5 Elements of Differential Geometry

Roughly speaking, differential geometry is a branch of mathematics that studies local and global properties of various geometrical objects like smooth curves, surfaces, and manifolds by means of differential calculus and equations. A concept in differential geometry that is of central importance in the present course of studies is *curvature*. The notion of curvature can be defined for parameterized curves (such as snakes), for images that are represented by their isovalues, and for images viewed as surfaces where the height is the gray-scale intensity.

2.5.1 *Parameterized Curves*

Let $x(p) = (x_1(p), x_2(p))$ with $0 \le p \le 1$ be a regular planar curve in R^2 (a curve is said to be regular if, for each p in [0, 1], $x_1'(p)$ and $x_2'(p)$ do not become zero simultaneously). We note that

- $T(p) = x'(p) = (x_1'(p), x_2'(p))$ is a tangent vector at $x(p)$,
- $N(p) = (-x_2'(p), x_1'(p))$ is a normal vector at $x(p)$,
- The arc length of the portion of curve between the starting point $x(0) = (x_1(0), x_2(0))$ and point $x(p) = (x_1(p), x_2(p))$ is denoted by $s(p)$ and is evaluated as $s(p) = \int_0^p \sqrt{(x_1'(r))^2 + (x_2'(r))^2}\, dr = \int_0^p |x'(r)|\, dr$.

It follows that

$$\frac{ds}{dp} = |x'(p)| = \left|\frac{dx(p)}{dp}\right|$$

thus if curve x is parameterized by arc length s, then we have

$$|T(s)| = |x'(s)| = \left|\frac{dx(s)}{ds}\right| = \frac{ds}{ds} = 1$$

In other words, when parameterized by arc length s, the tangent vector of a regular planar curve is a unit vector.

- The curvature tensor of curve $x(s)$ is defined by

$$\frac{dT(s)}{ds} = \frac{d^2x(s)}{ds^2}$$

Since vector $T(s)$ has a fixed length ($T(s)$ is a unit vector), the derivative of $|T(s)|^2$ vanishes. This gives $T'(s)\cdot T(s)=0$, namely the curvature tensor $T'(s)$ is orthogonal to the tangent vector $T(s)$. Since the normal vector is also orthogonal to the tangent vector and the normal vector parameterized by s is also a unit vector, we conclude that

$$\frac{dT(s)}{ds}=\kappa(s)N(s)$$

where $\kappa(s)=|T'(s)|=|x''(s)|$ is called the *curvature* of curve $x(s)$ and $1/\kappa(s)$ is called the *radius of the curvature*.

For an arbitrary parameterization $x(p) = (x_1(p), x_2(p))$, we have

$$\kappa(p)=\frac{x_1'(p)\,x_2''(p)-x_2'(p)\,x_1''(p)}{\left(x_1'(p)^2+x_2'(p)^2\right)^{3/2}} \tag{2.51}$$

and

$$\frac{1}{|x'(p)|}\frac{d}{dp}\left(\frac{x'(p)}{|x'(p)|}\right)=\kappa(p)\frac{N(p)}{|N(p)|}$$

2.5.2 *Curves as Isolevel of a Function*

Let us now consider the case where $x(s)$ is modeled as an isolevel of a function $u(x_1, x_2)$ as

$$x(s)=\{(x_1(s),\,x_2(s)):u(x_1(s),\,x_2(s))=k\}$$

for some constant k. By differentiating the equation $u(x_1(s), x_2(s)) = k$ with respect to s, we obtain

$$x_1'(s)\,u_{x_1}+x_2'(s)\,u_{x_2}=0 \tag{2.52}$$

hence $(x_1'(s),\,x_2'(s))$ and $(-u_{x_2},\,u_{x_1})$ are collinear and there exists a scalar λ such that

$$\begin{cases} x_1'(s)=-\lambda u_{x_2} \\ x_2'(s)=\lambda u_{x_1} \end{cases} \tag{2.53}$$

so vectors (u_{x1}, u_{x2}) and $(-u_{x2}, u_{x1})$ are normal and tangent to curve $x(s)$, respectively. Since $|x'(s)| = 1$, (2.53) gives $\lambda^2=1/|\nabla u|^2$.

By differentiating (2.52), we can write

$$(x_1'(s))^2\,u_{x_1^2}+(x_2'(s))^2\,u_{x_2^2}+2x_1'(s)x_2'(s)u_{x_1x_2}+x_1''(s)\,u_{x_1}+x_2''(s)\,u_{x_2}=0$$

which in conjunction with (2.53) gives

$$\lambda^2\left(u_{x_1}^2\,u_{x_1^2}+u_{x_2}^2\,u_{x_2^2}-2u_{x_1}u_{x_2}u_{x_1x_2}\right)+\frac{1}{\lambda}\left(x_1''(s)\,x_2'(s)-x_2''(s)\,x_1'(s)\right)=0$$

Using the above expression and (2.51), we obtain an expression for the curvature (assuming $|\nabla u| \neq 0$) as

$$\kappa = \frac{u_{x_1}^2 u_{x_1^2} + u_{x_2}^2 u_{x_2^2} - 2u_{x_1} u_{x_2} u_{x_1 x_2}}{\left(u_{x_1}^2 + u_{x_2}^2\right)^{3/2}} \tag{2.54}$$

From (2.54), it can readily be verified that

$$\kappa = \operatorname{div}\left(\frac{\nabla u}{|\nabla u|}\right) \tag{2.55}$$

2.5.3 *Images as Surfaces*

We now examine the case where images are seen as three-dimensional (3-D) surfaces. Let Ω be an open set in R^2 and $S(u,v)$ be a vector valued function which maps Ω to R^3 and defines a regular parameterized surface $\mathcal{S}$. That is, $S(u,v) = [s_1(u,v) \;\; s_2(u,v) \;\; s_3(u,v)]^T$ such that vectors

$$S_u(u,v) = \begin{bmatrix} \frac{\partial s_1}{\partial u} \\ \frac{\partial s_2}{\partial u} \\ \frac{\partial s_3}{\partial u} \end{bmatrix} \quad \text{and} \quad S_v(u,v) = \begin{bmatrix} \frac{\partial s_1}{\partial v} \\ \frac{\partial s_2}{\partial v} \\ \frac{\partial s_3}{\partial v} \end{bmatrix}$$

are linearly independent for every point (u,v) in Ω. Hence $S_u(u,v)$ and $S_v(u,v)$ form a basis in the tangent plane and $N(u,v) = \dfrac{S_u \times S_v}{|S_u \times S_v|}$ is a unit normal vector to surface $\mathcal{S}$. Associated with surface $S(u,v)$ there are two basic forms – the so-called first and second (quadratic) fundamental forms:

• Suppose $S = S(u,v)$ and $\tilde{S} = S(u+\Delta u, v+\Delta v)$ are two pints on surface $\mathcal{S}$, using Taylor expansion we write the vector from point S to point $\tilde{S}$ as

$$\overrightarrow{S\tilde{S}} = S(u+\Delta u, v+\Delta v) - S(u,v) = S_u \Delta u + S_v \Delta v + \cdots$$

If we define ds as the principal component of the length of vector $\overrightarrow{S\tilde{S}}$ in R^3, we have

$$ds^2 = |dS(u,v)|^2 = (S_u du + S_v dv) \cdot (S_u du + S_v dv) = |S_u|^2 (du)^2 + 2S_u \cdot S_v \, du \, dv + |S_v|^2 (dv)^2$$

or

$$ds^2 = E(u,v)\, du^2 + 2F(u,v)\, du\, dv + G(u,v)\, dv^2 \tag{2.56}$$

where $E(u,v) = |S_u|^2$, $F(u,v) = S_u \cdot S_v$, and $G(u,v) = |S_v|^2$. The quadratic differential form in (2.56) is called the *first fundamental form* of surface $\mathcal{S}$.

• To investigate the local geometric shape of a surface $\mathcal{S}$, consider the neighborhood of a point S on $\mathcal{S}$. Let T_s be the tangent plane of $\mathcal{S}$ at point $S = S(u,v)$, we compute the perpendicular distance δ from point $\tilde{S} = S(u+\Delta u, v+\Delta v)$ to plane T_s as follows:

$$\begin{aligned}\overrightarrow{S\tilde{S}} &= S(u+\Delta u, v+\Delta v) - S(u,v) \\ &= S_u \Delta u + S_v \Delta v + \frac{1}{2}\left(S_{uu}(\Delta u)^2 + 2S_{uv}\Delta u \Delta v + S_{vv}(\Delta v)^2\right) + \cdots\end{aligned}$$

and

$$\begin{aligned}2\delta &= 2\overrightarrow{S\tilde{S}} \cdot N \\ &= S_{uu} \cdot N\,(\Delta u)^2 + 2S_{uv} \cdot N\,\Delta u \Delta v + S_{vv} \cdot N\,(\Delta v)^2 + \cdots \\ &\equiv L(u,v)(\Delta u)^2 + 2M(u,v)\Delta u \Delta v + P(u,v)(\Delta v)^2\end{aligned}$$

We call $L(u,v)\,du^2 + 2M(u,v)\,du dv + P(u,v)\,dv^2$ the second fundamental form of surface $\mathcal{S}$, where

$$\begin{aligned}L(u,v) &= S_{uu} \cdot N = -S_u \cdot N_u \\ M(u,v) &= S_{uv} \cdot N = -\frac{1}{2}(S_u \cdot N_v + S_v N_u) \\ P(u,v) &= S_{vv} \cdot N = -S_v \cdot N_v\end{aligned} \tag{2.57}$$

• Using the above notation, we also have

$$\begin{aligned}&\left|S_u \times S_v\right| = \sqrt{EG - F^2} \\ &ds = \sqrt{EG - F^2}\; du\, dv \\ &\text{the mean curvature } H = \frac{EP + GL - 2FM}{2(EG - F^2)}\end{aligned} \tag{2.58}$$

2.6 Finite Difference Methods for PDEs: Basic Concepts

2.6.1 *An Example*

We begin with the one-dimensional heat equation

$$\begin{cases} \dfrac{\partial v}{\partial t} = \mu \dfrac{\partial^2 v}{\partial x^2} \quad \text{for } t > 0,\ x \in R \\ \\ \text{Initial condition: } v(0,x) = f(x) \end{cases} \tag{2.59}$$

where $\mu > 0$ is a constant. The heat equation belongs to the class of *parabolic* PDEs and has been studied thoroughly in the past. In a more abstract setting (2.59) can be put in the form of

$$\begin{cases} Lv = F \quad t > 0, x \in R \\ v(0,x) = f(x) \quad x \in R \end{cases} \tag{2.60}$$

where F is a known function defined in R and L is a linear differential operator that in the present case is given by

$$Lv = \frac{\partial v}{\partial t} - \mu \frac{\partial^2 v}{\partial x^2} \tag{2.61}$$

As the first step in solving the above equation numerically, the domain $\{(t,x): t \geq 0, -\infty < x < +\infty\}$ is discretized by placing a set of grid points with spacing Δx in x and Δt in t (see the figure below).

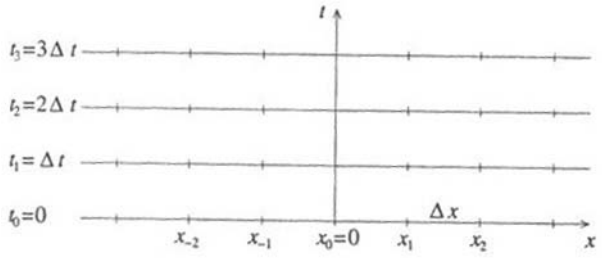


Figure A.1. Grid on the time–space domain.

We then seek for a *discrete function u* which is defined at point $(n\Delta t, i\Delta x)$, denoted by u_i^n, which is a good approximation of the solution $v(t, x)$ of equation (2.60). The function u is obtained as the solution of a discrete equation, which is an approximation of (2.60) and assume the form

$$\begin{cases} L_i^n u_i^n = F_i^n \qquad n = 1,2,\ldots, \; i \in (-\infty, +\infty), \\ u_i^0 = f(i\Delta x) \qquad\qquad i \in (-\infty, +\infty) \end{cases} \tag{2.62}$$

The starting point of developing a useful discrete equation (2.62) with desirable properties such as convergence, consistency, and stability is the Taylor expansion of smooth multivariate functions. In the present case where the spacing Δt and Δx are assumed small, we can write

$$v\big((n+1)\Delta t, i\Delta x\big) = \left(v + \Delta t \frac{\partial v}{\partial t} + \frac{\Delta t^2}{2}\frac{\partial^2 v}{\partial t^2}\right)(n\Delta t, i\Delta x) + O(\Delta t^3) \tag{2.63}$$

$$v\big(n\Delta t, (i+1)\Delta x\big) = \left(v + \Delta x \frac{\partial v}{\partial x} + \frac{\Delta x^2}{2}\frac{\partial^2 v}{\partial x^2}\right)(n\Delta t, i\Delta x) + O(\Delta x^3) \tag{2.64}$$

$$v\big(n\Delta t, (i-1)\Delta x\big) = \left(v - \Delta x \frac{\partial v}{\partial x} + \frac{\Delta x^2}{2}\frac{\partial^2 v}{\partial x^2}\right)(n\Delta t, i\Delta x) + O(\Delta x^3) \tag{2.65}$$

Using (2.63), we have

$$\frac{\partial v}{\partial t}(n\Delta t, i\Delta x) = \frac{v_i^{n+1} - v_i^n}{\Delta t} + O(\Delta t) \tag{2.66}$$

and using (2.64), (2.65), we obtain

$$\frac{\partial^2 v}{\partial x^2}(n\Delta t, i\Delta x) = \frac{v_{i+1}^n - 2v_i^n + v_{i-1}^n}{\Delta x^2} + O(\Delta x^2) \tag{2.67}$$

Combining (2.66) and (2.67) yields

$$\frac{\partial v}{\partial t}(n\Delta t, i\Delta x) - \mu \frac{\partial^2 v}{\partial x^2}(n\Delta t, i\Delta x) = \frac{v_i^{n+1} - v_i^n}{\Delta t} - \mu \frac{v_{i+1}^n - 2v_i^n + v_{i-1}^n}{\Delta x^2} + O(\Delta t) + O(\Delta x^2) \tag{2.68}$$

which suggests a reasonable approximation of equation (2.60) as

$$L_i^n = 0 \quad \text{where} \quad L_i^n = \frac{u_i^{n+1} - u_i^n}{\Delta t} - \mu \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2} \tag{2.69}$$

which can be expressed as

$$u_i^{n+1} = (1-2r)u_i^n + r(u_{i+1}^n + u_{i-1}^n) \quad \text{where} \quad r = \mu\Delta t / \Delta x^2 \tag{2.70}$$

The difference scheme in (2.70) is said to be an *explicit scheme* because the solution at time $(n+1)\Delta t$ are obtained only from the function values at time $n\Delta t$.

We stress that the solution of the discrete equation differs from that of the original equation. This is because during the formation of the discrete equation error has been introduced into the equation (see the approximation made in order to obtain (2.69) (and (2.70)) from (2.68)). Therefore, questions of specifying and quantifying any conditions on grid size and possibly other things for the discrete equation to produce a good approximation of the true solution arise. These and other questions will be addressed as we study the notions of convergence, consistency, and stability of the difference scheme in question.

2.6.2 Convergence

Definition The difference scheme in (2.62) approximating the PDE in (2.60) is said to be pointwise convergent if for any x and t, as $\big((n+1)\Delta t, i\Delta x\big)$ converges to (t, x), u_i^n converges to $v(x, t)$ as Δt and Δx converge to 0.

Example 2.15 Consider the difference scheme for the heat equation as described in (2.70), i.e.,

$$\begin{cases} u_i^{n+1} = (1-2r)u_i^n + r(u_{i+1}^n + u_{i-1}^n) \quad \text{where} \quad x \in R,\ r = \mu\Delta t / \Delta x^2 \\ u_i^0 = f(i\Delta x) \end{cases} \tag{2.71}$$

We show the solution of (2.71) converges pointwise to the solution of the initial-value problem (2.59) if the ratio $r = \mu\Delta t / \Delta x^2$ falls in the range between 0 and ½. To this end we need to estimate

$$z_l^n = u_i^n - v(n\Delta t, i\Delta x)$$

From (2.68) we can write

$$v_i^{n+1} = (1-2r)v_i^n + r(v_{i+1}^n + v_{i-1}^n) + O(\Delta t^2) + O(\Delta t\,\Delta x^2) \tag{2.72}$$

By subtracting (2.72) from (2.71), we obtain

$$z_i^{n+1} = (1-2r)\, z_i^n + r\,(z_{i+1}^n + z_{i-1}^n) + O(\Delta t^2) + O(\Delta t\, \Delta x^2) \tag{2.73}$$

hence

$$\left|z_i^{n+1}\right| \le (1-2r)\left|z_i^n\right| + r\left|z_{i+1}^n\right| + r\left|z_{i-1}^n\right| + C\left(\Delta t^2 + \Delta t\, \Delta x^2\right) \tag{2.74}$$

Now if we denote $Z^n \equiv \sup_{i\in Z}\{\left|z_i^n\right|\}$, then (2.74) implies that

$$Z^{n+1} \le Z^n + C(\Delta t^2 + \Delta t\, \Delta x^2)$$

which in turn gives

$$\begin{aligned} Z^{n+1} &\le Z^n + C(\Delta t^2 + \Delta t\, \Delta x^2) \\ &\le Z^{n-1} + 2C(\Delta t^2 + \Delta t\, \Delta x^2) \\ &\le \cdots \\ &\le Z^0 + (n+1)C\Delta t(\Delta t + \Delta x^2) \end{aligned}$$

Since $Z^0 = 0$, the above estimate leads to

$$\left|u_i^{n+1} - v\big((n+1)\Delta t, i\Delta x\big)\right| \le (n+1)\Delta t\, C\,(\Delta t + \Delta x^2) \tag{2.75}$$

which implies that

$$Z^{n+1} = \left\|z^{n+1}\right\|_{l^\infty} \to 0 \quad \text{as } (n+1)\Delta t \to t, \Delta t \to 0, \text{ and } \Delta x \to 0 \tag{2.76}$$

■

Definition The scheme in (2.62) approximating the PDE in (2.60) is said to be a convergent scheme at time t if as $(n+1)\Delta t \to t$

$$\left\|u^{n+1} - v^{n+1}\right\|_* \to 0 \quad \text{as} \quad \Delta x \to 0 \text{ and } \Delta t \to 0 \tag{2.77}$$

where $\|\cdot\|_*$ is a norm to be specified. Typical norms for $z = (\ldots, z_{-1}, z_0, z_1, \ldots)$ include

$$\|z\|_{l^\infty} = \sup_{i\in Z}\{|z_i|\}, \quad \|z\|_{l^2} = \sqrt{\sum_{i=-\infty}^{\infty}|z_i|^2}, \quad \|z\|_{l^{2,\Delta x}} = \sqrt{\sum_{i=-\infty}^{\infty}|z_i|^2\,\Delta x} \tag{2.78}$$

Definition A difference scheme (2.62) approximating the PDE in (2.60) is said to be a convergent scheme of order (p, q) if for any t, as $(n+1)\Delta t \to t$,

$$\left|u^{n+1} - v^{n+1}\right|_* = O(\Delta x^p) + O(\Delta t^q) \quad \text{as } \Delta x \to 0 \text{ and } \Delta t \to 0 \tag{2.79}$$

Example 2.16 For the scheme (2.71) of the heat equation, we have in Example 2.15 proved its convergence in the l^∞ norm. Moreover, it can be verified that the scheme is of order (2, 1). ■

The convergence of a difference scheme is usually difficult to prove, and most of the time its proof

utilizes the Lax theorem which is stated below.

Theorem 2.8 (P.D. Lax) A consistent two-level difference scheme for a well-posed linear initial problem is convergent if and only if it is stable.
There are two new concepts involved in the theorem, namely, *consistency* and *stability*.

• consistency is concerned with the error introduced by discretizing the equation, and it means that this error should tend to zero as the spacing Δt and Δx go to 0.

• stability means that small errors in the initial condition only cause small errors in the solution, a concept similar to that of well-posedness of a PDE.

2.6.3 Consistency

Definition The scheme in (2.62) approximating the PDE in (2.60) is said to be *pointwise consistent* at point(t, x) if for any smooth function $\varphi = \varphi(t, x)$

$$(L\varphi - F)\big|_i^n - \left[L_i^n \varphi(n\Delta t, i\Delta x) - F_i^n \right] \to 0 \tag{2.80}$$

as $\Delta t, \Delta x \to 0$ and $\big((n+1)\Delta t,\ i\Delta x\big) \to (t, x)$.

Example 2.17 From (2.68) it follows immediately that the scheme in (2. 70) is pointwise consistent with the PDE in (2.59). ■

To define consistency in terms of norms, we write a two-level difference scheme in the form of

$$u^{n+1} = Qu^n + \Delta t G^n \tag{2.81}$$

where $u^n = \left(\cdots, u_{-1}^n, u_0^n, u_1^n, \cdots\right)$, and $G^n = \left(\cdots, G_{-1}^n, G_0^n, G_1^n, \cdots\right)$, and Q is an operator.

Definition The scheme (2.62) is consistent with the PDE in (2.60) in a norm $\|\cdot\|_*$ if the solution v of the PDE satisfies

$$v^{n+1} = Qv^n + \Delta t G^n + \Delta t\, \tau^n, \quad \text{with } \left\|\tau^n\right\|_* \to 0$$

as $\Delta t, \Delta x \to 0$. The term τ^n is called the *truncature* term.

Definition The scheme (2.62) is said to be accurate of order (p, q) if

$$\left\|\tau^n\right\|_* = O(\Delta x^p) + O(\Delta t^q)$$

It can be readily verified that if a scheme is accurate of order (p, q) with $p, q \geq 1$, then it is a consistent scheme. Also, a scheme which is either consistent or accurate of order (p, q) is also pointwise consistent.

Example 2.18 To examine the consistency of the scheme

$$\frac{u_i^{n+1}-u_i^n}{\Delta t}=\mu\frac{u_{i+1}^n-2u_i^n+u_{i-1}^n}{\Delta x^2}$$

with the heat equation. Assume $v(t, x)$ is the solution of the PDE, then (2.68) implies that

$$\frac{v_i^{n+1}-v_i^n}{\Delta t}-\mu\frac{v_{i+1}^n-2v_i^n+v_{i-1}^n}{\Delta x^2}=O(\Delta t)+O(\Delta x^2)$$

The difference scheme in question can be written in the form of (2.81) as (2.70), i.e.

$$u_i^{n+1}=(1-2r)u_i^n+r(u_{i+1}^n+u_{i-1}^n)\quad\text{where}\quad r=\mu\Delta t/\Delta x^2$$

hence the truncature error is given by

$$\Delta t\,\tau_i^n=v_i^{n+1}-\left\{(1-2r)v_i^n+r(v_{i+1}^n+v_{i-1}^n)\right\} \tag{2.82}$$

by which a useful form of τ^n can be obtained using higher-order Taylor expansion of its right-hand term as follows.

$$v\big((n+1)\Delta t,i\Delta x\big)=\left(v+\Delta t\frac{\partial v}{\partial t}\right)(n\Delta t,i\Delta x)+\frac{\Delta t^2}{2}\frac{\partial^2 v}{\partial t^2}(t_1,i\Delta x) \tag{2.83}$$

$$v\big(n\Delta t,(i+1)\Delta x\big)=\left(v+\Delta x\frac{\partial v}{\partial x}+\frac{\Delta x^2}{2}\frac{\partial^2 v}{\partial x^2}+\frac{\Delta x^3}{6}\frac{\partial^3 v}{\partial x^3}\right)(n\Delta t,i\Delta x)+\frac{\Delta x^4}{24}\frac{\partial^4 v}{\partial x^4}(n\Delta t,x_1) \tag{2.84}$$

$$v\big(n\Delta t,(i-1)\Delta x\big)=\left(v-\Delta x\frac{\partial v}{\partial x}+\frac{\Delta x^2}{2}\frac{\partial^2 v}{\partial x^2}-\frac{\Delta x^3}{6}\frac{\partial^3 v}{\partial x^3}\right)(n\Delta t,i\Delta x)+\frac{\Delta x^4}{24}\frac{\partial^4 v}{\partial x^4}(n\Delta t,x_2) \tag{2.85}$$

where t_1 in (2.83) is a value between $n\Delta t$ and $(n+1)\Delta t$, x_1 in (2.84) is a value between $i\Delta x$ and $(i+1)\Delta x$, and x_2 in (2.85) is a value between $(i-1)\Delta x$ and $i\Delta x$. Using (2.83) – (2.85) and the fact that $v(t, x)$ is the solution of the heat equation, we can right the right-hand term of (2.82) as $\Delta t\,\tau_i^n$ where

$$\tau_i^n=\frac{\partial^2 v}{\partial t^2}(t_1,i\Delta x)\frac{\Delta t}{2}+\left(\frac{\partial^4 v}{\partial x^4}(n\Delta t,x_1)+\frac{\partial^4 v}{\partial x^4}(n\Delta t,x_2)\right)\frac{\Delta x^2}{24} \tag{2.86}$$

The expression in (2.86) explicitly indicates that the terms previously denoted by $O(\Delta t)$ and $O(\Delta x^2)$ etc. are such that may contain non-constant coefficients so smoothness assumptions may be needed to ensure the scheme's consistency. In the present case, we assume that both $\partial^2 v/\partial t^2$ and $\partial^4 v/\partial x^4$ are uniformly bounded in $[0, T] \times R$ for some $T > 0$. Then we can choose the sup-norm and claim that the scheme in question is accurate of order (2, 1). ■

2.6.4 Stability

Definition The two-level difference scheme

$$\begin{cases} u^{n+1} = Qu^n, \quad n > 0 \\ u^0 \text{ given} \end{cases} \tag{2.87}$$

where $u^n = \left(\cdots, u_{-1}^n, u_0^n, u_1^n, \cdots\right)$ is said to be stable with respect to norm $\|\cdot\|_*$ if there exist constants Δx_0 and Δt_0 and nonnegative constants K and β such that

$$\left\|u^{n+1}\right\|_* \le K e^{\beta t} \left\|u^0\right\|_* \tag{2.88}$$

for $0 \le t = (n+1)\Delta t, 0 < \Delta x \le \Delta x_0,$ and $0 < \Delta t \le \Delta t_0$.

• We note that the above definition is for homogeneous schemes. For a non-homogeneous scheme, its convergence can be ensured by proving the stability and convergence of the associated homogeneous scheme.

• Like convergence and consistency, we need to specify the norm to be used for stability analysis of a scheme.

• Also note that the above definition of stability allows the solution to grow with time.

Another commonly used definition of stability, which does not allow for exponential growth of the solution, is one that replaces (2.88) by

$$\left\|u^{n+1}\right\|_* \le K \left\|u^0\right\|_* \tag{2.89}$$

Obviously this definition is more restrictive relative to that of (2.88) and can be viewed as a special case of the previous definition with $\beta = 0$.

Example 2.19 Consider again the scheme

$$u_i^{n+1} = (1-2r)u_i^n + r(u_{i+1}^n + u_{i-1}^n) \quad \text{where} \quad r = \mu\Delta t / \Delta x^2 \le 1/2 \tag{2.90}$$

from which we have

$$\left|u_i^{n+1}\right| = (1-2r)\left|u_i^n\right| + r\left|u_{i+1}^n\right| + r\left|u_{i-1}^n\right| \le \left\|u^n\right\|_{l^\infty}$$

which implies that

$$\left\|u^{n+1}\right\|_{l^\infty} \le \left\|u^n\right\|_{l^\infty}$$

Therefore (2.88) is satisfied with $K = 1$ and $\beta = 0$. Since the stability is proved under the assumption that $r = \mu\Delta t / \Delta x^2 \le 1/2$, we say in this case the scheme is conditionally stable. ■

A useful tool for studying stability of difference schemes is Fourier analysis. Recall the Fourier and inverse Fourier transform of a continuous function $v(t, x)$ with respect to variable x are given by

$$\hat{v}(t,\omega)=\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty}e^{-j\omega x}v(t,x)dx \quad \text{and} \quad v(t,\omega)=\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty}e^{j\omega x}\hat{v}(t,x)d\omega$$

respectively. A property of the Fourier transform that is of particular use in the stability analysis is the well known Parseval identity which can be stated as

$$\|v\|_{L^2(R)}=\|\hat{v}\|_{L^2(R)}$$

Another useful property of the Fourier transform is that

$$\hat{v}_x(t,\omega)=-j\omega\hat{v}(t,\omega) \quad \text{and} \quad \hat{v}_{xx}(t,\omega)=-\omega^2\hat{v}(t,\omega) \tag{2.91}$$

The Fourier and inverse Fourier transform of a discrete sequence $\{u_m\}$ are defined by

$$\hat{u}(\xi)=\frac{1}{\sqrt{2\pi}}\sum_{m=-\infty}^{\infty}e^{-jm\xi}u_m \quad \text{and} \quad u_m=\frac{1}{\sqrt{2\pi}}\int_{-\pi}^{\pi}e^{jm\xi}\hat{u}(\xi)\,d\xi$$

respectively. The Parseval identity in this case is given by

$$\|u\|_{l^2}=\|\hat{u}\|_{L^2(-\pi,\pi)}$$

By definition it is required for a stable scheme to obey

$$\left\|u^{n+1}\right\|_{l^2}\le K\,e^{\beta(n+1)\Delta t}\left\|u^0\right\|_{l^2} \tag{2.92}$$

which is by Parseval identity equivalent to

$$\left\|\hat{u}^{n+1}\right\|_{L^2(-\pi,\pi)}\le K\,e^{\beta(n+1)\Delta t}\left\|\hat{u}^0\right\|_{L^2(-\pi,\pi)} \tag{2.93}$$

Thus the sequence $\{u^n\}$ is stable in l^2 if and only if sequence $\{\hat{u}^n\}$ is stable in $L^2(-\pi,\pi)$. Finally, we remark that there are discrete counterparts analogous to those in (2.91) as we will see in the next example.

Example 2.20 Here we examine the stability of the difference scheme

$$u_k^{n+1}=r\,u_{k+1}^n+(1-2r)u_k^n+r\,u_{k-1}^n \tag{2.94}$$

where $r=\mu\Delta t/\Delta x^2\le 1/2$. By applying Fourier transform to (2.94), we obtain

$$\hat{u}^{n+1}(\xi) = \frac{r}{\sqrt{2\pi}} \sum_{k=-\infty}^{\infty} e^{-jk\xi} u_{k+1}^n + (1-2r)\hat{u}^n(\xi) + \frac{r}{\sqrt{2\pi}} \sum_{k=-\infty}^{\infty} e^{-jk\xi} u_{k-1}^n$$
$$= \left(1 - 4r \sin^2\left(\frac{\xi}{2}\right)\right) \hat{u}^n(\xi)$$
$$= \left(1 - 4r \sin^2\left(\frac{\xi}{2}\right)\right)^{n+1} \hat{u}^0(\xi)$$
$$\equiv \left(\rho(\xi)\right)^{n+1} \hat{u}^0(\xi)$$

hence the condition in (2.93) will be satisfied (thus the stability of the scheme) with $K = 1$ and $\beta = 0$ if

$$\left|1 - 4r \sin^2\left(\frac{\xi}{2}\right)\right| \le 1 \quad \text{for} \quad \text{all } \xi \in [-\pi, \pi] \tag{2.95}$$

The condition in (2.95) is met if $r = \mu\Delta t / \Delta x^2 \le 1/2$. ■

Another popular technique for stability analysis of difference schemes, known as the *discrete von Neumann criterion*, is to consider a discrete Fourier mode

$$u_k^n = \xi^n e^{jmk\pi\Delta x} \quad \text{for } 0 \le m \le M$$

The idea is to insert this Fourier mode into the difference scheme and find the expression for ξ. A necessary condition for stability is obtained by restricting Δx and Δt so that $|\xi| \le 1$ which implies that term ξ^n will not grow without bound.

Example 2.21 We illustrate the discrete von Neumann criterion by applying it to the scheme

$$u_k^{n+1} = r\,u_{k+1}^n + (1-2r)\,u_k^n + r\,u_{k-1}^n \tag{2.96}$$

By inserting $u_k^n = \xi^n e^{jmk\pi\Delta x}$ into (2.96), we obtain

$$\xi^{n+1} e^{jmk\pi\Delta x} = \xi^n e^{jmk\pi\Delta x} \left(r\,e^{-jm\pi\Delta x} + (1-2r) + r\,e^{jm\pi\Delta x}\right)$$

which leads to

$$\xi = r\,e^{-jm\pi\Delta x} + (1-2r) + r\,e^{jm\pi\Delta x} = 1 - 4r \sin^2\left(\frac{m\pi\Delta x}{2}\right)$$

By requesting that $|\xi| \le 1$, we recover the result in (2.95), i.e.,

$$\left|1 - 4r \sin^2\left(\frac{m\pi\Delta x}{2}\right)\right| \le 1$$

which is satisfied if $r = \mu\Delta t / \Delta x^2 \le 1/2$. ■

2.6.5 *Explicit Schemes versus Implicit Schemes*

The discretization scheme used above for the heat equation is an *explicit scheme* as the values of *u* at $(n+1)\Delta t$ are fully determined by the values of *u* at time $n\Delta t$. One may consider more general schemes such as

$$u_i^{n+1} = u_i^n + \frac{\mu\Delta t}{h^2}\left(\lambda\left(u_{i+1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}\right) + (1-\lambda)\left(u_{i+1}^n - 2u_i^n + u_{i-1}^n\right)\right) \tag{2.97}$$

for the heat equation, where λ is a scalar parameter. If $\lambda \neq 0$, then (2.97) is an *implicit scheme* because the values of *u* at $((n+1)\Delta t, i\Delta x)$ are dependent not only on the values of *u* at time $n\Delta t$ but also on the values of *u* at $(n+1)\Delta t$ that located at positions other than $i\Delta x$. As a result, one needs to solve a linear system of equations in order to obtain the solution at time $(n+1)\Delta t$.
For $\lambda = 1$, (2.97) is fully implicit and for $\lambda = 0.5$ one obtains the so-called Crank-Nicholson scheme. Among other things, whether to use an explicit or an implicit scheme has an impact on stability and consistency. For example, for the scheme in (2.97) one can conclude the following:

- For $\lambda = 0$, the scheme is explicit (of order $O(\Delta t, h^2)$) and stable under the condition $\Delta t \le \frac{h^2}{2\mu}$. This condition implies that the time step has to be sufficiently small to ensure the convergence and stability. Obviously, choosing a small time step slows down the solution process.

- For $\lambda = 1/2$, the scheme is implicit (of order $O(\Delta t^2, h^2)$) and unconditionally stable.

- For $\lambda > 1/2$, the scheme is implicit (of order $O(\Delta t, h^2)$) and unconditionally stable.

2.6.6 *Difference Schemes for Hyperbolic Equations*

Consider the initial-value problem of the 1-D linear advection (also called transport or wave) equation:

$$\begin{cases} \dfrac{\partial v}{\partial t}(t,x) + a\dfrac{\partial v}{\partial x}(t,x) = 0, \quad t > 0,\ x \in R \\ v(0,x) = v_0(x) \end{cases} \tag{2.98}$$

where *a* is a constant. It can be verified that the solution of (2.98) assumes the form

$$v(t,x) = v_0(x - at) \tag{2.99}$$

hence, along lines of slope *a* (called characteristics), *v*(*t*, *x*) remains constant. This means that the information is propagated in the direction of the sign of *a*, e.g. from left to right if *a* is positive.

To solve Eq. (2.98) numerically, we use the Taylor expansion of *v*(*t*, *x*) as we did for the heat equation:

$$v\left((n+1)\Delta t, i\Delta x\right) = \left(v + \Delta t\frac{\partial v}{\partial t} + \frac{\Delta t^2}{2}\frac{\partial^2 v}{\partial t^2}\right)(n\Delta t, i\Delta x) + O(\Delta t^3) \tag{2.100}$$

$$v\left(n\Delta t,(i+1)\Delta x\right)=\left(v+\Delta x\frac{\partial v}{\partial x}+\frac{\Delta x^2}{2}\frac{\partial^2 v}{\partial x^2}\right)(n\Delta t,i\Delta x)+O(\Delta x^3) \tag{2.101}$$

$$v\left(n\Delta t,(i-1)\Delta x\right)=\left(v-\Delta x\frac{\partial v}{\partial x}+\frac{\Delta x^2}{2}\frac{\partial^2 v}{\partial x^2}\right)(n\Delta t,i\Delta x)+O(\Delta x^3) \tag{2.102}$$

The temporal derivative can be approximated using (2.100) as

$$\frac{\partial v}{\partial t}(n\Delta t,i\Delta x)=\frac{v_i^{n+1}-v_i^n}{\Delta t}+O(\Delta t) \tag{2.103}$$

For the spatial derivative, there are several options:

- approximation of the derivative based on *forward difference* using (2.101) as

$$\frac{\partial v}{\partial x}(n\Delta t,i\Delta x)=\frac{v_{i+1}^n-v_i^n}{\Delta x}+O(\Delta x)$$

- approximation of the derivative based on *backward difference* using (2.102) as

$$\frac{\partial v}{\partial x}(n\Delta t,i\Delta x)=\frac{v_i^n-v_{i-1}^n}{\Delta x}+O(\Delta x)$$

- approximation of the derivative based on *centered difference* by subtracting (2.102) from (2.101) as

$$\frac{\partial v}{\partial x}(n\Delta t,i\Delta x)=\frac{v_{i+1}^n-v_{i-1}^n}{2\Delta x}+O(\Delta x^2)$$

which lead to three different schemes of equation (2.98) as

$$u_i^{n+1}=u_i^n+a\,\Delta t\begin{cases}\delta_x^+u_i^n & \left(\equiv\dfrac{u_{i+1}^n-u_i^n}{\Delta x}\right) \quad \text{forward scheme}\\ \delta_x u_i^n & \left(\equiv\dfrac{u_{i+1}^n-u_{i-1}^n}{2\Delta x}\right) \quad \text{centered scheme}\\ \delta_x^-u_i^n & \left(\equiv\dfrac{u_i^n-u_{i-1}^n}{\Delta x}\right) \quad \text{backward scheme}\end{cases}$$

First we examine the centered scheme of the equation, i.e.,

$$u_i^{n+1}=u_i^n-a\,\Delta t\delta_x u_i^n=u_i^n-\frac{a\,\Delta t}{2\Delta x}\left(u_{i+1}^n-u_{i-1}^n\right) \tag{2.104}$$

By inserting the discrete Fourier mode $u_k^n=\xi^n e^{jmk\pi\Delta x}$ into (2.104), we obtain

$$\xi = 1 - j\frac{a\,\Delta t}{\Delta x}\sin(m\pi\Delta x) \quad \Rightarrow \quad |\xi|^2 = 1 + \left(\frac{a\,\Delta t}{\Delta x}\right)^2 \sin^2(m\pi\Delta x) \geq 1$$

hence scheme (2.104) is unstable. The reason for this instability is that the nature of the equation, that there is an information propagation in certain direction, was not taken into account in scheme (2.104).

To address the stability problem we consider the following scheme

$$u_i^{n+1} = u_i^n - \begin{cases} a\,\Delta t\,\delta_x^- u_i^n & \text{if } a > 0 \\ a\,\Delta t\,\delta_x^+ u_i^n & \text{if } a < 0 \end{cases}$$

which can be expressed as

$$u_i^{n+1} = u_i^n - \Delta t\left[\max(0,a)\delta_x^- u_i^n + \min(0,a)\delta_x^+ u_i^n\right] \tag{2.105}$$

In the literature the scheme in (2.105) is called an *upwind scheme* because it uses values in the direction of information propagation.

To examine the stability of scheme (2.105), we consider the following two cases:

Case 1: $a > 0$. By substitute $u_k^n = \xi^n e^{jmk\pi\Delta x}$ into (2.105), we obtain

$$|\xi| = 1 - 2C(1-C)\big(1-\cos(m\pi\Delta x)\big) \quad \text{with} \quad C = \frac{a\Delta t}{\Delta x} > 0$$

which implies that $|\xi| \leq 1$ if and only if $C \leq 1$.

Case 2: $a < 0$. With similar calculations we can conclude that $|\xi| \leq 1$ if and only if $-C \leq 1$.
Therefore, the stability condition for scheme (2.105) is give by

$$|a|\frac{\Delta t}{\Delta x} \leq 1 \tag{2.106}$$

which is often called a CFL condition after the work of Courant, Friedrichs, and Levy (1928).

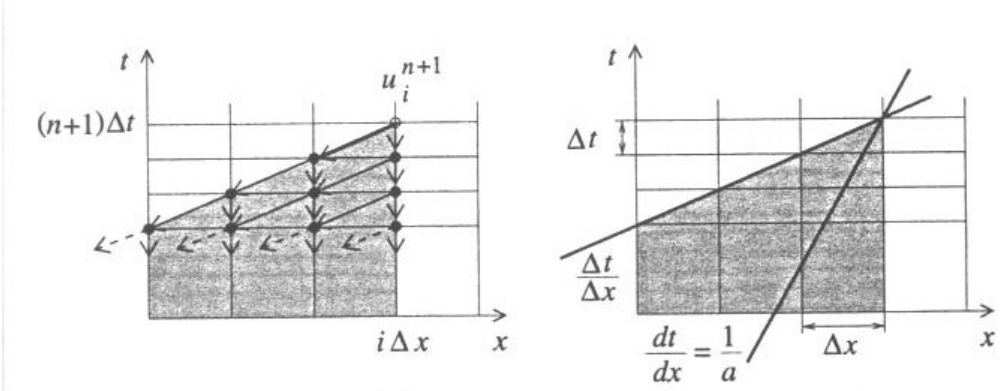
The CFL condition may be interpreted in terms of *domain of dependence*. In the continuous case, the information is propagated along the characteristics and their equation is $\frac{dt}{dx} = \frac{1}{a}$. In the discrete case with $a > 0$, (2.105) can be written as

$$u_i^{n+1} = \left(1 - \frac{a\,\Delta t}{\Delta x}\right)u_i^n + \frac{a\,\Delta t}{\Delta x}u_{i-1}^n$$

This allows us to define the *discrete domain of dependence* of u_i^{n+1}: u_i^{n+1} depends on u_i^n and u_{i-1}^n, u_i^n depends on u_i^{n-1} and u_{i-1}^{n-1}, etc. (see the figure below). Moreover, by the CFL condition we have

$$\frac{\Delta t}{\Delta x} \le \frac{1}{a} = \frac{dt}{dx}$$

which signifies that the characteristic line has to be confined to within the discrete domain of dependence. In other words, to maintain stability the discrete domain of dependence must contain the exact continuous domain of dependence.



(a) Discrete domain of dependence of u_i^{n+1} for (A.47) with $a > 0$

(b) Interpretation of the CFL condition

Figure A.3. Definition and interpretation of the discrete domain of dependence.

2.7 Finite Difference Methods for PDEs: Schemes for Image Analysis

2.7.1 *Discretizing PDEs*

A generic form of the PDEs to be studies in this course assumes the form

$$\begin{cases} Lv = F \qquad (t,x,y) \in R^+ \times \Omega \\ \dfrac{\partial v}{\partial N}(t,x,y) = 0 \quad \text{on} \quad R^+ \times \partial\Omega \quad \text{(Neumann boundary condition)} \\ v(0,x,y) = f(x,y) \quad \text{(initial condition)} \end{cases} \tag{2.107}$$

where Ω is the image domain and N is the normal to the boundary of Ω, denoted by $\partial\Omega$, and L is a second-order differential operator such as

$$\frac{\partial v}{\partial t}(t,x,y) + H\left(x,y,v(t,x,y),\nabla v(t,x,y),\nabla^2 v(t,x,y)\right) = 0$$

Example 2.22 The simplest PDE encountered in image analysis is perhaps the heat equation:

$$\begin{cases} \dfrac{\partial v}{\partial t} = \mu \Delta v = \mu \left(\dfrac{\partial^2 v}{\partial x^2} + \dfrac{\partial^2 v}{\partial y^2} \right) & (t, x, y) \in R^+ \times \Omega \\ \dfrac{\partial v}{\partial N}(t, x, y) = 0 \quad \text{on} \quad R^+ \times \partial\Omega & \\ v(0, x, y) = f(x, y) & \end{cases} \tag{2.108}$$

where μ is a constant. ■

For image analysis purposes, typically a uniform grid is associated with an image with the grid spacing

$$\Delta x = \Delta y = h$$

Often times $h = 1$ is chosen, meaning that the pixel size is chosen as the unit of reference. The position (ih, jh) are called vertices, node, or pixels interchangeably. We use $v_{i,j}^n$ and $u_{i,j}^n$ to denote the values of the exact solution and the discrete solution at location (ih, jh) and time $n\Delta t$, respectively. The figure below illustrates the convention of our discretization process.

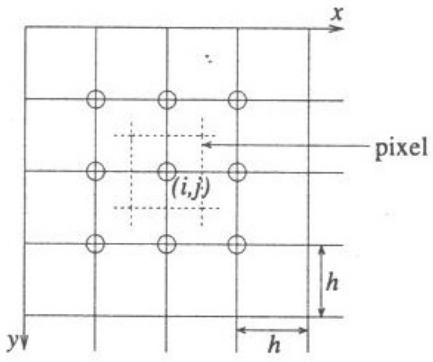


Figure A.5. Grid on the space domain. The circles indicate the vertices that belong to the 3 × 3 neighborhood of the vertex (i, j).

Example 23 To discretize the PDE in (2.108), we need to consider the following:

• The equation. As for the 1-D case, we can proceed by using Taylor expansions of $v(t, x, y)$ about point $(n\Delta t, ih, jh)$. If the second-order derivatives in x and y are discretized separately, we obtain

$$\frac{\partial v}{\partial t} - \mu \Delta v \Big|_{i,j}^{n} = \frac{v_{i,j}^{n+1} - v_{i,j}^{n}}{\Delta t} - \mu \frac{v_{i+1,j}^{n} + v_{i-1,j}^{n} + v_{i,j+1}^{n} + v_{i,j-1}^{n} - 4v_{i,j}^{n}}{h^2} + O(\Delta t) + O(h^2)$$

which suggests the difference scheme

$$u_{i,j}^{n+1} = u_{i,j}^{n} + \frac{\mu \Delta t}{h^2} \left(u_{i+1,j}^{n} + u_{i-1,j}^{n} + u_{i,j+1}^{n} + u_{i,j-1}^{n} - 4u_{i,j}^{n} \right) \tag{2.109}$$

• The boundary condition. The Neumann boundary condition can be taken into account by a symmetry

procedure. If the value of a pixel outside the domain is needed, we use the value of the pixel that is symmetric with respect to the boundary.

- The initial condition is discretized as $u_{i,j}^0 = f_{i,j} = f(ih, jh)$.

The figure below shows the effect of an isotropic diffusion of a simple image via the heat equation.

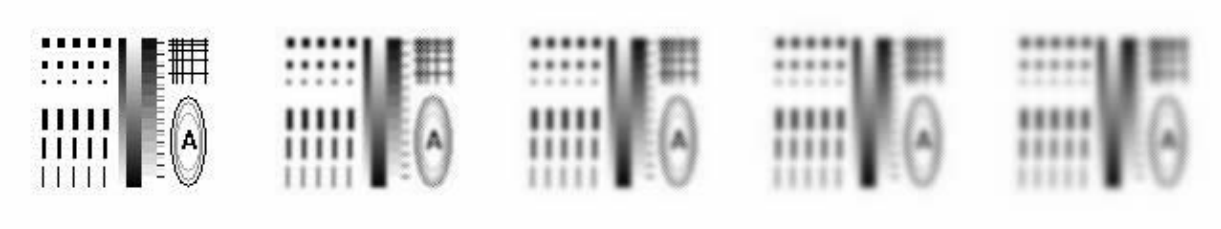


Figure A.6 (new): Example of isotropic diffusion via heat equation as applied the resolution chart of size 128 by 128 with $\mu = 5$, $\Delta t = 0.05$, $h = 1$, and $n = 0$, 2, 4, 6, and 8.

We remark that the scheme in (2.109) is obtained by discretizing the Laplacian as a sum of second-order derivatives in x and y directions as

$$\Delta v|_{i,j} \approx \frac{v_{i+1,j}^n + v_{i-1,j}^n + v_{i,j+1}^n + v_{i,j-1}^n - 4v_{i,j}^n}{h^2} \qquad (2.110)$$

which does not take the 2-D nature of the Laplacian into account. The term "2-D nature" is taken to mean that the Laplacian operator is rotationally invariant, i.e., if we apply a rotation of centre (x, y) to image v with an angle $\theta \in [0, 2\pi)$, then $\Delta v(x, y)$ remains constant for all θ. As the discrete domain is concerned, typically we may require that $\Delta v|_{i,j}$ remain invariant under rotations of $\pi/4$. As we can see from the figure below, the invariance does not hold for the scheme in (2.110) since we would get 1 o 2 depending on the situation.

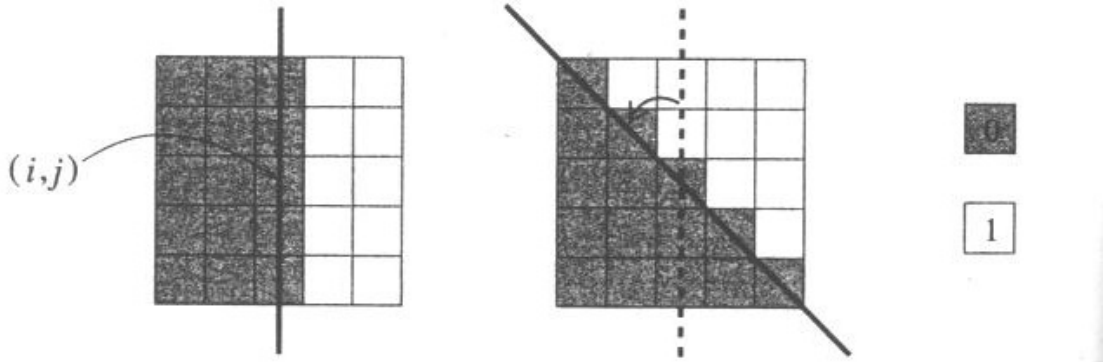


Figure A.7. Example of a binary image representing a vertical edge and the same image after a rotation of $\pi/4$ radians. Rotationally invariant operators estimated at the point in the middle should yield the same value in both situations.

One way to overcome the difficulty is to use the complete 3×3 neighborhood as follows:

$$\Delta v|_{i,j} \approx \lambda \frac{v_{i+1,j} + v_{i-1,j} + v_{i,j+1} + v_{i,j-1} - 4v_{i,j}}{h^2} + (1-\lambda) \frac{v_{i+1,j+1} + v_{i-1,j+1} + v_{i+1,j-1} + v_{i-1,j-1} - 4v_{i,j}}{2h^2} \qquad (2.111)$$

where $\lambda \in [0,1]$ is a constant to be chosen. It can be verifies that (2.111) is a consistent scheme and that for the two situations in the above figure to produce identical results, we need to choose λ = 1/3 which gives

$$\Delta v|_{i,j} \approx \frac{v_{i+1,j+1} + v_{i-1,j+1} + v_{i+1,j-1} + v_{i-1,j-1} + v_{i+1,j} + v_{i-1,j} + v_{i,j+1} + v_{i,j-1} - 8v_{i,j}}{3h^2}$$

For the discretization of first-order derivatives in x and y, a "1-D" centered approximation of $\partial v / \partial x$ is given by

$$\left.\frac{\partial v}{\partial x}\right|_{i,j} \approx \delta_x v_{i,j} = \frac{v_{i+1,j} - v_{i-1,j}}{2h} \tag{2.112}$$

that does not take the data's 2-D nature into consideration. This becomes more apparent as we examine the norm of the gradient of u in the two scenarios in Fig. A.7: we obtain either 1/2 or $1/\sqrt{2}$. One way to address the problem is to use more pixels in the approximation as in the following scheme:

$$\left.\frac{\partial v}{\partial x}\right|_{i,j} \approx \lambda \frac{v_{i+1,j} - v_{i-1,j}}{2h} + \frac{(1-\lambda)}{2}\left(\frac{v_{i+1,j+1} - v_{i-1,j+1}}{2h} + \frac{v_{i+1,j-1} - v_{i-1,j-1}}{2h}\right) \tag{2.113}$$

where λ is a parameter to be chosen. By applying (2.113) to the two scenarios and request that the results be identical to each other yields $\lambda = \sqrt{2} - 1$.

We remark that using more points is generally good not only for rotation-invariance but also for robustness of the results against noise because using more points is somewhat equivalent to smooth the data before the approximation.

2.7.2 *Image Restoration by Energy Minimization*

Here we consider the image restoration method presented in Sec. 3.1.3, where a key problem is to minimize the functional

$$J(v,b) = \frac{1}{2}\iint_\Omega \left(Rv - v_0\right)^2 dx\,dy + \lambda \iint_\Omega \left(b\,|\nabla v|^2 + \psi(b)\right) dx\,dy \tag{2.114}$$

with respect to functions *v* and *b*. The half-quadratic minimization algorithm developed there can be outlined as follows:

For given (v^0, b^0), do:

Step 1: $v^{n+1} = \arg\min_v J(v, b^n)$, i.e.,

$$\begin{cases} R^* R v^{n+1} - \operatorname{div}\left(b^n \nabla v^{n+1}\right) = R u_0 \quad \text{in } \Omega \\ \left. b^n \dfrac{\partial v^{n+1}}{\partial N}\right|_{\partial\Omega} = 0 \end{cases}$$

Step 2: $b^{n+1} = \arg\min_b J(v^{n+1}, b)$, i.e., $b^{n+1} = \dfrac{\varphi'\left(|\nabla v^{n+1}|\right)}{2|\nabla v^{n+1}|}$.

Step 3: Set $n := n + 1$ and repeat from step 1 until convergence is achieved.

For discretization of the equations involved, the only term that may be difficult to deal with is the divergence operator. More specifically the problem to address here for given b and v to approximate $\operatorname{div}\left(b\nabla v\right)$ at node (i, j).

We start by writing the divergence operator as

$$\operatorname{div}\left(b\nabla v\right) = \frac{\partial}{\partial x}\left(b\frac{\partial v}{\partial x}\right) + \frac{\partial}{\partial y}\left(b\frac{\partial v}{\partial y}\right)$$

A difference scheme for $\operatorname{div}\left(b\nabla v\right)$ using 3 × 3 neighborhood and combining forward and backward differences is given by

$$\begin{aligned}\operatorname{div}\left(b\nabla v\right)\big|_{i,j} &\approx \delta_x^+(b_{i,j}\delta_x^- v_{i,j}) + \delta_y^+(b_{i,j}\delta_y^- v_{i,j}) \\ &= \frac{1}{h^2}\left(b_{i+1,j}v_{i+1,j} + b_{i,j}v_{i-1,j} + b_{i,j+1}v_{i,j+1} + b_{i,j}v_{i,j-1} - \left(b_{i+1,j} + b_{i,j+1} + 2b_{i,j}\right)v_{i,j}\right)\end{aligned}$$

A drawback of the above scheme is its asymmetry as the values of b at $((i-1)h, jh)$ and $(ih, (j-1)h)$ have not been used. A solution to the problem is to use the following approximation for the derivatives:

$$\delta_x^* v_{i,j} = \frac{v_{i+1/2,j} - v_{i-1/2,j}}{h} \quad \text{and} \quad \delta_y^* v_{i,j} = \frac{v_{i,j+1/2} - v_{i,j-1/2}}{h}$$

where $v_{i\pm1/2,j\pm1/2}$ is the value of v at $\left(\left(i \pm \frac{1}{2}\right)h, \left(j \pm \frac{1}{2}\right)h\right)$ that can be obtained by interpolation. We then have

$$\begin{aligned}\operatorname{div}\left(b\nabla v\right)\big|_{i,j} &\approx \delta_x^*(b_{i,j}\delta_x^* v_{i,j}) + \delta_y^*(b_{i,j}\delta_y^* v_{i,j}) \\ &= \frac{1}{h^2}\left(b_{+0}v_{i+1,j} + b_{-0}v_{i-1,j} + b_{0+}v_{i,j+1} + b_{0-}v_{i,j-1} - \left(b_{+0} + b_{-0} + b_{0+} + b_{0-}\right)v_{i,j}\right)\end{aligned} \tag{2.115}$$

Note that the scheme in (2.115) uses the values of v at $\left((i \pm 1)h, (j \pm 1)h\right)$, but interpolation is needed for b. In order to take into account the diagonal values, we may consider employing

$$\begin{aligned}&\operatorname{div}\left(b\nabla v\right)\big|_{i,j} \approx \\ &= \frac{\lambda_p}{h^2}\left(b_{+0}v_{i+1,j} + b_{-0}v_{i-1,j} + b_{0+}v_{i,j+1} + b_{0-}v_{i,j-1} - \beta_p v_{i,j}\right) \\ &+ \frac{\lambda_d}{h^2}\left(b_{++}v_{i+1,j+1} + b_{--}v_{i-1,j-1} + b_{-+}v_{i-1,j+1} + b_{+-}v_{i+1,j-1} - \beta_d v_{i,j}\right)\end{aligned} \tag{2.116}$$

where $b_{\pm,\pm} = b_{i\pm1/2, j\pm1/2}$,

$$\begin{cases} \beta_p = b_{0+} + b_{0-} + b_{+0} + b_{-0} \\ \beta_d = b_{++} + b_{--} + b_{+-} + b_{-+} \end{cases}$$

and λ_p and λ_d are two weights to be chosen. The first condition for determining the weights is that the scheme must be consistent, and it can be verified that this means that

$$\lambda_p + 2\lambda_d = 1 \tag{2.117}$$

Possible ways to determine these weights include:

- choose (λ_p, λ_d) to be constants, e.g., $(\lambda_p, \lambda_d) = (1/2,\ 1/4)$.
- choose (λ_p, λ_d) by taking into account the orientation of the gradient of *v* as shown in the figure below.

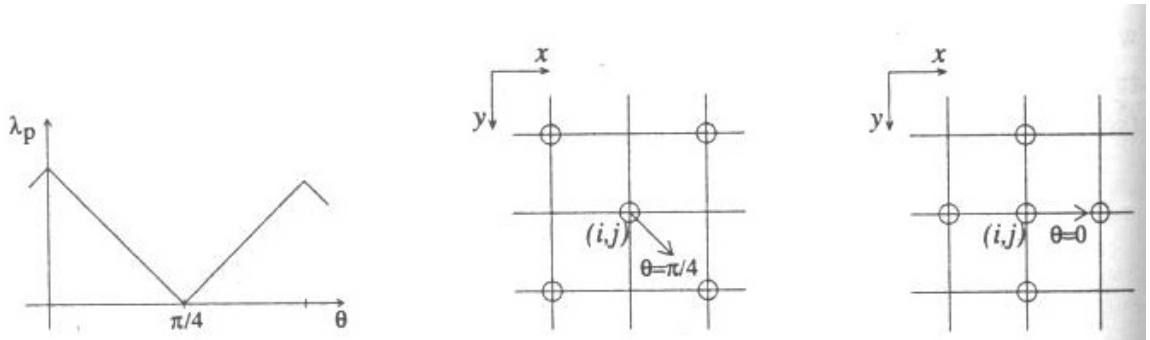


Figure A.11. Adaptive choice of the coefficients (λ_p, λ_d) as a function of θ, the orientation of ∇v. The two right-hand figures show which points will be used in the discretization of the divergence term in two specific situations.

2.7.3 *Curve Evolution with the Level-Set Method*

Here we address the discretization of PDEs governing curve evolution, where the curves are identified as level sets of function u(t, x, y). To this end we shall examine the geodesic active contours model given by

$$\frac{\partial v}{\partial t} = g\left(|\nabla I|\right)|\nabla v|\,\mathrm{div}\left(\frac{\nabla v}{|\nabla v|}\right) + \alpha\, g\left(|\nabla I|\right)|\nabla v| + \nabla g \cdot \nabla v \tag{2.118}$$

Eq. (2.118) involves two types of terms: the first term is of parabolic type and the last two terms are of hyperbolic type. Therefore the discretization of these terms in the PDE needs to be done appropriately in accordance with their nature. The main idea is that the parabolic term can be discretized using central finite difference while hyperbolic terms shall be approximated by nonoscillatory upwind schemes. For clarity we shall examine difference schemes for these terms individually.

Mean Curvature Motion

Consider

$$\begin{cases} \dfrac{\partial v}{\partial t} = |\nabla v| \operatorname{div}\left(\dfrac{\nabla v}{|\nabla v|}\right) \\ v(0,x,y) = v_0(x,y) \end{cases} \tag{2.119}$$

Eq. (2.119) is a parabolic equation with diffusive effects (like the heat equation). Hence, instead of upwind schemes, we should use central differences as

$$u_{i,j}^{n+1} = u_{i,j}^{n} + \Delta t \sqrt{\left(\delta_x u_{i,j}^{n}\right)^2 + \left(\delta_y u_{i,j}^{n}\right)^2}\, K_{i,j}^{n}$$

where $K_{i,j}^{n}$ is the central finite-difference approximation of the curvature

$$K = \operatorname{div}\left(\frac{\nabla v}{|\nabla v|}\right) = \frac{v_{xx} v_y^2 + v_{yy} v_x^2 - 2 v_x v_y v_{xy}}{\left(v_x^2 + v_y^2\right)^{3/2}}$$

Difficulties may occur when ∇v is undefined, or $|\nabla v| = 0$ or $|\nabla v| = +\infty$. To deal with this kind of problems, we need to develop techniques that can prevent the gradient from vanishing (or blowing up). This can be done by reinitializing the function *v* from time to time to a signed distance function. More specifically we perform (2.119) in a discrete manner until some step *n*, then we solve the following auxiliary PDE

$$\begin{cases} \dfrac{\partial \varphi}{\partial t} + \operatorname{sign}(\varphi)\left(|\nabla \varphi| - 1\right) = 0 \\ \varphi(0,x,y) = v(n\Delta t, x, y) \end{cases} \tag{2.220}$$

The solution φ^{∞} of (2.220) (as *t* tends to infinity) is a signed distance function whose zero-level set is the same as that of the function $v(n\Delta t, x, y)$. Then we run Eq. (2.119) again with the initial condition $v(0,x,y) = \varphi^{\infty}(x,y)$. Practically this reinitialization needs to be carried out every *n* = 20 iterations of the curve evolution, and then about 5 to 10 iterations of Eq. (2.220) are usually performed. Of course the actual numbers of iterations are dependent on the problem at hand as well as the time step size.

Constant Speed Evolution

We now consider a constant-speed evolution equation

$$\begin{cases} \dfrac{\partial v}{\partial t} = c|\nabla v| \\ v(0,x,y) = v_0(x,y) \end{cases} \tag{2.221}$$

where *c* is a constant. The equation describes a motion in the direction normal to the front. For *c* = 1, it is also referred to as *grass fire*, since it stimulates a grass fire wave-front propagation. Eq. (2.221) can be approximated by a nonoscillatory upwind scheme

$$u_{i,j}^{n+1} = u_{i,j}^{n} + \Delta t \nabla^{+} u_{i,j}^{n}$$

where

$$\nabla^+ u_{i,j}^n = \left[\max\left(\delta_x^- u_{i,j}^n, 0\right)^2 + \min\left(\delta_x^+ u_{i,j}^n, 0\right)^2 + \max\left(\delta_y^- u_{i,j}^n, 0\right)^2 + \min\left(\delta_y^+ u_{i,j}^n, 0\right)^2 \right]^{1/2}$$

Pure Advection Equation

The equation is given by

$$\begin{cases} \dfrac{\partial v}{\partial t} = A(x,y)\cdot\nabla v \\ v(0,x,y) = v_0(x,y) \end{cases} \tag{2.222}$$

where $A(x, y) = (A_1(x, y), A_2(x, y))$. Here we use a simple upwind scheme which checks the sign of each component of A and constructs a one-side upwind difference in the appropriate direction:

$$\begin{aligned} u_{i,j}^{n+1} = u_{i,j}^n + \Delta t \Big[& \max\left((A_1)_{i,j}^n, 0\right)\delta_x^- u_{i,j}^n + \min\left((A_1)_{i,j}^n, 0\right)\delta_x^+ u_{i,j}^n \\ & + \max\left((A_2)_{i,j}^n, 0\right)\delta_y^- u_{i,j}^n + \min\left((A_2)_{i,j}^n, 0\right)\delta_y^+ u_{i,j}^n \Big] \end{aligned}$$

Image Segmentation by Geodesic Active Contour Model

We now consider the model in (2.118), which can be seen as the sum of the previous discretization. In this way, we obtain

$$\begin{aligned} u_{i,j}^{n+1} = u_{i,j}^n + \Delta t \Big[& g_{i,j} K_{i,j}^n \left[(\delta_x u_{i,j}^n)^2 + (\delta_y u_{i,j}^n)^2 \right]^{1/2} \\ & + \alpha \left[\max\left(g_{i,j}, 0\right)\nabla^+ + \min\left(g_{i,j}, 0\right)\nabla^- \right] u_{i,j}^n \\ & + \max\left((g_x)_{i,j}^n, 0\right)\delta_x^- u_{i,j}^n + \min\left((g_x)_{i,j}^n, 0\right)\delta_x^+ u_{i,j}^n \\ & + \max\left((g_y)_{i,j}^n, 0\right)\delta_y^- u_{i,j}^n + \min\left((g_y)_{i,j}^n, 0\right)\delta_y^+ u_{i,j}^n \Big] \end{aligned}$$

where $\nabla^- u_{i,j}^n$ is obtained from the expression of $\nabla^+ u_{i,j}^n$ by switching the plus and minus signs.

Problems

2.1 Evaluate the approximation error of function $f(x) = \sin(x)$ by a 1st-order polynomial $p(x) = 2x/\pi$ over interval $[0, \pi/2]$ in the function space $C(0, \pi/2)$.

(i) Compute $e_{0,\infty} = \| f(x) - p(x) \|_{0,\infty}$

(ii) Compute $e_{1,\infty} = \| f(x) - p(x) \|_{1,\infty}$

(iii) Find a 1st-order polynomial $p^*(x)$ with minimum approximation error $e_{0,\infty} = \| f(x) - p^*(x) \|_{0,\infty}$ among all 1st-order polynomials.

2.2 Prove Lemma 2.5.

2.3 Compute the total variation of the following functions:

(i) $u_1(x) = 1 + \cos(x)$ on $[0, \pi]$

(ii) $u_2(x) = 1 + \cos(x) + 0.01\sin(1000x)$ on $[0, \pi]$

(iii) Comment on the computation results obtained in (i) and (ii).

(iv) $u_3(x, y) = 0.1x^4 - x^2 + 0.75y^2 + 0.12\cos(xy) - x$ on $\{(x, y) : -\pi \le x, y \le \pi\}$

(v) $u_4(x, y) = u_3(x, y) + 0.1\sin[10000(x + y)]$ on $\{(x, y) : -\pi \le x, y \le \pi\}$

(vi) Comment on the computation results obtained in (iv) and (v).

2.4 Derive the Euler-Lagrange equation of the variational minimization problem (2.35) where

$$f = \sqrt{u_x^2 + u_y^2}$$
$$g_1 = u - u_0$$
$$g_2 = (u - u_0)^2 - \sigma^2 / |\Omega|$$

where $|\Omega|$ denotes the area of region Ω.

2.5 View $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$ as a linear operator (transformation) from space R^3 into itself. Compute the norm of A defined by $\|A\|_2 = \sup_{x \neq 0, x \in R^3} \frac{\|Ax\|_2}{\|x\|_2}$, where $\|x\|_2$ is the standard Euclidean norm $\|x\|_2 = \left(x_1^2 + x_2^2 + x_3^2\right)^{1/2}$.

2.6 Let $F(x)$ be a convex function of n real variables. Show that, for any $\lambda_i \ge 0$ with $\sum_{i=1}^{n} \lambda_i = 1$,

$$F\left(\sum_{i=1}^{n} \lambda_i x_i\right) \le \sum_{i=1}^{n} \lambda_i F(x_i).$$

2.7 Prove properties (P1) – (P7) in Sec. 2.3.2.

2.8 Prove properties (P8) in Sec. 2.3.2.

2.9 Prove properties (P10) in Sec. 2.3.2.

2.10 Show that L^p for each p in $1 \le p \le +\infty$ is a Banach space.

2.11 Consider the circle defined by $(x_1 - 2)^2 + x_2^2 = 4$.

(i) Parameterize the circle and compute its curvature by using (2.51).

(ii) Construct a function $u(x_1, x_2)$ so that the circle is an isolevel of $u(x_1, x_2)$. Then compute its curvature using (2.54) or (2.55).

2.12 Examine the difference scheme in (2.109) for the initial-and-boundary value problem of the heat equation (2.108) where the initial value is the test image circles256.

(i) Use $\mu = 5,\ h = 1,$ and $\Delta t = 0.05$. Show the diffusion results for n = 50, 100, 150, and 200.

(ii) Use $\mu = 5,\ h = 1,$ and $\Delta t = 0.051$. Show the diffusion results for n = 50, 100, 150, and 200.

(iii) Carry out an analysis on the computation results obtained in (i) and (ii).

Chapter 3 IMAGE RESTORATION

3.1 Energy-Based Methods

3.1.1 *Introduction*

Degradation often occurs during image acquisition, formation, transformation, and recording process, mainly in terms of noise contamination, defocusing, and motion blur. We shall call the processing methodology aimed to reduce or eliminate the effects of such degradation *image restoration*.

A common image model for dealing with image restoration problems is given by

$$u_0 = H\,u + w \tag{3.1}$$

where u: $\Omega \subset R^2 \to R$ is the original image describing a real scene, u_0 is the observed image of the same scene, i.e. a degraded version of u, w denotes white additive Gaussian noise with zero mean and variance σ^2, and R is a linear operator representing the blur. Typically H is modeled as a convolution operator. Given observed data u_0, the restoration problem is to reconstruct (or estimate) image u based on model (3.1).

In more general terms the restoration problem is an inverse problem or a deconvolution problem. In accordance with the maximum likelihood principle, an approximation of u can be identified by solving the least-squares problem

$$\inf_u \iint_\Omega \left(Hu - u_0\right)^2 dx\,dy \tag{3.2}$$

If the minimum of (3.2) exists, then it must satisfy the following equation

$$H^*H\,u - H^*u_0 = 0 \tag{3.3}$$

where H^* denotes the adjoint of operator H. It turns out that solving the problem in (3.3) is ill posed, because H^*H is not always invertible, and even if H^*H happens to be invertible, it may possess small eigenvalues that may cause numerical instability of the solution. To address this ill-posedness problem it is always necessary to *regularize* the problem.

A classical way to regularize the ill-posed minimization problem (3.2) was proposed in 1960's by Tikhonov and Arsenin who considered the minimization of the modified functional

$$F(u) = \iint_\Omega \left|\nabla u\right|^2 dx\,dy + \lambda \iint_\Omega \left(Hu - u_0\right)^2 dx\,dy \tag{3.4}$$

where the second term measures the *fidelity* to the data, the first term is a *regularization* (*smoothing*) term to control the gradient of u (hence the noise is reduced), and λ is a positive weighting constant.

For the functional to be well defined, function u itself as well as its first-order derivatives must be in L^2,

hence the function space associated with (3.4) is $W^{1,2}(\Omega)$ defined by

$$W^{1,2}(\Omega) = \left\{ u \in L^2(\Omega), \nabla u \in L^2(\Omega)^2 \right\}$$

Under suitable assumptions on blurring operator *H*, the problem $\inf_u \left\{ F(u), u \in W^{1,2}(\Omega) \right\}$ admits a unique solution which is characterized by the Euler-Lagrange equation

$$\Delta u - \lambda H^* \left(H u - u_0 \right) = 0 \tag{3.5a}$$

with the Neumann boundary condition

$$\left. \frac{\partial u}{\partial N} \right|_{\partial\Omega} = 0 \quad (N \text{ is the outward normal to } \partial\Omega) \tag{3.5b}$$

On comparing (3.5) with (3.3), the ill-posed equation (3.3) is now modified by adding a weighted Laplacian, thus it becomes a Neumann boundary-value problem of second-order elliptic PDE. A serious drawback of using (3.5) for image restoration is that the presence of the Laplacian simply means very strong *isotropic* smoothing and, as a result, it does not preserve edges. This oversmoothing can also explained through (3.4) where the L^p norm (with $p = 2$) of the gradient allows us to remove the noise but also penalizes too much the image's edges. One would therefore expect an improved solution by decreasing *p* from 2 to 1. Rudin, Osher, and Fatemi [8][9] are among the first to investigate the problem along this line. In the next section we shall study their methods.

3.1.2 *The Rudin-Osher-Fatemi Method for De-Noising*

The main difference between the methods proposed in [8][9] and many well known earlier attempts is that in [8][9] images are regarded as two-variable *functions of bounded variation* (BV) and the de-noising and de-blurring problems are resolved by minimizing the total variation of the image subject to some variational constraints deduced from the model used and the statistics of the noise component. It is this BV framework that allows analysis and processing of piecewise smooth functions with jumps as seen in virtually all natural and synthetic images.

To understand the Rudin-Osher-Fatemi (ROF) algorithms from a more technical perspective, we note that the essential modification in (3.4) made in the ROF algorithms is in the smoothing term where the L^2 norm of the gradient is changed to the L^1 norm. Recall that the L^p norm of the vector function $\nabla u = [u_x \; u_y]^T$ is given by

$$\left\| \nabla u \right\|_{L^p} = \left[\iint_\Omega \left| \nabla u \right|^p dx\, dy \right]^{1/p}, \quad \left| \nabla u \right| = \sqrt{u_x^2 + u_y^2}$$

In particular, when $p = 2$ and 1, we have

$$\left\| \nabla u \right\|_{L^2} = \left[\iint_\Omega \left| \nabla u \right|^2 dx\, dy \right]^{1/2} = \left[\iint_\Omega (u_x^2 + u_y^2)\, dx\, dy \right]^{1/2}$$

$$\left\| \nabla u \right\|_{L^1} = \iint_\Omega \left| \nabla u \right| dx\, dy = \iint_\Omega \sqrt{u_x^2 + u_y^2}\, dx\, dy$$

It follows that the smoothing term in (3.4) is (the square root of) the L^2 norm of the gradient, while the L^1 norm of the gradient is equal to the total variation (TV) of $u(x, y)$, see Eqs. (2.41), (2.42), and (2.48).

The modified functional is now given by

$$F(u) = \iint_\Omega |\nabla u| \, dx\, dy + \frac{\lambda}{2} \iint_\Omega \left(Hu - u_0 \right)^2 dx\, dy \tag{3.6}$$

For the important special case of *de-noising* where the blurring effect can be neglected (i.e. operator H is the identity), the model in (3.1) becomes

$$u_0(x, y) = u(x, y) + w(x, y) \tag{3.7}$$

and the modified functional (3.6) in this case is given by

$$F(u) = \iint_\Omega |\nabla u| \, dx\, dy + \frac{\lambda}{2} \iint_\Omega \left(u - u_0 \right)^2 dx\, dy \tag{3.8}$$

Problem Formulation for the De-Noising Problem

It follows from model (3.7) and noise statistics that the observed data u_0 and reconstructed image u must satisfy the conditions

$$\iint_\Omega u \, dx\, dy = \iint_\Omega u_0 \, dx\, dy \tag{3.9}$$

and

$$\iint_\Omega \left(u - u_0 \right)^2 dx\, dy = \sigma^2 \tag{3.10}$$

An immediate consequence of (3.10) is that the functional in (3.8) becomes

$$F(u) = \iint_\Omega |\nabla u| \, dx\, dy + \frac{\lambda}{2} \sigma^2$$

hence minimizing $F(u)$ under constraints (3.9) and (3.10) is the same as minimizing the TV of $u(x, y)$ subject to the same constraints. Namely the image de-noising can be formulated as

$$\underset{u}{\text{minimize}} \quad J_0(u) = \iint_\Omega \sqrt{u_x^2 + u_y^2} \, dx\, dy \tag{3.11a}$$

$$\text{subject to:} \quad \iint_\Omega u \, dx\, dy = \iint_\Omega u_0 \, dx\, dy \tag{3.11b}$$

$$\frac{1}{2} \iint_\Omega \left(u - u_0 \right)^2 dx\, dy = \frac{\sigma^2}{2} \tag{3.11c}$$

A Solution Procedure

By defining the *Lagragian* of the problem (3.11) as the functional

$$L(u,\lambda_1,\lambda_2)=\iint_\Omega\left\{\sqrt{u_x^2+u_y^2}+\lambda_1(u-u_0)+\frac{\lambda_2}{2}\left[(u-u_0)^2-\frac{\sigma^2}{|\Omega|}\right]\right\}dx\,dy \tag{3.12}$$

and applying the Euler-Lagrange equation (2.37) to functional (3.12), it can be concluded that the solution of problem (3.11) satisfies the second-order partial differential equation

$$\frac{\partial}{\partial x}\left(\frac{u_x}{\sqrt{u_x^2+u_y^2}}\right)+\frac{\partial}{\partial y}\left(\frac{u_y}{\sqrt{u_x^2+u_y^2}}\right)-\lambda_1-\lambda_2(u-u_0)=0 \tag{3.13a}$$

subject to boundary condition

$$\left.\frac{\partial u(x,y)}{\partial N}\right|_{\partial\Omega}=0 \tag{3.13b}$$

where N denotes the outward normal to the boundary of region Ω, which is a reasonable requirement for images defined on a rectangular region and undergone a symmetric extension along the boundary of Ω. Moreover, because of constraint (3.13b) the Lagrange multiplier λ_1 can be dropped from (3.13a) since a solution of

$$\frac{\partial}{\partial x}\left(\frac{u_x}{\sqrt{u_x^2+u_y^2}}\right)+\frac{\partial}{\partial y}\left(\frac{u_y}{\sqrt{u_x^2+u_y^2}}\right)-\lambda(u-u_0)=0 \tag{3.14a}$$

$$\left.\frac{\partial u(x,y)}{\partial N}\right|_{\partial\Omega}=0 \tag{3.14b}$$

automatically satisfies $\iint_\Omega(u-u_0)\,dx\,dy=0$, i.e., constraint (3.11b).

In [8], problem (3.14) is solved by imbedding it into a nonlinear *parabolic* equation with time t (or scale) as an evolution parameter. Numerically this is equivalent to the gradient descent method. In doing so, we need to solve

$$\frac{\partial u}{\partial t}=\frac{\partial}{\partial x}\left(\frac{u_x}{\sqrt{u_x^2+u_y^2}}\right)+\frac{\partial}{\partial y}\left(\frac{u_y}{\sqrt{u_x^2+u_y^2}}\right)-\lambda(t)\cdot(u-u_0)\quad\text{for } t>0,\ (x,y)\in\Omega \tag{3.15a}$$

$$u(x,y,0)\text{ given} \tag{3.15b}$$

$$\left.\frac{\partial u(x,y)}{\partial N}\right|_{\partial\Omega}=0 \tag{3.15c}$$

$$\iint_\Omega[u(x,y,t)-u_0(x,y)]^2\,dx\,dy\equiv\sigma^2 \tag{3.15d}$$

where the initial value in (3.15b) is typically taken to be the original noisy image $u_0(x, y)$. As time t increases, the solution of the (3.15) converges to its "steady state", becoming time invariant thus the limiting solution of (3.15) satisfies (3.14), a de-noised version of the image.

Notice that the Lagrange multiplier λ in (3.15a) is *time-dependent* and must be evaluated in order to keep the time-evolution of the solution $u(x, y, t)$ going forward. By multiplying (3.15a) by $(u - u_0)$, using (3.15d), integrating by parts over region Ω and using (3.15c), and assuming that the steady state has been reached, we obtain a formula for updating the Lagrange multiplier $\lambda(t)$ as

$$\lambda(t) = -\frac{1}{\sigma^2}\iint_{\Omega}\left[\sqrt{u_x^2 + u_y^2} - \left(\frac{(u_0)_x \cdot u_x}{\sqrt{u_x^2 + u_y^2}} + \frac{(u_0)_y \cdot u_y}{\sqrt{u_x^2 + u_y^2}}\right)\right] dx\, dy \tag{3.16}$$

A Difference Scheme for Solving (3.15), (3.16)

Given below is a difference scheme proposed in [8] for computing a numerical solution of problem (3.11) by solving (3.15) on an evolving time grids. For the sake of notation simplicity, in the difference scheme a *square* region $R = [0, 1] \times [0, 1]$ has been assumed, however scheme can be applied to non-square images after some straightforward modifications.

$$x_i = ih,\ y_j = jh,\ \text{for } i, j = 0,1,...,K, \text{ with } Kh = 1$$

$$t_n = n\Delta t, \qquad \text{for } n = 0,1,.....$$

$$u_{i,j}^n = u(x_i, y_j, t_n)$$

$$u_{i,j}^0 = u_0(x_i, y_j)$$

For $i, j = 0,1,...,K-1$, do

$$u_{i,j}^{n+1} = u_{i,j}^n$$

$$+\Delta t\left[\delta_x^-\left(\frac{\delta_x^+ u_{i,j}^n}{\left[(\delta_x^+ u_{i,j}^n)^2 + (m(\delta_y^+ u_{i,j}^n, \delta_y^- u_{i,j}^n))^2\right]^{1/2}}\right)\right]$$

$$+\Delta t\left[\delta_y^-\left(\frac{\delta_y^+ u_{i,j}^n}{\left[(\delta_y^+ u_{i,j}^n)^2 + (m(\delta_x^+ u_{i,j}^n, \delta_x^- u_{i,j}^n))^2\right]^{1/2}}\right)\right]$$

$$-\Delta t \cdot \lambda^n\left(u_{i,j}^n - u_0(ih, jh)\right) \tag{3.17a}$$

with

$$u_{0,j}^n = u_{1,j}^n, u_{K,j}^n = u_{K-1,j}^n, u_{i,0}^n = u_{i,1}^n, \text{ and } u_{i,K}^n = u_{i,K-1}^n.$$

$$\delta_x^{\mp} u_{i,j} = \mp(u_{i\mp1,j} - u_{i,j})/h$$

$$\delta_y^{\mp} u_{i,j} = \mp(u_{i,j\mp1} - u_{i,j})/h$$

$$m(a,b) = \min\text{mod}(a,b) \equiv \left(\frac{\text{sgn}\, a + \text{sgn}\, b}{2}\right)\min(|a|,|b|)$$

where λ^n is computed based on (3.16) as

$$\lambda^n = -\frac{h^2}{\sigma^2}\left[\sum_i\sum_j \sqrt{\left(\delta_x^+ u_{i,j}^n\right)^2 + \left(\delta_y^+ u_{i,j}^n\right)^2} - \frac{(\delta_x^+ u_{i,j}^0)(\delta_x^+ u_{i,j}^n)}{\sqrt{\left(\delta_x^+ u_{i,j}^n\right)^2 + \left(\delta_y^+ u_{i,j}^n\right)^2}} - \frac{\left(\delta_y^+ u_{i,j}^0\right)\left(\delta_y^+ u_{i,j}^n\right)}{\sqrt{\left(\delta_x^+ u_{i,j}^n\right)^2 + \left(\delta_y^+ u_{i,j}^n\right)^2}}\right] \quad (3.17b)$$

A restriction on step size Δt is imposed for numerical stability of the scheme as

$$\frac{\Delta t}{h^2} \le c \quad \text{for some constant } c \quad (3.18)$$

Implementation Considerations

It has been demonstrated [R1][R2] that replacing the TV objective function $J_0(u)$ in (3.11a) with

$$J_\varepsilon(u) = \iint_\Omega \sqrt{u_x^2 + u_y^2 + \varepsilon}\, dx\, dy \quad (3.19)$$

where ε is a small positive scalar offers several computational advantages such as differentiability of $J_\varepsilon(u)$ at u with $\nabla u = 0$. For a bounded region Ω, we have

$$J_0(u) \le J_\varepsilon(u) \le J_0(u) + \sqrt{\varepsilon}\,|\,\Omega\,|$$

hence

$$\lim_{\varepsilon \to 0} J_\varepsilon(u) = J_0(u)$$

A difference scheme for the problem of minimizing $J_\varepsilon(u)$ subject to constraints (3.11b) and (3.11c) can be obtained by modifying the scheme in (3.17) where an ε is added to each of the square roots.

Our next remark is concerned with a notation convention that we shall adopt throughout our computer simulations. In space R^2, we shall place a *x*-*y* coordinate system shown in Fig. 3.1 for a domain Ω where an image of interest is defined as a function $u(x, y)$. After discretization the image is associated with a matrix of size *M* by *N*. It is our notation convention, the matrix is denoted by $\{u_{i,j}\}$ with first index *i* corresponding to variable *y* and the second index *j* corresponding to *x*. Evidently, such a convention is consistent with the coordinate system in Fig. 3.1, but in it the two discrete indices are placed in a reversed order relative to the order of continuous variables *x* and *y* is function $u(x, y)$.

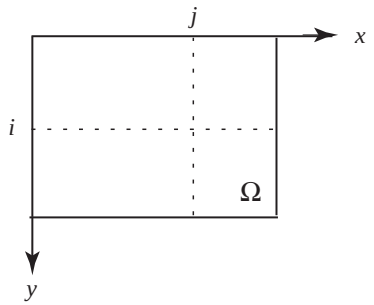


Figure 3.1: Notation convention

MATLAB Code and Experimental Results

Listed below are two MATLAB functions that implement the Rudin-Osher-Fatemi algorithm [8]. In the implementation, the objective function is replaced by $J_\varepsilon(u)$ in (3.19) with an $\varepsilon = 10^{-4}$. The main routine named tv_rof_true.m is given in the first m-file which calls for the second MATLAB function ex_sym2.m which extends an image of size M by N to (M+2) by (N+2) to satisfy the Neumann boundary condition (3.15c). We also remark that in the MATLAB code the spatial step size has been set to $h = 1$ (unlike in (3.17) where h was set to $1/K$ for an image of size K by K), which simplifies the evaluation of the many terms involving discrete gradient. At the same time, the term h^2 in (3.17b) should be replaced by $1/(MN)$ for an image of size M by N because h^2 in (3.17b) represents the area element $dx \cdot dy$.

1. Function tv_rof_true.m

```
% The code is based on the scheme proposed in L. I. Rudin, S. Osher, and E. Fatemi,
% "Nonlinear total variation based noise removal algorithms," Physica D, vol. 60, pp. 259-268, % 1992.
% Inputs:
% f   -- test image
% dt  -- stepsize in time
% st  -- state for Gaussian white noise generation
% sig -- sigma, standard deviation of the noise
% lam -- parameter lambda (i.e. the weight associated  with the fidelity term)
% K   -- number of iterations.
% Outputs:
% u       -- restored image
% fn      -- noise-corrupted image
% psnr0 -- PSNR before de-noising
% psnr1 -- PSNR after denoising
% cpt     -- CPU time in seconds
% Examples:
% [u,fn,lam,psnr0,psnr1,cpt] = tv_rof_true(reschart128,0.25,7,40,128);
% [u,fn,lam,psnr0,psnr1,cpt] = tv_rof_true(camera256,0.25,19,30,108);
% [u,fn,lam,psnr0,psnr1,cpt] = tv_rof_true(boat512,0.25,19,30,115); (18.5689 dB, 28.0671 dB)
% Written by W.-S. Lu, University of Victoria.
% Last modified: Jan. 4, 2008.
function [u,fn,lam,psnr0,psnr1] = tv_rof_true(f,dt,st,sig,K)
epsi = 1e-4;
[M,N] = size(f);
randn('state',st)
fn = f + sig*randn(M,N);
uw = fn;
uww = ex_sym2(uw);
ua = uww(2:(M+1),3:(N+2)); % u0_i+1,j
ub = uww(3:(M+2),2:(N+1)); % u0_i,j+1
uaw = ua - uw;
ubw = ub - uw;
uaw2 = uaw.^2;
ubw2 = ubw.^2;
uaw0 = uaw;
ubw0 = ubw;
lam(1) = 0;
lamw = 0;
k = 0;
t0 = cputime;
while k < K,
   uc = uww(2:(M+1),1:N);     % u_i-1,j
   ud = uww(3:(M+2),1:N);     % u_i-1,j+1
```

```
    ue = uww(1:M,2:(N+1));     % u_i,j-1
    uf = uww(1:M,3:(N+2));     % u_i+1,j-1
    ug = uww(1:M,1:N);         % u_i-1,j-1
    uwc = uw - uc;
    uwe = uw - ue;
    udc = ud - uc;
    ucg = uc - ug;
    ufe = uf - ue;
    ueg = ue - ug;
    uwc2 = uwc.^2;
    uwe2 = uwe.^2;
    amw = (0.5*(sign(ubw)+sign(uwe))).*min(abs(ubw),abs(uwe));
    amw2 = amw.^2;
    bmw = (0.5*(sign(udc)+sign(ucg))).*min(abs(udc),abs(ucg));
    bmw2 = bmw.^2;
    cmw = (0.5*(sign(uaw)+sign(uwc))).*min(abs(uaw),abs(uwc));
    cmw2 = cmw.^2;
    dmw = (0.5*(sign(ufe)+sign(ueg))).*min(abs(ufe),abs(ueg));
    dmw2 = dmw.^2;
    fu = fn - uw;
    T1 = uaw./(sqrt(uaw2 + amw2 + epsi));
    T2 = uwc./(sqrt(uwc2 + bmw2 + epsi));
    T3 = ubw./(sqrt(ubw2 + cmw2 + epsi));
    T4 = uwe./(sqrt(uwe2 + dmw2 + epsi));
    T5 = lamw*fu;
    uw_new = uw + dt*(T1 - T2 + T3 - T4 + T5);
    uw = uw_new;
    uww = ex_sym2(uw);
    ua = uww(2:(M+1),3:(N+2)); % u0_i+1,j
    ub = uww(3:(M+2),2:(N+1)); % u0_i,j+1
    uaw = ua - uw;
    ubw = ub - uw;
    uaw2 = uaw.^2;
    ubw2 = ubw.^2;
    srt = sqrt(uaw2 + ubw2 + epsi);
    P = (uaw0.*uaw + ubw0.*ubw)./srt;
    lamw = -sum(sum(srt-P))/((M*N)*sig^2);
    lam(k+2) = lamw;
    k = k + 1;
end
cpt = cputime – t0;
u = uw;
map = gray(256);
figure(1)
subplot(221)
imshow(f,map)
title('test image')
subplot(222)
imshow(fn,map)
title('noisy image')
subplot(223)
imshow(u,map)
title('image restored by true ROF')
subplot(224)
plot(lam)
axis([1 K+1 0 1.2*max(lam)])
axis square
```

```
title('Lagrange multiplier lambda(t)')
mse0 = (norm(fn-f,'fro'))^2/(M*N);
disp('PSNR before restoration:')
psnr0 = 10*log10(255^2/mse0)
mse1 = (norm(u-f,'fro'))^2/(M*N);
disp('PSNR after ROF restoration:')
psnr1 = 10*log10(255^2/mse1)
disp('CPU time in seconds')
cpt
```

2. Function ex_sym2.m

```
% To expand the input 2-D signal, uw, of size
% M x N to signal uwe of size (M+2) x (N+2) by
% symmetriical reflection so that the "main part"
% of the 1st column of uwe equals the 2nd column
% of uw, and the "main part" of the last column of
% uwe equals the (N-1)th column of uw, and the
% 1st and last rows of uwe are generated similarly.
% Written by W.-S. Lu, University of Victoria.
% Last modified: Jan. 3, 2008.
function uwe = ex_sym2(uw)
[M,N] = size(uw);
w = zeros(M+2,N+2);
w(2:(M+1),2:(N+1)) = uw;
w(1,2:(N+1)) = uw(2,:);
w((M+2),2:(N+1)) = uw((M-1),:);
w(2:(M+1),1) = uw(:,2);
w(2:(M+1),(N+2)) = uw(:,(N-1));
w(1,1) = uw(2,2);
w(1,(N+2)) = uw(2,(N-1));
w((M+2),1) = uw((M-1),2);
w((M+2),(N+2)) = uw((M-1),(N-1));
uwe = w;
```

Example 3.1

```
[u,fn,lam,psnr0,psnr1,cpt] = tv_rof_true(reschart128,0.25,7,40,128);
```

Input image: reschart128
Standard deviation: 40
randn state: 7
step size in time: 0.25
number of iterations: 128
PSNR† of the noisy image: 16.0555 dB
PSNR of the restored image: 23.6269 dB
Figure: 3.2
CPU time: 4.0156 s

† Recall the PSNR of a restored image $\hat{u}(x, y)$ of size $M \times N$ versus the perfect clean image $u(x, y)$ is defined by

$$\text{PSNR} = 10\log_{10}\left(\frac{255^2}{\text{MSE}}\right) \text{ (dB)}, \quad \text{where} \quad \text{MSE} = \frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N}[\hat{u}(i, j) - u(i, j)]^2$$

Note that the definition of PSNR differs from SNR which is defined as

$$\text{SNR} = 20\log_{10}\frac{\|u[i, j]\|_F}{\|\hat{u}[i, j] - u[i, j]\|_F} \text{ (dB)}, \quad \text{where} \quad \|u[i, j]\|_F = \sqrt{\sum_{i=1}^{M}\sum_{j=1}^{N}u(i, j)^2}$$

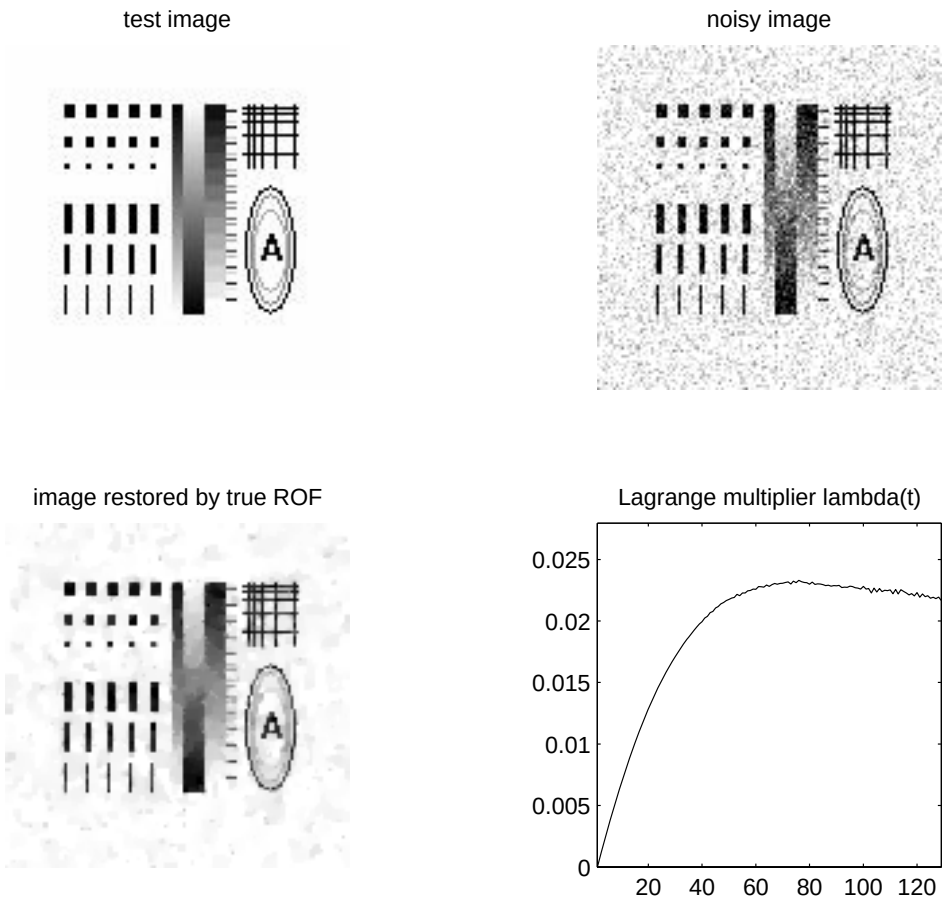


Figure 3.2: For Example 3.1

Example 3.2

[u,fn,lam,psnr0,psnr1,cpt] = tv_rof_true(camera256,0.25,19,30,108);
Input image: camera256
Standard deviation: 30
randn state: 19
step size in time: 0.25
number of iterations: 108
PSNR of the noisy image: 18.5888 dB
PSNR of the restored image: 26.9485 dB
Figure: 3.3
CPU time: 15.5469 s

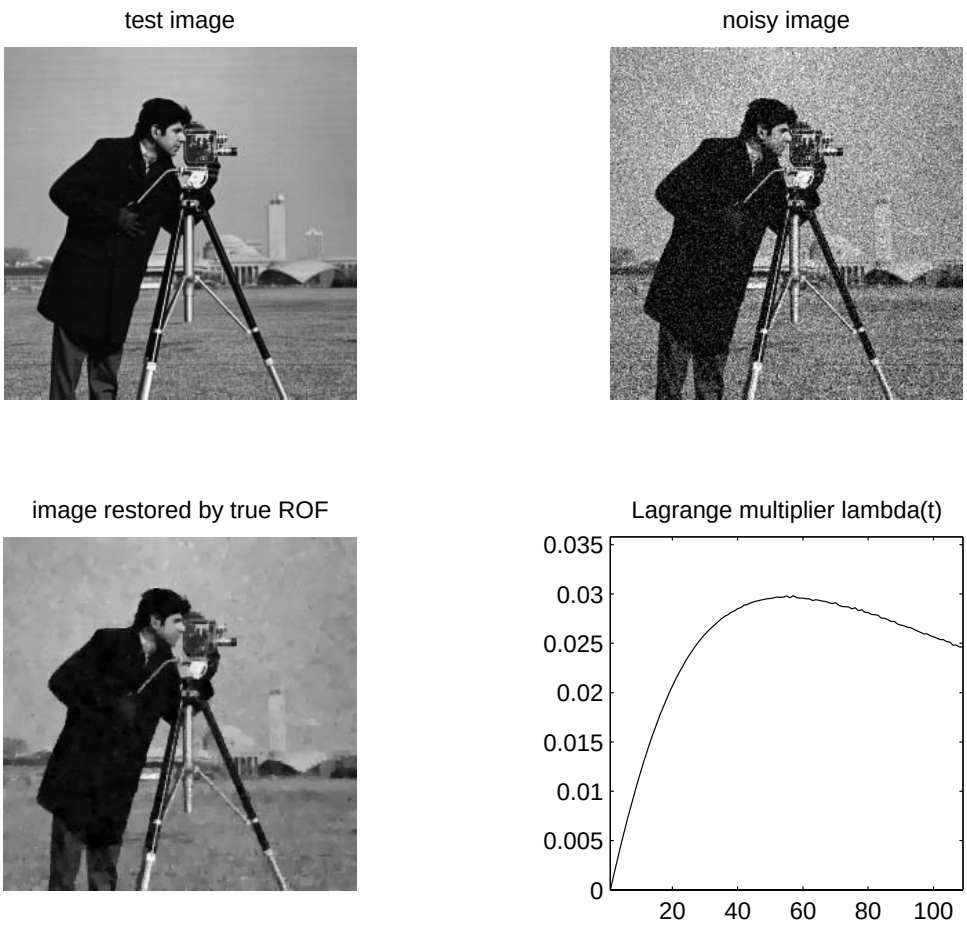


Figure 3.3: For Example 3.2

Example 3.3

```
[u,fn,lam,psnr0,psnr1,cpt] = tv_rof_true(boat512,0.25,19,30,115);
```

Input image: boat512
Standard deviation: 30
randn state: 19
step size in time: 0.25
number of iterations: 115
PSNR of the noisy image: 18.5689 dB
PSNR of the restored image: 28.0670 dB
Figure: 3.4
CPU time: 65.1094 s

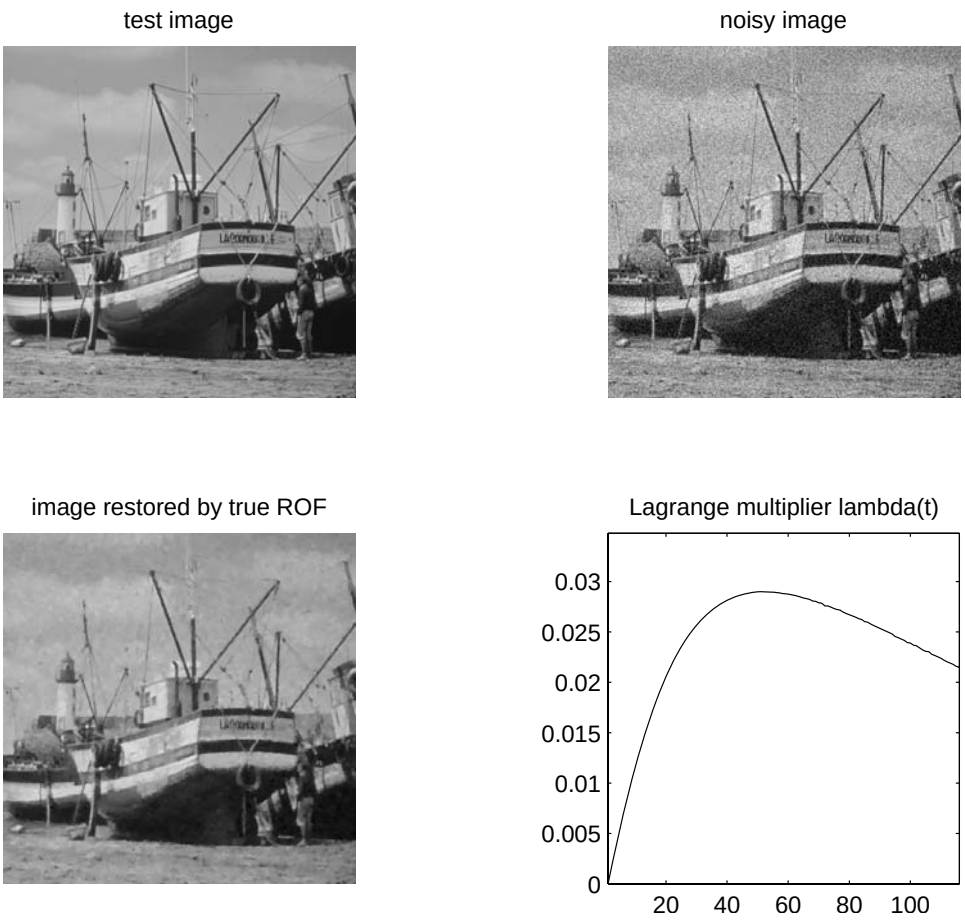


Figure 3.4: For Example 3.3.

3.1.3 *ROF De-Noising: A Simplified Problem Formulation*

A simplified problem formulation for the de-noising problem studied in Sec. 3.1.2 is to consider the unconstrained minimization of the functional

$$F_\lambda(u) = \iint_\Omega \sqrt{u_x^2 + u_y^2}\, dx\, dy + \frac{\lambda}{2} \iint_\Omega (u - u_0)^2\, dx\, dy \tag{3.20}$$

where the first and second terms on the right-hand side are regularization and fidelity terms, respectively, and $\lambda > 0$ is a weighting parameter ([R1]-[R4]). Here the idea is to minimize the TV so as to remove the noise while ensuring a kind of closeness of the restored image to the observed image u_0 in the L^2 sense.

The EL equation associated with (3.20) is given by

$$\frac{\partial}{\partial x}\left(\frac{u_x}{\sqrt{u_x^2 + u_y^2}}\right) + \frac{\partial}{\partial y}\left(\frac{u_y}{\sqrt{u_x^2 + u_y^2}}\right) - \lambda(u - u_0) = 0 \tag{3.21a}$$

$$\left.\frac{\partial u(x,y)}{\partial N}\right|_{\partial\Omega} = 0 \tag{3.21b}$$

Note that equation (3.21) appears to be identical to Eq. (3.14) although parameter λ in (3.14a) is the Lagrange multiplier associated with constrained (3.11c) while λ in (3.21a) is merely a positive weighting *constant*. For this reason, (3.21) is imbedded into the parabolic initial-boundary-value problem

$$\frac{\partial u}{\partial t} = \frac{\partial}{\partial x}\left(\frac{u_x}{\sqrt{u_x^2+u_y^2}}\right)+\frac{\partial}{\partial y}\left(\frac{u_y}{\sqrt{u_x^2+u_y^2}}\right)-\lambda(u-u_0) \quad \text{for } t>0,\ (x,y)\in\Omega \tag{3.22a}$$

$$u(x, y, 0) \text{ given} \tag{3.22b}$$

$$\left.\frac{\partial u(x,y)}{\partial N}\right|_{\partial\Omega} = 0 \tag{3.22c}$$

where λ is the same constant as in (3.20), whose value is usually a priori chosen. A reasonable difference scheme for (3.22) can be obtained from (3.17a) by fixing parameter λ as a time invariant constant:

$$\begin{aligned}
&x_i = ih,\ y_j = jh, \text{ for } i, j = 0,1,...,K, \text{ with } Kh = 1\\
&t_n = n\Delta t, \quad \text{for } n = 0,1,.....\\
&u_{i,j}^n = u(x_i, y_j, t_n)\\
&u_{i,j}^0 = u_0(x_i, y_j)\\
&\text{For } i, j = 0,1,...,K-1, \text{ do}\\
&u_{i,j}^{n+1} = u_{i,j}^n\\
&+\Delta t\left[\delta_x^-\left(\frac{\delta_x^+ u_{i,j}^n}{\left[(\delta_x^+ u_{i,j}^n)^2+(m(\delta_y^+ u_{i,j}^n, \delta_y^- u_{i,j}^n))^2\right]^{1/2}}\right)\right]\\
&+\Delta t\left[\delta_y^-\left(\frac{\delta_y^+ u_{i,j}^n}{\left[(\delta_y^+ u_{i,j}^n)^2+(m(\delta_x^+ u_{i,j}^n, \delta_x^- u_{i,j}^n))^2\right]^{1/2}}\right)\right]\\
&-\Delta t\cdot\lambda\left(u_{i,j}^n - u_0(ih, jh)\right)
\end{aligned} \tag{3.23}$$

with

$$u_{0,j}^n = u_{1,j}^n, u_{K,j}^n = u_{K-1,j}^n, u_{i,0}^n = u_{i,1}^n, \text{ and } u_{i,K}^n = u_{i,K-1}^n.$$

$$\delta_x^{\mp} u_{i,j} = \mp(u_{i\mp1,j} - u_{i,j})/h$$

$$\delta_y^{\mp} u_{i,j} = \mp(u_{i,j\mp1} - u_{i,j})/h$$

$$m(a,b) = \min\text{mod}(a,b) \equiv \left(\frac{\text{sgn}\, a + \text{sgn}\, b}{2}\right)\min(|a|,|b|)$$

As marked earlier, for implementation purposes the spatial size h can be set as $h = 1$, and a small positive scalar ε is usually added to each square root in (3.23) to avoid numerical difficulties.
Another difference scheme which simplifies (3.23) and also works well is to replace the main equation in (3.23) by

$$\begin{aligned} u_{i,j}^{n+1} &= u_{i,j}^{n} \\ &+\Delta t\left[\delta_x^-\left(\frac{\delta_x^+ u_{i,j}^n}{\left[(\delta_x^+ u_{i,j}^n)^2+(\delta_y^+ u_{i,j}^n)^2+\varepsilon\right]^{1/2}}\right)\right] \\ &+\Delta t\left[\delta_y^-\left(\frac{\delta_y^+ u_{i,j}^n}{\left[(\delta_x^+ u_{i,j}^n)^2+(\delta_y^+ u_{i,j}^n)^2+\varepsilon\right]^{1/2}}\right)\right] \\ &-\Delta t\cdot\lambda\left(u_{i,j}^n - u_0(ih, jh)\right) \end{aligned} \tag{3.24}$$

Experimental Results

MATLAB functions based on schemes (3.23) and (3.24) are applied to the same test images used earlier and the results obtained are given below.

Example 3.4
(a)
[u,fn,lam,psnr0,psnr1,cpt] = tv_rof(reschart128,0.25,7,40,0.02,123); (based on scheme (3.23))
Input image: reschart128
Standard deviation: 40
randn state: 7
step size in time: 0.25
λ : 0.02
number of iterations: 123
PSNR of the noisy image: 16.0555 dB
PSNR of the restored image: 23.6168 dB
CPU time: 3.1563 s
(b)
[u,fn,lam,psnr0,psnr1,cpt] = tv_l2(reschart128,0.25,7,40,0.006,119); (based on scheme (3.24))
Input image: reschart128
Standard deviation: 40
randn state: 7
step size in time: 0.25
λ : 0.006
number of iterations: 119
PSNR of the noisy image: 16.0555 dB
PSNR of the restored image: 22.5641 dB
CPU time: 2.8719 s

Example 3.5
(a)
[u,fn,lam,psnr0,psnr1,cpt] = tv_rof(camera256,0.25,19,30,0.01,86); (based on scheme 3.23))
Input image: camera256
Standard deviation: 30

randn state: 19
step size in time: 0.25
λ : 0.01
number of iterations: 86
PSNR of the noisy image: 18.5888 dB
PSNR of the restored image: 26.9234 dB
CPU time: 11.0469 s

(b)

[u,fn,lam,psnr0,psnr1,cpt] = tv_l2(camera256,0.25,19,30,0.0075,100); (based on scheme 3.24))
Input image: camera256
Standard deviation: 30
randn state: 19
step size in time: 0.25
λ : 0.0075
number of iterations: 100
PSNR of the noisy image: 18.5888 dB
PSNR of the restored image: 26.3663 dB
CPU time: 10.1094 s

Example 3.6

(a)

[u,fn,psnr0,psnr1,cpt] = tv_rof(boat512,0.25,19,30,0.006,90); (based on scheme 3.23))
Input image: boat512
Standard deviation: 30
randn state: 19
step size in time: 0.25
λ : 0.006
number of iterations: 90
PSNR of the noisy image: 18.5689 dB
PSNR of the restored image: 28.1022 dB
CPU time: 44.2344 s

(b)

[u,fn,psnr0,psnr1,cpt] = tv_l2(boat512,0.25,19,30,0.0005,100); (based on scheme 3.23))
Input image: boat512
Standard deviation: 30
randn state: 19
step size in time: 0.25
λ : 0.0005
number of iterations: 100
PSNR of the noisy image: 18.5689 dB
PSNR of the restored image: 28.0497 dB
CPU time: 40.0156 s

3.1.4 *The ROF Method for Image De-Blurring*

Most practical images are subject to degradations by noise corruption and blurriness, due primarily to imperfect performance in image acquisition, transmission, and so on. As Eq. (3.1) indicates, the degradation process may be modeled by a blurring operation H applied to the original (pure and clean) image $u(x,y)$ plus an additive noise $w(x, y)$, yielding observed data image $u_0(x, y)$. Specifically, the

blurring operation may be modeled as a convolution which involves a kernel $h(x, y)$. The specific form of $h(x, y)$ depends on the nature of the blurring operation. Thus (3.1) assumes a more specific form

$$u_0(x, y) = h(x, y) \otimes u(x, y) + w(x, y) \tag{3.25}$$

Alternatively, the model in (3.25) has an equivalent representation in the frequency-domain as

$$U_0(\omega_1, \omega_2) = H(\omega_1, \omega_2) \cdot U(\omega_1, \omega_2) + W(\omega_1, \omega_2) \tag{3.26}$$

where the terms in capital letters are the Fourier transforms of the corresponding terms in (3.25). Since convolution in the spatial domain corresponds to multiplication in the frequency domain, we note that the convolution operation "$\otimes$" in (3.25) becomes a conventional multiplication "$\cdot$" in (3.26).

3.1.4.1 *Models of Blurring Operation*

Several models of blurring operation have been used as they provide insight into image restoration problems. These models can produce blurring results that closely mimic real-world imperfect images obtained subject to physical or environmental constraints during image acquisition. Below we introduce two most common models of blurring operations.

A. Blurring Due to Uniform Linear Motion

This model can be used to simulate the blurry result caused by uniform linear motion between the image and the sensor during image acquisition. Suppose an image $u(x, y)$ undergoes certain uniform planar motion that is characterized by the velocity v_x along the *x* direction and velocity v_y along the *y* direction. In this case the blurring model in the frequency domain can be described as

$$H(\omega_1, \omega_2) = \frac{1}{\pi(\omega_1 v_x + \omega_2 v_y)} \sin\left[\pi(\omega_1 v_x + \omega_2 v_y)\right] e^{-j\pi(\omega_1 v_x + \omega_2 v_y)} \tag{3.27}$$

To generate an image blurred by uniform linear motion, we compute the 2-D Fourier transform of the image, multiply it by $H(\omega_1, \omega_2)$ in (3.27), and then take the inverse 2-D Fourier transform of the product.

Concerning discrete implementations of (3.25) and (3.26), the discrete version of (3.25) is the 2-D discrete convolution of the impulse response of the 2-D digital linear motion filter with a discretized version of image $u(x, y)$ plus a discrete version of noise $w(x, y)$. In MATLAB, the impulse response matrix *H* of a 2-D linear motion filter can be obtained using H = fspecial('motion', len, theta). Convolving H with an discrete image returns with an image that mimics the linear motion of a camera by len pixels, with an angle of theta degrees in a counter-clockwise direction. The convolution of H with a digital image A can be implemented using B = imfilter(A, H, 'option') where input option can be symmetric, replicate, or circular. For example, linear motion of 15 pixels along 45, 0, and 90 degrees for test image *lena256* can be obtained using

```
Hm45 = fspecial('motion',15,45);
A45 = imfilter(lena256,Hm45,'replicate');
Hm0 = fspecial('motion',15,0);
A0 = imfilter(lena256,Hm0,'replicate');
Hm90 = fspecial('motion',15,90);
```

```
A90 = imfilter(lena256,Hm90,'replicate');
A = [lena256 A45; A0 A90];
map = gray(256);
imshow(A, map)
```

And linear motion of 7 pixels along 45, 0, and 90 degrees for test image *reschart128* can be obtained using

```
Hm45 = fspecial('motion',7,45);
A45 = imfilter(reschart128,Hm45,'replicate');
Hm0 = fspecial('motion',7,0);
A0 = imfilter(reschart128,Hm0,'replicate');
Hm90 = fspecial('motion',7,90);
A90 = imfilter(reschart128,Hm90,'replicate');
A = [reschart128 A45; A0 A90];
map = gray(256);
imshow(A, map)
```

The results of these linear motion blurs for the two test images are depicted in Figs. 3.5 and 3.6, respectively.

Figure 3.5: Clockwise from upper-left: (a) Original *lena256*, (b) *lena256* blurred with 15-pixel linear motion along 45 degrees, (c) *lena256* blurred with 15-pixel linear motion along 90 degrees, (d) *lena256* blurred with 15-pixel linear motion along 0 degree.

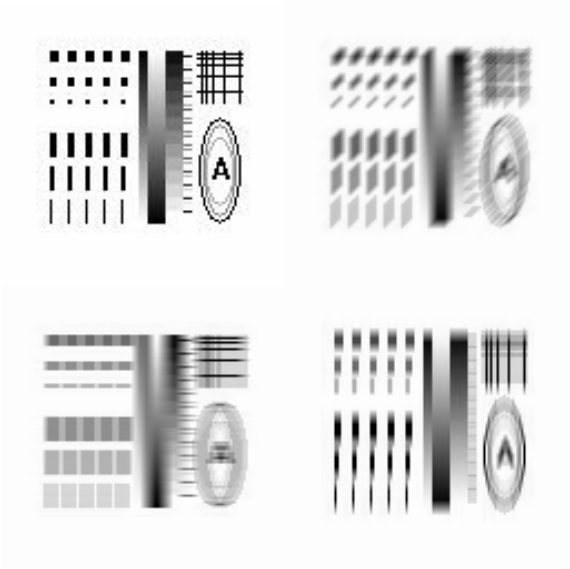


Figure 3.6: Clockwise from upper-left: (a) original *reschart128*, (b) *reschart128* blurred with 7-pixel linear motion along 45 degrees, (c) *reschart128* blurred with 7-pixel linear motion along 90 degrees, (d) *reschart128* blurred with 7-pixel linear motion along 0 degree.

B. Blurring Due to Gaussian Lowpass Filtering

The model in this case is characterized by a two-dimensional Gaussian kernel which can be described by its transfer function in the frequency-domain as

$$H(\omega_1, \omega_2) = e^{-(\omega_1^2 + \omega_2^2)/2\sigma^2} \tag{3.28}$$

where σ is a measure of the spread of the Gaussian surface. Like the discrete implementation of the linear motion blur, discrete Gaussian blur can be performed using MATLAB by command H = fspecial('gaussian',hsize,sigma) which returns a rotationally symmetric Gaussian lowpass filter of size hsize with standard deviation sigma. Parameter hsize can be a vector specifying the number of rows and columns in H or a scalar, in which case H is a square matrix. As examples, lowpass filtering of image *camera256* using a 7×7 Gaussian kernel with $\sigma = 1$, $\sigma = 3$, and $\sigma = 5$ can be performed using the code

```
H1 = fspecial('gaussian',7,1);
A1 = imfilter(camera256,H1,'replicate');
H3 = fspecial('gaussian',7,3);
A3 = imfilter(camera256,H3,'replicate');
H5 = fspecial('gaussian',7,5);
A5 = imfilter(camera256,H5,'replicate');
A = [lena256 A1; A5 A3];
map = gray(256);
imshow(A, map)
```

and lowpass filtering of image *reschart128* using a 7×7 Gaussian kernel with $\sigma = 1$, $\sigma = 2$, and $\sigma = 3$ can be performed using the code

```
H1 = fspecial('gaussian',7,1);
A1 = imfilter(reschart128,H1,'replicate');
H2 = fspecial('gaussian',7,2);
```

```
A2 = imfilter(reschart128,H2,'replicate');
H3 = fspecial('gaussian',7,3);
A3 = imfilter(reschart128,H3,'replicate');
A = [lena256 A1; A3 A2];
map = gray(256);
imshow(A, map)
```

The blurred images obtained are shown in Figs. 3.7 and 3.8, respectively.

Figure 3.7: Clockwise from upper-left: (a) original *camera256*, (b) *camera256* blurred by 7 × 7 Gaussian lowpass filter with $\sigma = 1$, (c) Gaussian blurred *camera256* with $\sigma = 3$, (d) Gaussian blurred *camera256* with $\sigma = 5$.

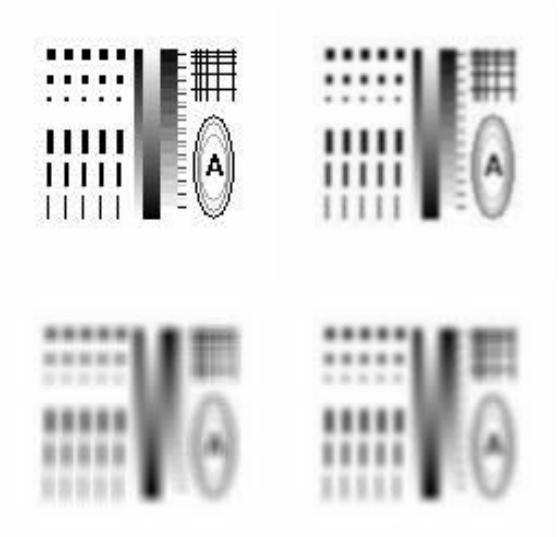


Figure 3.8: Clockwise from upper-left: (a) original *reschart128*, (b) *reschart128* blurred by 7 × 7 Gaussian lowpass filter with $\sigma = 1$, (c) Gaussian blurred *reschart128* with $\sigma = 2$, (d) Gaussian blurred *reschart128* with $\sigma = 3$.

3.1.4.2 *Problem Formulation*

The image de-blurring problem is investigated in a similar variational optimization framework. Following the notation employed in [9], the image model is given by (3.1), i.e.,

$$u_0(x, y) = (Hu)(x, y) + w(x, y) \quad \text{for } (x, y) \text{ in } \Omega \tag{3.29}$$

where H is typically a convolutional type integral operator for modeling several common blurring processes such as averaging, Gaussian low-pass, Laplacian of Gaussian (LoG), and motion blurs.

The variational optimization problem in this case can be formulated as

$$\underset{u}{\text{minimize}} \quad J_0(u) = \iint_\Omega \sqrt{u_x^2 + u_y^2}\, dx\, dy \tag{3.30a}$$

$$\text{subject to:} \quad \iint_\Omega u\, dx\, dy = \iint_\Omega u_0\, dx\, dy \tag{3.30b}$$

$$\iint_\Omega \left(Hu - u_0\right)^2 dx\, dy = \sigma^2 \tag{3.30c}$$

The Euler-Lagrange equation corresponding to problem (3.30) is given by

$$\frac{\partial}{\partial x}\left(\frac{u_x}{\sqrt{u_x^2 + u_y^2}}\right) + \frac{\partial}{\partial y}\left(\frac{u_y}{\sqrt{u_x^2 + u_y^2}}\right) - \lambda H^*(Hu - u_0) = 0 \tag{3.31a}$$

$$\left.\frac{\partial u(x, y)}{\partial N}\right|_{\partial\Omega} = 0 \tag{3.31b}$$

where H^* denotes the *adjoint operator* for operator H. In [9], it is proposed that (3.31) be solved by calculating the steady-state solution of the imbedding parabolic PDE

$$\frac{\partial u}{\partial t} = \frac{\partial}{\partial x}\left(\frac{u_x}{\sqrt{u_x^2 + u_y^2}}\right) + \frac{\partial}{\partial y}\left(\frac{u_y}{\sqrt{u_x^2 + u_y^2}}\right) - \lambda(t) \cdot H^*(Hu - u_0) \quad \text{for } t > 0,\ (x, y) \in \Omega \tag{3.32a}$$

$$\left.\frac{\partial u(x, y)}{\partial N}\right|_{\partial\Omega} = 0 \tag{3.32b}$$

where, using a technique similar to that for the de-noising case, the time-varying Lagrange multiplier $\lambda = \lambda(t)$ is found to be

$$\lambda(t) = \frac{\iint_\Omega \left[\frac{\partial}{\partial x}\left(\frac{u_x}{\sqrt{u_x^2 + u_y^2}}\right) + \frac{\partial}{\partial y}\left(\frac{u_y}{\sqrt{u_x^2 + u_y^2}}\right)\right] \cdot H^*\left(Hu - u_0\right) dx\, dy}{\iint_\Omega \left[H^*\left(Hu - u_0\right)\right]^2 dx dy} \tag{3.33}$$

3.1.4.3 *Implementation Issues*

We conclude this section with a remark on some implementation issues of the algorithm that are concerned with discrete implementation of Hu and of H^*v in (3.32a) and (3.33). In the continuous setting, $(Hu)(x,y)$ represents an image obtained by 2-D convolution of image $u(x,y)$ with a blurring kernel $h(x,y)$:

$$(Hu)(x,y) = h(x,y) \otimes u(x,y) = \iint h(x-x', y-y')\, u(x',y')\, dx'dy'$$

For discrete implementation, the image becomes a matrix $u = \{u_{i,j}\}$ and the blurring kernel becomes a (typically small-size impulse-response matrix H), the discrete version of the above is given by

$$Hu = H \odot \{u_{i,j}\}$$

where $\odot$ denote the 2-D discrete convolution.

Concerning with discrete implementation of H^*v in (3.32a) and (3.33), we recall the definition $\langle Hu, v\rangle = \langle u, H^*v\rangle$ where $\langle\cdot,\cdot\rangle$ denotes the inner product in L^2. Thus we can write

$$\begin{aligned}
&\iint \left(\iint h(x-x', y-y')\, u(x',y')\, dx'dy' \right) \cdot v(x,y)\, dx\, dy \\
&= \iint u(x',y') \cdot \left(\iint h(x-x', y-y')\, v(x,y) dx\, dy \right) dx'dy' \\
&= \iint u(x',y') \cdot \left(\iint h(-(x'-x), -(y'-y))\, v(x,y) dx\, dy \right) dx'dy' \\
&= \iint u(x',y') \cdot \left(\iint h^*(x'-x, y'-y)\, v(x,y) dx\, dy \right) dx'dy'
\end{aligned}$$

where $h^*(x,y) = h(-x,-y)$. It follows that the discrete impulse-response matrix of H^* can be obtained by flipping discrete impulse-response matrix H from left to right and then flipping the resulting matrix up-side-down, i.e., H_star = flipud(fliplr(H)).

3.1.5 *A General Analysis of the BV Framework*

Motivated by the successful use of the functional in (3.6), we now consider a more general energy functional

$$E(u) = \iint_\Omega \varphi\left(|\nabla u|\right) dx\, dy + \frac{\lambda}{2} \iint_\Omega \left(Hu - u_0\right)^2 dx\, dy \tag{3.34}$$

where $\varphi(\cdot)$ is a smooth function whose form and properties will be studied in what follows. Suppose (3.34) admits a minimum point $u(x, y)$, then it satisfies the Euler-Lagrange equation

$$\operatorname{div}\left(\frac{\varphi'(|\nabla u|)}{|\nabla u|} \nabla u \right) - \lambda H^* \left(Hu - u_0\right) = 0 \tag{3.35}$$

For the purpose of image restoration, it is desirable to have a $\varphi(|\nabla u|)$ with the local property that

(a) at locations where intensity variations are weak (small gradient), $\varphi(|\nabla u|)$ encourages isotropic smoothing (i.e. smoothing in the same way in all directions);

(b) in a neighborhood of an edge C, the gradient of the image has large magnitude. To preserve the edge while smoothing the image (to remove noise), the smoothing is allowed only along the edge but not across it. In other words, the diffusion process must be anisotropic.

Note that at a point on edge C the unit vector $N(x, y) = \nabla u/|\nabla u|$ is normal to the edge, and the unit-length tangent $T(x, y)$ to C is a vector that is normal to $N(x, y)$ with $|T(x, y)| = 1$. Let the Hessian of $u(x, y)$ be denote by

$$\nabla^2 u = \begin{bmatrix} u_{xx} & u_{xy} \\ u_{xy} & u_{yy} \end{bmatrix}$$

then the second-order derivatives of $u(x, y)$ in the T-direction and N-direction can be evaluated as

$$\begin{aligned} u_{TT} &= T^T \nabla^2 u(x,y) T = \frac{1}{|\nabla u|^2}\left(u_x^2 u_{yy} - 2u_x u_y u_{xy} + u_y^2 u_{xx}\right) \\ u_{NN} &= N^T \nabla^2 u(x,y) N = \frac{1}{|\nabla u|^2}\left(u_x^2 u_{xx} + 2u_x u_y u_{xy} + u_y^2 u_{yy}\right) \end{aligned} \tag{3.36}$$

If we write the second term on the left side of (3.35) explicitly as

$$\operatorname{div}\left(\frac{\varphi'(|\nabla u|)}{|\nabla u|}\nabla u\right) = \frac{\partial}{\partial x}\left(\frac{\varphi'(|\nabla u|)}{|\nabla u|}u_x\right) + \frac{\partial}{\partial y}\left(\frac{\varphi'(|\nabla u|)}{|\nabla u|}u_y\right) = \frac{\varphi'(|\nabla u|)}{|\nabla u|}\Delta u + \frac{\partial}{\partial x}\left(\frac{\varphi'(|\nabla u|)}{|\nabla u|}\right)\cdot u_x + \frac{\partial}{\partial y}\left(\frac{\varphi'(|\nabla u|)}{|\nabla u|}\right)\cdot u_y$$

then by using (3.36) and the fact that $\Delta u = u_{TT} + u_{NN}$ we can verify that

$$\operatorname{div}\left(\frac{\varphi'(|\nabla u|)}{|\nabla u|}\nabla u\right) = \frac{\varphi'(|\nabla u|)}{|\nabla u|}\, u_{TT} + \varphi''(|\nabla u|)\, u_{NN} \tag{3.37}$$

hence (3.35) can be expressed as

$$\left(\frac{\varphi'(|\nabla u|)}{|\nabla u|}\, u_{TT} + \varphi''(|\nabla u|)\, u_{NN}\right) - \lambda H^*\left(Hu - u_0\right) = 0 \tag{3.38}$$

Now, to meet requirement (a), we impose

$$\varphi'(0) = 0 \quad \text{and} \quad \lim_{s\to 0^+}\frac{\varphi'(s)}{s} = \lim_{s\to 0^+}\varphi''(s) = \varphi''(0) > 0 \tag{3.39}$$

so that at points where $|\nabla u|$ is small, (3.38) becomes

$$\varphi''(0)\,\Delta u - \lambda\, H^*\left(Hu - u_0\right) = 0 \tag{3.40}$$

which means a uniform smoothing (see (3.5a)).

To satisfy requirement (b), we further impose

$$\lim_{s\to+\infty}\varphi''(s)=0 \quad \text{and} \quad \lim_{s\to+\infty}\frac{\varphi'(s)}{s}=\beta>0 \tag{3.41}$$

so that at points with large $|\nabla u|$, the coefficient of u_{NN} is annihilated while the coefficient of u_{TT} does not vanish. Unfortunately, the two constraints in (3.41) are not compatible and a compromise has to be made. We may for instance impose that both $\varphi''(s)$ and $\varphi'(s)/s$ approach zero as s goes to infinity with *different rate*:

$$\lim_{s\to+\infty}\varphi''(s)=\lim_{s\to+\infty}\frac{\varphi'(s)}{s}=0 \ \text{ and } \ \lim_{s\to+\infty}\frac{\varphi''(s)}{\varphi'(s)/s}=0 \tag{3.42}$$

There are many functions satisfying conditions (3.39) and (3.42), an example of such functions is the so-called *hypersurface minimal function*

$$\varphi(s)=\sqrt{s^2+1} \tag{3.43}$$

Moreover, existence and uniqueness of the solution in $\mathrm{BV}(\Omega)$ for a relaxed version of the functional in (3.34) can be established with mathematical rigor under the following assumptions [1, Sec. 3.2.3]:

A1. $\varphi(s)$ is a strictly convex, non decreasing function from R^+ to R^+ with $\varphi(0)=0$.

A2. $\varphi(s)$ goes to infinity as s goes to infinity, with a linear grow in the sense that there exist constants $c>0$ and $b\geq 0$ such that $cs-b\leq\varphi(s)\leq cs+b$ for $s\geq 0$.

A3. H is a linear operator from $L^2(\Omega)$ to $L^2(\Omega)$ satisfying $H.1\neq 0$.

It can be readily verified that the slightly modified hypersurface minimal function

$$\varphi(s)=2\sqrt{s^2+1}-2 \tag{3.44}$$

satisfies assumptions A1 and A2.

3.1.6 *An Efficient Semi-Implicit Scheme based on Additive Operator Splitting (AOS)*

A semi-implicit scheme based on an AOS technique for nonlinear diffusion is proposed in [13] and has since gained increased popularity due to its high efficiency. The method is quite general and can be applied to a variety of variational PDE models. For illustration purposes, below the technique is illustrated by applying it to the denoising problem formulated as minimizing the function in (3.20), i.e.,

$$F_\lambda(u)=\iint_\Omega\sqrt{u_x^2+u_y^2}\,dx\,dy+\frac{\lambda}{2}\iint_\Omega(u-u_0)^2\,dx\,dy \tag{3.45}$$

whose EL equation is given by

$$\frac{\partial}{\partial x}\left(\frac{u_x}{\sqrt{u_x^2+u_y^2}}\right)+\frac{\partial}{\partial y}\left(\frac{u_y}{\sqrt{u_x^2+u_y^2}}\right)-\lambda(u-u_0)=0 \tag{3.46a}$$

$$\left.\frac{\partial u(x,y)}{\partial N}\right|_{\partial\Omega}=0 \tag{3.46b}$$

As usual Eq. (3.46) is imbedded into the parabolic initial-boundary-value problem

$$\frac{\partial u}{\partial t}=\frac{\partial}{\partial x}\left(\frac{u_x}{\sqrt{u_x^2+u_y^2}}\right)+\frac{\partial}{\partial y}\left(\frac{u_y}{\sqrt{u_x^2+u_y^2}}\right)-\lambda(u-u_0) \quad \text{for } t>0,\ (x,y)\in\Omega \tag{3.47a}$$

$$u(x, y, 0) \text{ given} \tag{3.47b}$$

$$\left.\frac{\partial u(x,y)}{\partial N}\right|_{\partial\Omega}=0 \tag{3.47c}$$

If we define function

$$g(|\nabla u|)=\frac{1}{|\nabla u|} \tag{3.48}$$

and differential operators

$$A_1(u)=\frac{\partial}{\partial x}\left(g(|\nabla u|)\frac{\partial}{\partial x}\right),\qquad A_2(u)=\frac{\partial}{\partial y}\left(g(|\nabla u|)\frac{\partial}{\partial y}\right) \tag{3.49}$$

then (3.47a) can be expressed as

$$\frac{\partial u}{\partial t}=\big(A_1(u)+A_2(u)\big)u-\lambda(u-u_0) \tag{3.50}$$

Note that operators $A_1(u)$ and $A_2(u)$ are "*symmetrical*" to each other and each operates along *one direction*. A typical *explicit* scheme for the PDE in (3.50) is given by

$$\frac{u_{i,j}^{n+1}-u_{i,j}^{n}}{\tau}=\big(A_1(u_{i,j}^{n})+A_2(u_{i,j}^{n})\big)u_{i,j}^{n}-\lambda(u_{i,j}^{n}-u_{i,j}^{0}) \tag{3.51}$$

where τ is a step-size in time. The computational complexity of scheme (3.51) is low. It can be shown [13] that the explicit scheme creates a discrete scale space provided that the time step size remains sufficiently small, more precisely

$$\tau<\frac{1}{\max\limits_{i}\sum\limits_{j\neq i}a_{i,j}(u^n)} \tag{3.52}$$

where $a_{i,j}$ will be defined shortly. Eq. (3.52) turns out to be a severe limitation on the time step-size, because of this the explicit scheme is not very efficient.

A Semi-Implicit Scheme

We now consider a *semi-implicit* scheme

$$\frac{u_{i,j}^{n+1}-u_{i,j}^{n}}{\tau}=\left(A_1(u_{i,j}^{n})+A_2(u_{i,j}^{n})\right)u_{i,j}^{n+1}-\lambda(u_{i,j}^{n}-u_{i,j}^{0})$$

which leads to

$$\left(I-\tau\sum_{l=1}^{2}A_l(u_{i,j}^{n})\right)u_{i,j}^{n+1}=u_{i,j}^{n}-\tau\lambda(u_{i,j}^{n}-u_{i,j}^{0}) \tag{3.53}$$

or equivalently,

$$\begin{aligned} u_{i,j}^{n+1}&=\left(I-\tau\sum_{l=1}^{2}A_l(u_{i,j}^{n})\right)^{-1}\eta\left(u_{i,j}^{n}\right)\\ \eta\left(u_{i,j}^{n}\right)&=u_{i,j}^{n}-\tau\lambda(u_{i,j}^{n}-u_{i,j}^{0}) \end{aligned} \tag{3.54}$$

An AOS Technique to Deal With Scheme (3.54)

Additive operator splitting (AOS) is a general technique that reduces a multivariable computation scheme to several single-variable schemes. For the *m*-variable case, AOS replaces $\left(I-\tau\sum_{l=1}^{m}A_l(u_{i,j}^{n})\right)^{-1}$ by $\frac{1}{m}\sum_{l=1}^{m}\left[I-m\tau A_l(u_{i,j}^{n})\right]^{-1}$. For image processing purposes, $m = 2$ and applying AOS to (3.54) is to replace the inverse matrix there by

$$\frac{1}{2}\sum_{l=1}^{2}\left[I-2\tau A_l(u_{i,j}^{n})\right]^{-1}$$

This leads to the scheme

$$u_{i,j}^{n+1}=\frac{1}{2}\sum_{l=1}^{2}\left[I-2\tau A_l(u_{i,j}^{n})\right]^{-1}\eta\left(u_{i,j}^{n}\right) \tag{3.55}$$

Unlike (3.54) where the operator inversion involves both *x* and *y* directions, using (3.55) one takes the advantage of performing two *one-dimensional* processing because each operator A_l acts only along a direction that is associated with a single variable. More specifically, we write (3.55) as

$$\begin{aligned} u_{i,j}^{n+1}&=r_{i,j}^{n}+s_{i,j}^{n}\\ r_{i,j}^{n}&=\frac{1}{2}\left[I-2\tau A_1(u_{i,j}^{n})\right]^{-1}\eta\left(u_{i,j}^{n}\right)\\ s_{i,j}^{n}&=\frac{1}{2}\left[I-2\tau A_2(u_{i,j}^{n})\right]^{-1}\eta\left(u_{i,j}^{n}\right) \end{aligned} \tag{3.56}$$

Now if we regard $u_{i,j}^n$, $r_{i,j}^n$, $s_{i,j}^n$, and $\eta(u_{i,j}^n)$ in (3.56) as matrices of size *M* by *N*, since A_1 is an operator acting along *x* direction (i.e., row by row), matrix $r_{i,j}^n$ can be generated row by row as follows:

For *k* from 1 to *M*, do:

(a) take the *k*th row of matrix $u_{i,j}^n$ to generate a matrix A_1 of size *N* by *N* and solve the linear system equation

$$\left[I - 2\tau A_1(u_{i,j}^n) \right] x_k = \frac{1}{2} d_k \tag{3.57}$$

for x_k, where d_k is the transpose of the *k*th row of matrix $\eta(u_{i,j}^n)$.

(b) the *k*th row of matrix $r_{i,j}^n$ is obtained as the transpose of x_k.

Similarly, since A_2 is an operator acting along *y* direction (i.e., column by column), matrix $s_{i,j}^n$ can be generated column by column as follows:

For *k* from 1 to *N*, do:

(a) take the *k*th column of matrix $u_{i,j}^n$ to generate a matrix A_2 of size *M* by *M* and solve the linear system equation

$$\left[I - 2\tau A_2(u_{i,j}^n) \right] y_k = \frac{1}{2} e_k \tag{3.58}$$

for y_k, where e_k is the *k*th column of matrix $\eta(u_{i,j}^n)$.

(b) the *k*th column of matrix $s_{i,j}^n$ is obtained as y_k.

A Matrix Representation of 1-D Operator A_1

For notation simplicity, here we consider operator *A* given by

$$A(u) = \frac{\partial}{\partial x}\left(g(|\nabla u|) \frac{\partial}{\partial x} \right) \tag{3.59}$$

where $u = u(x, y)$ and the operator acts on *u* itself(see (3.50)). Since the operator is performed along the *x* direction, corresponding to the semi-implicit scheme (3.51) we consider $A\left(u_{i,j}^n\right) u_{i,j}^{n+1}$ with *j* (i.e. variable *y*) fixed. We see that operator *A* is determined by a 1-D data set – the *j*th column of matrix $u_{i,j}^n$ -- which will be denoted by u_i^n, and that the operator acts on a 1-D data set – the *j*th column of matrix $u_{i,j}^{n+1}$ -- which will be denoted by u_i^{n+1}. The difference scheme used for operator *A* in (3.59) as it applies to u_i^{n+1} is given by

$$
\begin{aligned}
&\frac{\partial}{\partial x}\left(g(|\nabla u_i^n|)\frac{\partial u_i^{n+1}}{\partial x} \right) \\
&\approx \frac{1}{2}\left[\delta_x^- \left(g(|\nabla u_i^n|)\delta_x^+ u_i^{n+1} \right) + \delta_x^+ \left(g(|\nabla u_i^n|)\delta_x^- u_i^{n+1} \right) \right] \\
&= \frac{1}{2h^2}\left[g(|\nabla u_i^n|)\left(u_{i+1}^{n+1} - u_i^{n+1} \right) - g(|\nabla u_{i-1}^n|)\left(u_i^{n+1} - u_{i-1}^{n+1} \right) \right. \\
&\left. + g(|\nabla u_{i+1}^n|)\left(u_{i+1}^{n+1} - u_i^{n+1} \right) - g(|\nabla u_i^n|)\left(u_i^{n+1} - u_{i-1}^{n+1} \right) \right] \\
&= \frac{1}{2h^2} \sum_{k \in S(i)} \left(g(|\nabla u_k^n|) + g(|\nabla u_i^n|) \right)\left(u_k^{n+1} - u_i^{n+1} \right)
\end{aligned}
\tag{3.60}
$$

where $S(i)$ is the set of two neighbor indices of i (a boundary index has only one neighbor index). Term gradient $g\left(|\nabla u_i^n|\right)$ is approximated using central difference as

$$
g_i^n \equiv g\left(|\nabla u_i^n|\right) \approx g\left(\sqrt{\left(\frac{u_{i+1,j}^n - u_{i-1,j}^n}{2h} \right)^2 + \left(\frac{u_{i,j+1}^n - u_{i,j-1}^n}{2h} \right)^2} \right) \tag{3.61}
$$

where the evaluation for boundary indices can be carried out by symmetrical expanding of $u_{i,j}^n$. Eqs. (3.60) and (61) now give

$$
\frac{\partial}{\partial x}\left(g(|\nabla u_i^n|)\frac{\partial u_i^{n+1}}{\partial x} \right) \approx \sum_{k \in S(i)} \frac{g_k^n + g_i^n}{2h^2}\left(u_k^{n+1} - u_i^{n+1} \right) \tag{3.62}
$$

Based on (3.62), $A\left(u_i^n\right)u_i^{n+1}$ for $i = 1, 2, \ldots, N$ can be written in a matrix-vector form as Au^{n+1} where $A = \{a_{i,k}\}$ is an N by N *tridiagonal matrix* with

$$
a_{i,k} = \begin{cases} \dfrac{g_i^n + g_k^n}{2h^2} & \text{if } k \in S(i) \\ -\displaystyle\sum_{l \in S(i)} \frac{g_i^n + g_l^n}{2h^2} & \text{if } k = i \\ 0 & \text{elsewhere} \end{cases} \tag{3.63}
$$

The Thomas Algorithm for Solving Tridiagonal Linear Systems

The Thomas algorithm for solving a tridiagonal linear system $Bu = d$ with

$$B = \begin{bmatrix} \alpha_1 & \beta_1 & 0 & \cdots & 0 \\ \gamma_1 & \alpha_2 & \beta_2 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \gamma_{N-2} & \alpha_{N-1} & \beta_{N-1} \\ 0 & 0 & 0 & \gamma_{N-1} & \alpha_N \end{bmatrix}$$

consists of three simple steps [13]:

Step1: (*LR Decomposition*) Perform an LR decomposition of B, i.e., $B = LR$ with L a lower bidiagonal

$$L = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ l_1 & 1 & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & l_{N-1} & 1 \end{bmatrix}$$

and R an upper bidiagonal

$$R = \begin{bmatrix} m_1 & r_1 & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & m_{N-1} & r_{N-1} \\ 0 & \cdots & 0 & m_N \end{bmatrix}$$

where $r_i = \beta_i$ for $i = 1, 2, \ldots, N-1$, and m_i and l_i can be computed by simple recursions as

$$\begin{aligned} & m_1 = \alpha_1 \\ & \text{for } i = 1 : N-1, \\ & \quad l_i = \gamma_i / m_i \\ & \quad m_{i+1} = \alpha_{i+1} - l_i \beta_i \\ & \text{end} \end{aligned}$$

Step 2: (*Forward Substitution*) To solve $LRu = d$, we first solve $Ly = d$ for y in this step (and then solve $Ru = y$ for u in the next step). Solving $Ly = d$ is done using the following recursion:

$$\begin{aligned} & y_1 = d_1 \\ & \text{for } \ i = 2 : N, \\ & \quad y_i = d_i - l_{i-1} y_{i-1} \\ & \text{end} \end{aligned}$$

Step 3: (*Backward Substitution*) Finally the following recursion is used to solve $Ru = y$ for u:

$$\begin{aligned} & u_N = y_N / m_N \\ & \text{for } \ i = (N-1) : -1 : 1, \\ & \quad u_i = (y_i - \beta_i u_{i+1}) / m_i \\ & \text{end} \end{aligned}$$

Remark: Obviously, the algorithm works only if matrices *L* and *R* are nonsingular. Matrix *L* is always nonsingular, and *R* is nonsingular if and only if all m_i are nonzero that holds true when matrix *B* is *strictly diagonally dominant* meaning that

$$|b_{i,i}| > \sum_{j \neq i} |b_{i,j}| \tag{3.64}$$

We'll see that for the tridiagonal systems involved in the semi-implicit scheme, when the AOS technique is applied the condition in (3.64) is always satisfied.

The Weickert-Romeny-Viergever (WRV) Algorithm

In summary, in the WRV algorithm the semi-implicit scheme for problem (3.47) is given by

$$u_{i,j}^{n+1} = \frac{1}{2}\left[I - 2\tau A_1(u_{i,j}^n)\right]^{-1} \eta\left(u_{i,j}^n\right) + \frac{1}{2}\left[I - 2\tau A_2(u_{i,j}^n)\right]^{-1} \eta\left(u_{i,j}^n\right) \tag{3.65a}$$

$$\eta\left(u_{i,j}^n\right) = u_{i,j}^n - \tau\,\lambda\,(u_{i,j}^n - u_{i,j}^0) \tag{3.65b}$$

Based on (3.63) it can be readily verified that the matrix involved in each term on the right side of (3.65a) is a strictly diagonally dominant tridiagonal matrix regardless of time step-size τ. Consequently, each term can be obtained by solving tridiagonal systems *line by line* using the Thomas algorithm.

An observation made in our computer simulations is that it may be advantageous to use a *time-varying* step-size τ to maximize denoising performance with a limited number of iterations. The parameter τ in (3.65) in this case is modified to $\tau^{(n)}$ and the modified scheme is given by

$$u_{i,j}^{n+1} = \frac{1}{2}\left[I - 2\tau^{(n)} A_1(u_{i,j}^n)\right]^{-1} \eta\left(u_{i,j}^n\right) + \frac{1}{2}\left[I - 2\tau^{(n)} A_2(u_{i,j}^n)\right]^{-1} \eta\left(u_{i,j}^n\right) \tag{3.66a}$$

$$\eta\left(u_{i,j}^n\right) = u_{i,j}^n - \tau^{(n)}\,\lambda\,(u_{i,j}^n - u_{i,j}^0) \tag{3.66b}$$

MATLAB Code and Experimental Results

The code consists of four MATLAB functions, namely tv_l2_aos.m, aos_tv2.m, sol_tri.m, and ex_sym2.m. The main routine is tv_l2_aos.m which implements scheme (3.66) and calls for aos_tv2.m to carry out an AOS-based iteration. Function aos_tv2.mrequires sol_tri.m to solve tridiagonal systems using the Thomas algorithm.

1. Function tv_l2_aos.m

```
% The code is based on the implementation scheme of the
% TV-plus-L2 (ROF) algorithm using the additive operator
% splitting (AOS) technique for semi-implicit schemes which
% is described in J. Weickert, B. M. ter Romeny, and M. A.
% Viergever, "Efficient and reliable schemes for nonlinear
% diffusion filtering," IEEE Trans. Iamge Processing, vol. 7,
% no. 3, pp. 398-410, March 1998.
% Inputs:
```

```
% f      -- test image
% tau  -- a set of K time stepsizes to be used in K iterations.
% st    -- state for Gaussian white noise generation
% sig  -- sigma, standard deviation of the noise
% lam -- parameter lambda (i.e. the weight associated
%          with the fidelity term)
% K    -- number of iterations.
% Outputs:
% u       -- restored image
% fn      -- noise-corrupted image
% psnr0 -- PSNR before de-noising
% psnr1 -- PSNR after denoising
% cpt     -- CPU time in seconds
% Examples:
% [u,fn,psnr0,psnr1,cpt] = tv_l2_aos(reschart128,[51 51 2 1.75 1.4],7,40,0.06,2); (21.6231dB)
% [u,fn,psnr0,psnr1,cpt] = tv_l2_aos(camera256,[8 7 5.5 4.5 3],19,30,0.04,5); (26.3542dB)
% [u,fn,psnr0,psnr1,cpt] = tv_l2_aos(boat512,[21 11 6 4 2.5],19,30,0.04,5); (27.9045dB)
% Written by W.-S. Lu, University of Victoria.
% Last modified: Jan. 10, 2008.

function [u,fn,psnr0,psnr1,cpt] = tv_l2_aos(f,tau,st,sig,lam,K)
h = 1;
h2 = 2*h;
epsi = 1e-4;
[M,N] = size(f);
randn('state',st)
fn = f + sig*randn(M,N);
uw = fn;
k = 0;
t0 = cputime;
while k < K,
   uww = ex_sym2(uw);
   ua = uww(2:(M+1),3:(N+2)); % u_i+1,j
   ub = uww(3:(M+2),2:(N+1)); % u_i,j+1
   uc = uww(2:(M+1),1:N);     % u_i-1,j
   ue = uww(1:M,2:(N+1));     % u_i,j-1
   uac = (ua - uc)/h2;
   ube = (ub - ue)/h2;
   G = 1./sqrt(uac.^2 + ube.^2 + epsi);
   X = uw + (tau(k+1)*lam)*(fn - uw);
   uw_new = aos_tv2(G,X,tau(k+1),h);
   uw = uw_new;
   k = k + 1;
end
cpt = cputime - t0;
u = uw;
map = gray(256);
figure(1)
subplot(221)
imshow(f,map)
title('test image')
subplot(222)
imshow(fn,map)
title('noisy image')
subplot(223)
imshow(u,map)
title('image restored by TV+L2')
```

```
mse0 = (norm(fn-f,'fro'))^2/(M*N);
disp('PSNR before restoration:')
psnr0 = 10*log10(255^2/mse0)
mse1 = (norm(u-f,'fro'))^2/(M*N);
disp('PSNR after TV+L2 restoration:')
psnr1 = 10*log10(255^2/mse1)
title('CPU time in seconds:')
cpt
```

2. Function aos_tv2.m

```
function u = aos_tv2(G,X,dt,h)

[M,N] = size(X);
u1 = zeros(M,N);
hi = dt/(h^2);
for i = 1:M,
   xw = X(i,:);
   gw = G(i,:);
   a = zeros(N,1);
   b = zeros(N-1,1);
   a(1) = 1 + hi*(gw(1)+gw(2));
   for k = 1:N-2,
      a(k+1) = 1 + hi*(gw(k)+2*gw(k+1)+gw(k+2));
      b(k) = -hi*(gw(k)+gw(k+1));
   end
   a(N) = 1 + hi*(gw(N-1)+gw(N));
   b(N-1) = -hi*(gw(N-1)+gw(N));
   g = b;
   z = sol_tri(a,b,g,xw);
   u1(i,:) = z(:)';
end
u2 = zeros(M,N);
for i = 1:N,
   xw = X(:,i);
   gw = G(:,i);
   a = zeros(M,1);
   b = zeros(M-1,1);
   a(1) = 1 + hi*(gw(1)+gw(2));
   for k = 1:M-2,
      a(k+1) = 1 + hi*(gw(k)+2*gw(k+1)+gw(k+2));
      b(k) = -hi*(gw(k)+gw(k+1));
   end
   a(M) = 1 + hi*(gw(M-1)+gw(M));
   b(M-1) = -hi*(gw(M-1)+gw(M));
   g = b;
   z = sol_tri(a,b,g,xw);
   u2(:,i) = z(:);
end
u = 0.5*(u1 + u2);
```

3. Function sol.tri.m

```
% To solve a strictly giagonally dominant
% tri-diagonal linear system of equations
% using the so-called Thomas algorithm.
% Written by W.-S. Lu, University of Victoria.
% Last modified: Jan. 14, 2008.
```

```
function z = sol_tri(a,b,g,x)
N = length(a);
r = b;
m(1) = a(1);
for i = 1:(N-1),
   l(i) = g(i)/m(i);
   m(i+1) = a(i+1) - l(i)*b(i);
end
y(1) = x(1);
for i = 2:N,
   y(i) = x(i) - l(i-1)*y(i-1);
end
z(N) = y(N)/m(N);
for i = (N-1):-1:1,
   z(i) = (y(i) - b(i)*z(i+1))/m(i);
end
z = z(:);
```

4. Function ex_sym2.m

```
% To expand the input 2-D signal, uw, of size
% M x N to signal uwe of size (M+2) x (N+2) by
% symmetriical reflection so that the "main part"
% of the 1st column of uwe equals the 2nd column
% of uw, and the "main part" of the last column of
% uwe equals the (N-1)th column of uw, and the
% 1st and last rows of uwe are generated similarly.
% Written by W.-S. Lu, University of Victoria.
% Last modified: Jan. 3, 2008.
function uwe = ex_sym2(uw)
[M,N] = size(uw);
w = zeros(M+2,N+2);
w(2:(M+1),2:(N+1)) = uw;
w(1,2:(N+1)) = uw(2,:);
w((M+2),2:(N+1)) = uw((M-1),:);
w(2:(M+1),1) = uw(:,2);
w(2:(M+1),(N+2)) = uw(:,(N-1));
w(1,1) = uw(2,2);
w(1,(N+2)) = uw(2,(N-1));
w((M+2),1) = uw((M-1),2);
w((M+2),(N+2)) = uw((M-1),(N-1));
uwe = w;
```

Example 3.7

```
[u,fn,psnr0,psnr1,cpt] = tv_l2_aos(reschart128,[51 51 2 1.75 1.4],7,40,0.06,2);
```

Input image: reschart128
Standard deviation: 40
randn state: 7
step size in time: [51 51 2 1.75 1.4]
λ: 0.06
number of iterations: 5
PSNR of the noisy image: 16.0555 dB
PSNR of the restored image: 21.6231 dB
CPU time: 1.2969 s

Example 3.8
[u,fn,psnr0,psnr1,cpt] = tv_l2_aos(camera256,[8 7 5.5 4.5 3],19,30,0.04,5);
Input image: camera256
Standard deviation: 30
randn state: 19
step size in time: [8 7 5.5 4.5 3]
λ : 0.04
number of iterations: 5
PSNR of the noisy image: 18.5888 dB
PSNR of the restored image: 26.3542 dB
CPU time: 5.2344 s

Example 3.9
[u,fn,psnr0,psnr1,cpt] = tv_l2_aos(boat512,[21 11 6 4 2.5],19,30,0.04,5);
Input image: boat512
Standard deviation: 30
randn state: 19
step size in time: [21 11 6 4 2.5]
λ : 0.04
number of iterations: 5
PSNR of the noisy image: 18.5689 dB
PSNR of the restored image: 27.9045 dB
CPU time: 21.4219 s

3.1.7 A Dual Formulation and Chambolle's Projection Algorithm

A. Chambolle in 2004 proposed a projection algorithm for total variation minimization based on a *dual* formulation [R5]. The algorithm has since been found producing excellent results with high efficiency. Below we start with some mathematical preliminaries [R6][R7] that are required in order to describe the algorithm.

3.1.7.1 *Epigraph and Conjugate Functional*

Let f be a function defined on a subset S of R^n, whose values are real or $\pm\infty$. The set

$$\{(u,\beta) : u \in S,\ \beta \in R,\ \beta \geq f(u)\}$$

is called the epigraph of f and is denoted by epi f. We note that f is a convex function on S if and only if epi f is a convex set in R^{n+1}.

Let V and V^* be two vector spaces that are placed in duality by a bilinear pairing (inner product for Hilbert spaces) denoted by $\langle \cdot, \cdot \rangle$. Let $J(u)$ be a functional defined on V, and one tries to find an affine functional of the form $L_{v,\alpha}(u) = \langle u, v \rangle - \alpha$ that is everywhere less than $J(u)$. In other words, one seeks to find a constant α such that $\alpha \geq \langle u, v \rangle - J(u)$ for all $u \in V$. This is possible if α is taken to be (or is greater than or equal to) $\sup_u \{\langle u, u^* \rangle - J(u)\}$. This serves as motivation to define the *conjugate functional* of $J(u)$ as

$$J^*(v) = \sup_u \left(\langle u, v \rangle - J(u) \right) \quad \text{for } v \in V^* \tag{3.67}$$

The mapping from J to J^* is called the *Legend-Fenchel transform*. Obviously $J^*(v)$ is a functional defined on V^*. The significance of the conjugate functional J^* can be understood in terms of epigraph relationships. As a matter of fact it can easily be verified that

$$(v, \alpha) \in \text{epi } J^* \quad \Leftrightarrow \quad \alpha \geq \langle u, v \rangle - \beta \text{ for all } (u, \beta) \in \text{epi } J \tag{3.68}$$

If we write $L_{u,\beta}(v) = \langle u, v \rangle - \beta$, then (3.68) becomes

$$\alpha \geq J^*(v) \quad \Leftrightarrow \quad \alpha \geq L_{u,\beta}(v) \text{ for all } (u, \beta) \in \text{epi } J$$

which implies that

$$\alpha \geq J^*(v) \quad \Leftrightarrow \quad \alpha \geq \sup_{(u,\beta) \in \text{epi } J} L_{u,\beta}(v) \tag{3.69}$$

In other words, $J^*(v)$ is the *pointwise supremum of the family of all affine functionals* $L_{u,\beta}(v)$ *for* $(u, \beta) \in \text{epi } J$. The set of functionals on a vector space, say W, where each is the pointwise supremum of a family of continuous affine functionals turns out to be of importance as it has several useful properties. In literature (see [R6] for example), the above set is denoted by $\Gamma(W)$. In fact we can state (see [R6, Chapter 1] for details):

(P1) All functionals $J(u) \in \Gamma(V)$ are convex and l.s.c.

(P2) The following properties are equivalent to each other:

- (i) $J(u) \in \Gamma(V)$
- (ii) $J(u)$ is a convex and l.s.c. functional, and if $J(u)$ takes the value $-\infty$, then $J(u)$ is identically equal to $-\infty$.

We can repeat the process of defining conjugate functional J^* to define the *bi-conjugate* (*bipolar*) functional J^{**} as

$$J^{**}(u) = \sup_{v \in V^*} \left\{ \langle v, u \rangle - J^*(v) \right\} \tag{3.70}$$

It can be shown ([R6, p. 18]) that if $J(u) \in \Gamma(V)$ then $J^{**}(u) = J(u)$.

Example 3.10

(a) It can be readily verified that TV functional $J(u) = \iint_\Omega \sqrt{u_x^2 + u_y^2}\, dx\, dy$ defined in Sobolev space $W^{1,1}(\Omega)$ convex and l.s.c., and always assumes nonnegative values. Hence by property (P2), we conclude that $J(u) \in \Gamma(V)$.

(b) The functional

$$J(u) = \sup \left\{ \int_\Omega u(x) \operatorname{div} \varphi(x)\, dx : \varphi \in C_0^1(\Omega), |\, \varphi(x) \,| \leq 1 \ \forall x \in \Omega \right\} \tag{3.71}$$

defined $u \in L^1(\Omega)$ in space BV can also be shown convex and l.s.c. assumes nonnegative values, hence by (P2) $J(u)$ in (3.71) also belongs to $\Gamma(V)$. The pairing $\langle u, v \rangle$ in (3.67) in this case can be written as $\langle u, v \rangle = \int_\Omega u(x)v(x)\,dx$, so with (3.67) and (3.71) we have

$$\begin{aligned} J^*(v) &= \sup_u \left(\langle u, v \rangle - J(u) \right) \\ &= \sup_u \left(\int_\Omega u(x)v(x)dx - \sup_{\varphi \in \Phi} \left(\int_\Omega u(x)\,\mathrm{div}\varphi(x)\,dx \right) \right) \\ &= \sup_u \left(\int_\Omega u(x)v(x)dx + \inf_{\varphi \in \Phi} \left(\int_\Omega u(x)\,\mathrm{div}\varphi(x)\,dx \right) \right) \\ &= \sup_u \inf_{\varphi \in \Phi} \left(\int_\Omega u(x)\left(v(x) + \mathrm{div}\varphi(x) \right) dx \right) \end{aligned} \tag{3.72}$$

where $\Phi = \left\{ \varphi : \varphi \in C_0^1(\Omega), |\, \varphi(x) \,| \le 1 \ \forall x \in \Omega \right\}$. If for function $v(x)$ there exists a sequence $\varphi_n(x) \in \left\{ \varphi : \varphi \in C_0^1(\Omega), |\, \varphi(x) \,| \le 1 \ \forall x \in \Omega \right\}$ such that $-\mathrm{div}\varphi_n(x) \to v(x)$, then the infimum in (3.72) assumes value zero for any u(x), thus $J^*(v) = 0$. On the other hand if v(x) is not such a function, than $v(x) + \mathrm{div}\varphi(x)$ will never be a zero function, consequently one can always find u(x) to make the last integral in (3.72) arbitrary large. Summarizing, we conclude that functional $J^*(v)$ is the *characteristic function* of the set

$$K = \text{the closure of } \left\{ \mathrm{div}\varphi : \varphi \in C_0^1(\Omega), |\, \varphi(x) \,| \le 1 \ \forall x \in \Omega \right\} \tag{3.73a}$$

i.e.,

$$J^*(v) = \chi_K(v) = \begin{cases} 0 & \text{if } v \in K \\ +\infty & \text{elsewhere} \end{cases} \tag{3.73b}$$

A point to stress here is that it can easily be verified that set *K* defined above is *convex*. In other words, the conjugate of a functional can be regarded as a characteristic function of closed convex set.

Furthermore, since $J(u) \in \Gamma(V)$, we have $J^{**}(u) = J(u)$ which by using (3.70) leads to

$$\begin{aligned} J(u) &= \sup_{v \in V^*} \left\{ \langle v, u \rangle - J^*(v) \right\} \\ &= \sup_{v \in K} \left\{ \langle v, u \rangle - J^*(v) \right\} \\ &= \sup_{v \in K} \langle v, u \rangle \end{aligned} \tag{3.74}$$

■

3.1.7.2 *Subgradients of Functionals*

Subgradient is a concept generalizing that of the ordinary gradient. The generalization can be carried out naturally in a framework of a pair of vector spaces that are dual to each other.

Definition Given a functional $J(u)$ in space V, w is said to be a *subgradient* of $J(u)$ and denoted by $w \in \partial J(u)$ if $J(u)$ is finite and

$$J(v) \geq J(u) + \langle w, v-u \rangle \text{ for all } v \in V \tag{3.75}$$

To understand the rationale of this definition, consider the affine functional

$$L(v) = J(u) + \langle w, v-u \rangle = \langle w, v \rangle + J(u) - \langle w, u \rangle \tag{3.76}$$

where w $\in V^*$ satisfies (3.75). We see that this linear functional is *exact* in the sense that $L(v)$ is everywhere less than or equal to $J(v)$ and, at point u, we have $L(u) = J(u)$. In the case of a convex functional J, we see (3.76) describes a "line" underneath the entire graph of J (i.e., epi J) and touches upon $J(u)$ at point u, and the line has a slope w.

An immediate consequence of this definition is the following statement: Function u solves the minimization problem $\underset{v \in V}{\text{minimize}} \; J(v)$ if and only if $0 \in \partial J(u)$.

Moreover, it can be shown ([R6, p.22]) that if $J(u) \in \Gamma(V)$, then

$$w \in \partial J(u) \Leftrightarrow u \in \partial J^*(w) \tag{3.77}$$

3.1.7.3 *Discrete TV and Divergence*

For simplicity we consider digital images of size $N \times N$ and denote by X the Euclidean space R^{NxN}. If $u \in X$, the discrete gradient ∇u is in space $Y = X \times X$ is given by $(\nabla u)_{i,j} = \left((\nabla u)^1_{i,j}, (\nabla u)^2_{i,j} \right)$ where

$$\begin{aligned} (\nabla u)^1_{i,j} &= \begin{cases} u_{i+1,j} - u_{i,j} & \text{if } i < N \\ 0 & \text{if } i = N \end{cases} \\ (\nabla u)^2_{i,j} &= \begin{cases} u_{i,j+1} - u_{i,j} & \text{if } j < N \\ 0 & \text{if } j = N \end{cases} \end{aligned} \tag{3.78}$$

for $i, j = 1, 2, \ldots, N$. Accordingly, the discrete total variation of u is defined by

$$J(u) = \sum_{1 \leq i,j \leq N} |(\nabla u)_{i,j}| \tag{3.79}$$

with $|y| = \sqrt{y_1^2 + y_2^2}$ for $y \in R^2$. The $J(u)$ in (3.79) is a discretization of the TV defined in Example 3.10a.

To find a characterization of $J(u)$ similar to the standard TV definition in (3.71) in the discrete setting, let us first define the inner product in space $Y = X \times X$ as

$$\langle p, q \rangle_Y = \sum_{1 \leq i,j \leq N} \left(p^1_{i,j} q^1_{i,j} + p^2_{i,j} q^2_{i,j} \right) \quad \text{for} \quad p = (p^1, p^2), q = (q^1, q^2)$$

Following (3.71), the discrete TV can be defined as

$$J(u) = \sup_{p \in P} \langle u, \operatorname{div} p \rangle_X \quad \text{for } p \in P = \{ p \in Y : | p_{i,j} | \le 1 \} \tag{3.80}$$

which can be written as

$$J(u) = \sup_{v \in K} \langle u, v \rangle_X \quad \text{where } K = \{ \operatorname{div} p : p \in Y, | p_{i,j} | \le 1, \forall i, j = 1, 2, \ldots, N \} \tag{3.81}$$

We note that (3.81) is merely the discrete counterpart of (3.74) and (3.73a). The discrete divergence div p is related to discrete gradient ∇u as

$$\langle u, \operatorname{div} p \rangle_X = \langle -\nabla u, p \rangle_Y \tag{3.82}$$

Based on this and (3.78), we can evaluate the discrete divergence as

$$(\operatorname{div} p)_{i,j} = \begin{cases} p^1_{i,j} - p^1_{i-1,j} & \text{if } 1 < i < N \\ p^1_{i,j} & \text{if } i = 1 \\ -p^1_{i-1,j} & \text{if } i = N \end{cases} + \begin{cases} p^2_{i,j} - p^2_{1,j-1} & \text{if } 1 < j < N \\ p^2_{i,j} & \text{if } j = 1 \\ -p^2_{1,j-1} & \text{if } j = N \end{cases} \tag{3.83}$$

3.1.7.4 *Problem Formulation and the Euler-Lagrange Equation*

As studied in Sec. 3.1.3, the denoising problem can be formulated as solving the unconstrained problem

$$\underset{u \in X}{\text{minimize}} \ \frac{\| u - u_0 \|^2}{2\lambda} + J(u) \tag{3.84}$$

where u_0 is the observed data and the norm involved is the Euclidean norm in X given by $\| u \|^2 = \langle u, u \rangle_X$ and $\lambda > 0$ is a weight. The EL equation associated with problem (3.84) can be stated as

$$0 \in u - u_0 + \lambda \, \partial J(u) \tag{3.85}$$

3.1.7.5 *Main Points of the Algorithm*

We begin by writing (3.85) as $(u_0 - u)/\lambda \in \partial J(u)$ which by using (3.77) gives $u \in \partial J^* \left((u_0 - u)/\lambda \right)$. By writing the last expression as

$$\frac{u_0}{\lambda} \in \frac{u_0 - u}{\lambda} + \frac{1}{\lambda} \partial J^* \left(\frac{u_0 - u}{\lambda} \right)$$

it is realized that $w = (u_0 - u)/\lambda$ is the minimizer of the functional

$$\frac{\| w - (u_0/\lambda) \|^2}{2} + \frac{1}{\lambda} J^*(w) \tag{3.86}$$

Since J^* is given by (3.73b), the minimizer must be in set K so that $J^*(w) = 0$ and at the same time w is required to minimize the Euclidean norm in (3.86). This occurs when w is the projection of u_0/λ onto set K, i.e., $w = \pi_K(u_0/\lambda)$ and, therefore,

$$u = u_0 - \lambda w = u_0 - \lambda \pi_K(u_0/\lambda) = u_0 - \pi_{\lambda K}(u_0) \tag{3.87}$$

The problem now amounts to computing the nonlinear projection of u_0 onto convex set λK. From (3.81) we have $\lambda K = \left\{\lambda \operatorname{div} p : p \in Y, |p_{i,j}|^2 - 1 \le 0 \ \forall i, j = 1,2,\ldots N\right\}$, hence computing projection $\pi_{\lambda K}(u_0)$ amounts to solving

$$\begin{aligned} &\text{minimize} \quad \left\|\lambda \operatorname{div} p - u_0\right\| \\ &\text{subject to}: \ p \in Y, \ |p_{i,j}|^2 - 1 \le 0 \ \ \forall i, j = 1,2,\ldots,N \end{aligned} \tag{3.88}$$

3.1.7.6 *Solving Problem (3.88)*

Problem (3.88) has a total of N^2 constraints, it is associated with N^2 Lagrange multipliers $\{\alpha_{i,j}\}$. The Karush-Kuhn-Tucker (KKT) conditions for problem (3.88) are given by

$$-\left(\nabla\left(\lambda \operatorname{div} p - u_0\right)\right)_{i,j} + \alpha_{i,j} p_{i,j} = 0 \qquad \forall i, j = 1,2,\ldots,N \tag{3.89a}$$

and

$$\alpha_{i,j} \begin{cases} > 0 \ \text{ if } \ |p_{i,j}| = 1 \\ = 0 \ \text{ if } \ |p_{i,j}| < 1 \end{cases} \tag{3.89b}$$

which are referred to as complimentarity conditions. From these two equations, we conclude that

$$\alpha_{i,j} = \left|\left(\nabla\left(\lambda \operatorname{div} p - g\right)\right)_{i,j}\right|$$

leading (3.89a) to

$$\left(\nabla\left(\operatorname{div} p - u_0/\lambda\right)\right)_{i,j} - \left|\left(\nabla\left(\operatorname{div} p - u_0/\lambda\right)\right)_{i,j}\right| p_{i,j} = 0 \tag{3.90}$$

Eq. (3.90) serves a basis on which the following semi-implicit gradient descent algorithm is proposed [R5]:

$$p_{i,j}^{n+1} = p_{i,j}^{n} + \tau\left[\left(\nabla\left(\operatorname{div} p^n - u_0/\lambda\right)\right)_{i,j} - \left|\left(\nabla\left(\operatorname{div} p^n - u_0/\lambda\right)\right)_{i,j}\right| p_{i,j}^{n+1}\right] \tag{3.91}$$

which gives

$$p_{i,j}^{n+1} = \frac{p_{i,j}^{n} + \tau\left(\nabla\left(\operatorname{div} p^n - u_0/\lambda\right)\right)_{i,j}}{1 + \tau\left|\left(\nabla\left(\operatorname{div} p^n - u_0/\lambda\right)\right)_{i,j}\right|} \tag{3.92}$$

3.1.7.7 *Convergence and Application to Denoising*

It can be shown [R5] that if $\tau \le 1/8$, then $\lambda \operatorname{div} p^n$ converges to $\pi_{\lambda K}(u_0)$ as $n \to \infty$. The criterion for stopping the iterations is to check whether or not the maximum variation between $p_{i,j}^n$ and $p_{i,j}^{n+1}$ is less than a tolerance ε and $\varepsilon = 0.01$ was found to be a good choice [R5]. It was also remarked in [R5] that in practice it appears that the optimal constant for stability and convergence of the algorithm is not 1/8 but 1/4. The reason for this remains unknown.

In [R5] the algorithm is applied to the denoising problem that is formulated as

$$\begin{aligned} &\text{minimize} \quad J(u) \\ &\text{subject to: } \|u - u_0\|^2 = N^2\sigma^2 \end{aligned} \tag{3.93}$$

where N^2 is the total number of pixels in the image. Obviously the problem in (3.93) is a constrained problem and the value of $\lambda > 0$ involved in the projection $\pi_{\lambda K}$ needs to be determined such that $\|\pi_{\lambda K} u_0\|^2 = N^2\sigma^2$. Below we give a step-by-step description of the algorithm but refer the reader to reference [R5] for the technical details.

Chambolle's Denoising Algorithm

Step 1: Choose an arbitrary value $\lambda_0 > 0$, set $n = 0$ and convergence tolerances ε_1 and ε_2.

Step 2: Apply the algorithm in Sec. 3.1.7.6 with tolerance ε_1 to compute $v_n = \pi_{\lambda_n K}(u_0)$ and $f_n = \|v_n\|$.

Step 3: Update $\lambda_{n+1} = (N\sigma / f_n)\lambda_n$, apply the algorithm in Sec. 3.1.7.6 with tolerance ε_1 to compute $v_{n+1} = \pi_{\lambda_{n+1} K}(u_0)$ and $f_{n+1} = \|v_{n+1}\|$.

Step 4: If $|f_{n+1} - f_n| < \varepsilon_2$, stop and compute solution $u = u_0 - v_{n+1}$; otherwise set $n = n + 1$ and repeat from Step 3.

It is shown in [R5] that as $n \to \infty$, $f_n \to N\sigma$ and $u_0 - v_n$ converges to the unique solution of problem (3.93). Finally, it is remarked [R5] that in practice, one can replace λ with the new value $N\sigma / \|\operatorname{div} p^n\|$ after each iteration (3.92) and get a very quick convergence to the solution of the problem in (3.93).

3.1.7.8 *MATLAB Code and Experimental Results*

The main function that implements Chambolle's projection algorithm is chambolle.m which calls for another function ex_sym1.m. These functions are listed below.

1. Function chambolle.m

```
% The code is based on the implementation scheme of the
% Chambolle2004  algorithm described in
% Chambolle, "An algorithm for total variation minimization
% and applications," J. Math. Imaging and Vision, vol. 20,
% pp. 89-97, 2004.
% Inputs:
% f    -- test image
```

```
% st   -- state for Gaussian white noise generation
% sig  -- sigma, standard deviation of the noise
% lam0 -- initial value of lambda (the weight associated
%         with the fidelity term)
% K    -- number of iterations.
% Outputs:
% u      -- restored image
% fnoise -- noise-corrupted image
% fn     -- norm of vn in the algorithm
% psnr0  -- PSNR before de-noising
% psnr1  -- PSNR after denoising
% cpt    -- CPU time in seconds
% Examples:
% [u,fnoise,fn,psnr0,psnr1,cpt] = chambolle(reschart128,7,40,0.1,4);
% [u,fnoise,fn,psnr0,psnr1,cpt] = chambolle(camera256,19,30,0.1,5);
% [u,fnoise,fn,psnr0,psnr1,cpt] = chambolle(boat512,19,30,0.1,5);
% Written by W.-S. Lu, University of Victoria.
% Last modified: Jan. 15, 2008.

function [u,fnoise,fn,psnr0,psnr1,cpt] = chambolle(f,st,sig,lam0,K)
epsi = 0.01;
tau = 0.25;
[M,N] = size(f);
randn('state',st)
fnoise = f + sig*randn(M,N);
lamw = lam0;
nsig = sqrt(M*N)*sig;
k = 0;
t0 = cputime;
fn = zeros(K,1);
while k < K,
   p1 = zeros(M,N);
   p2 = zeros(M,N);
   p = [p1 p2];
   er = 1;
   while er >= epsi,
      pw1 = ex_sym1(p1);
      pw2 = ex_sym1(p2);
      pc1 = pw1(2:(M+1),1:N);
      pe2 = pw2(1:M,2:(N+1));
      dp1 = p1 - pc1;
      dp1(:,1) = p1(:,1);
      dp1(:,N) = -p1(:,N-1);
      dp2 = p2 - pe2;
      dp2(1,:) = p2(1,:);
      dp2(N,:) = -p2(N-1,:);
      dp = dp1 + dp2;
      pw = dp - fnoise/lamw;
      pww = ex_sym1(pw);
      pwa = pww(2:(M+1),3:(N+2));
      pwb = pww(3:(M+2),2:(N+1));
      gp1 = pwa - pw;
      gp2 = pwb - pw;
      gp = [gp1 gp2];
      pnw1 = p + tau*gp;
      pnw2 = 1 + tau*abs(gp);
      p_new = pnw1./pnw2;
```

```
        er = max(max(abs(p - p_new)));
        p = p_new;
        p1 = p(:,1:N);
        p2 = p(:,(N+1):2*N);
    end
    p1 = p(:,1:N);
    p2 = p(:,(N+1):2*N);
    pw1 = ex_sym1(p1);
    pw2 = ex_sym1(p2);
    pc1 = pw1(2:(M+1),1:N);
    pe2 = pw2(1:M,2:(N+1));
    dp1 = p1 - pc1;
    dp1(:,1) = p1(:,1);
    dp1(:,N) = -p1(:,N-1);
    dp2 = p2 - pe2;
    dp2(1,:) = p2(1,:);
    dp2(N,:) = -p2(N-1,:);
    dp = dp1 + dp2;
    vk = lamw*dp;
    f1k = sqrt(sum(sum(vk.^2)));
    lamw = (nsig/f1k)*lamw;
    fn(k+1) = f1k;
%     f2k = sqrt(sum(sum(dp.^2)));
%     lamw = nsig/f2k;
%     fn(k+1) = f2k;
    k = k + 1;
end
cpt = cputime - t0;
u = fnoise - vk;
map = gray(256);
figure(1)
subplot(221)
imshow(f,map)
title('test image')
subplot(222)
imshow(fnoise,map)
title('noisy image')
subplot(223)
imshow(u,map)
title('image restored by Chambolle projection')
mse0 = (norm(fnoise-f,'fro'))^2/(M*N);
disp('PSNR before restoration:')
psnr0 = 10*log10(255^2/mse0)
mse1 = (norm(u-f,'fro'))^2/(M*N);
disp('PSNR after TV+L2 restoration:')
psnr1 = 10*log10(255^2/mse1)
disp('CPU time in seconds:')
cpt
```

2. Function ex_sym1.m

```
% To expand the input 2-D signal, uw, of size
% M x N to signal uwe of size (M+2) x (N+2)
% so that the "main part" of the 1st column of
% uwe equals the 1st column of uw, and the "main
% part" of the last column of uwe equals the last
```

```
% column of uw, and the 1st and last rows of uwe
% are generated similarly.
% Written by W.-S. Lu, University of Victoria.
% Last modified: Jan. 20, 2008.
function uwe = ex_sym1(uw)
[M,N] = size(uw);
w = zeros(M+2,N+2);
w(2:(M+1),2:(N+1)) = uw;
w(1,2:(N+1)) = uw(1,:);
w((M+2),2:(N+1)) = uw(M,:);
w(2:(M+1),1) = uw(:,1);
w(2:(M+1),(N+2)) = uw(:,N);
w(1,1) = uw(1,1);
w(1,(N+2)) = uw(1,N);
w((M+2),1) = uw(M,1);
w((M+2),(N+2)) = uw(M,N);
uwe = w;
```

Example 3.11

```
[u,fnoise,fn,psnr0,psnr1,cpt] = chambolle(reschart128,7,40,0.1,4);
```

Input image: reschart128
Standard deviation: 40
randn state: 7
$\tau = 0.25$
initial value of λ : 0.1
number of iterations: 4
PSNR of the noisy image: 16.0555 dB
PSNR of the restored image: 23.7505 dB
CPU time: 1.0469 s

The restored image is shown in Fig. 3.9.

Example 3.12

```
[u,fnoise,fn,psnr0,psnr1,cpt] = chambolle(camera256,19,30,0.1,5);
```

Input image: camera256
Standard deviation: 30
randn state: 19
$\tau = 0.25$
initial value of λ : 0.1
number of iterations: 5
PSNR of the noisy image: 18.5888 dB
PSNR of the restored image: 26.8392 dB
CPU time: 8.2656 s

The restored image is shown in Fig. 3.10.

Example 3.13

```
[u,fnoise,fn,psnr0,psnr1,cpt] = chambolle(boat512,19,30,0.1,5);
```

Input image: boat512
Standard deviation: 30
randn state: 19
$\tau = 0.25$

initial value of λ : 0.1
number of iterations: 5
PSNR of the noisy image: 18.5689 dB
PSNR of the restored image: 27.9308 dB
CPU time: 37.2500 s

The restored image is shown in Fig. 3.11.

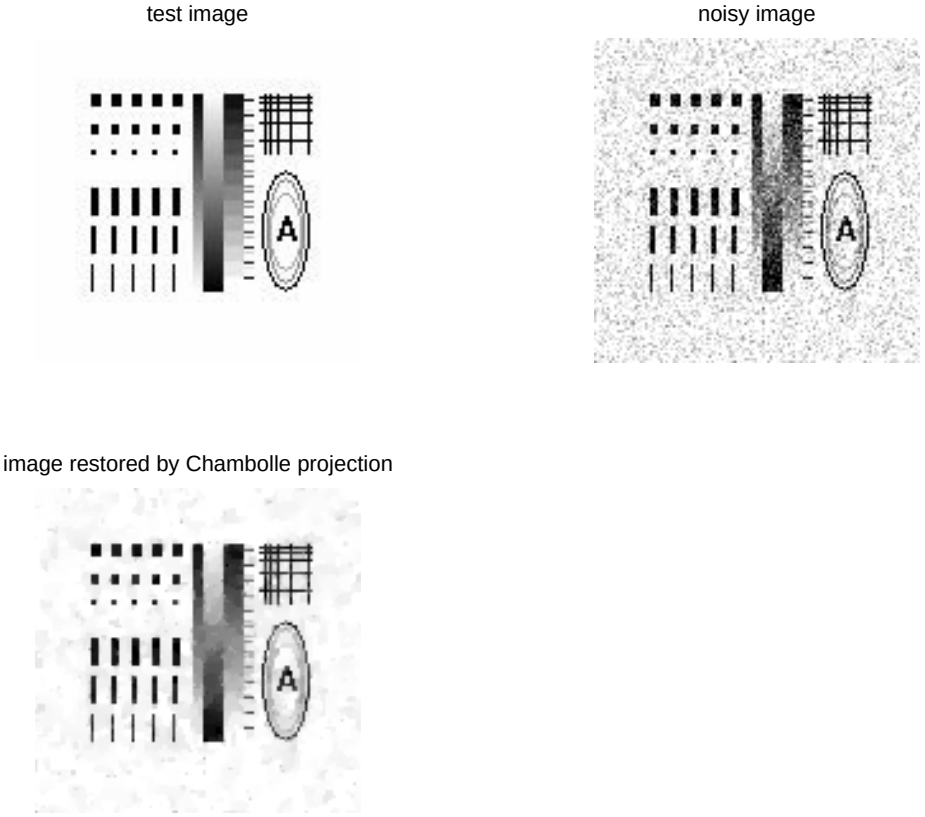


Figure 3.9: For Example 3.11

test image

noisy image

image restored by Chambolle projection

Figure 3.10: For Example 3.12

test image

noisy image

image restored by Chambolle projection

Figure 3.11: For Example 3.13

Additional References

[R1] R. Acar and C. R. Vogel, "Analysis of bounded variation penalty methods for ill-posed problems," *Inverse Problems*, vol. 10, pp. 1217-1229, 1994.
[R2] C. R. Vogel and M. E. Oman, "Iterative methods for total variation denoising," *SIAM J. Sci. Comput.*, vol. 17, no. 1, pp. 227-238, Jan. 1996.
[R3] F. Santosa and W. Symes, "Reconstruction of blocky impedance profiles from normal-incidence reflection seismographs which are band-limited and miscalibrated," *Wave Motion*, vol. 10, pp. 209-230, 1988.
[R4] C. R. Vogel, "Total variation regularization for ill-posed problems," Tech. Report, Dept. of Math. Sci., Montana State University, 1993.
[R5] A. Chambolle, "An algorithm for total variation minimization and applications," *J. Math. Imaging and Vision*, vol. 20, pp. 89-97, 2004.
[R6] I. Ekeland and R. Teman, *Convex Analysis and Variational Problems,* North-Holland Publishing Company, Amsterdam, 1976.
[R7] R. T. Rockafflar and R. J.-B. Wets, *Variational Analysis*, Springer 1998.

3.2 Image Restoration by Nonlinear Diffusion

In this section we study image restoration techniques based on nonlinear diffusion. Our focus in this section will be on the well-known Perona-Malik anisotropic diffusion method [11] and the method by Catte-Lions-Morel-Coll [12].

3.2.1 Image Diffusion in Scale Space

Diffusion of a noisy image modeled by

$$u_0(x, y) = u(x, y) + w(x, y)$$

in a scale space means to perform lowpass filtering with gradually narrowing passband. Here we regard filtered image as an element in a scale space where a coarser scale corresponds to a narrower passband. Typically, *isotropic diffusion* of an image $u_0(x, y)$ can be carried out by convolving the image with a Gaussian kernel:

$$u(x, y, \sigma) = G_\sigma(x, y) \otimes u_0(x, y) \tag{3.94a}$$

where

$$G_\sigma(x, y) = \frac{1}{4\pi\sigma} e^{-(x^2+y^2)/4\sigma} \tag{3.94b}$$

It follows that for $\sigma > 0$ we always have

$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} G_\sigma(x, y)\, dx\, dy = 1$$

and

$$\lim_{\sigma\to 0} G_\sigma(x, y) = \begin{cases} 0 & \text{for } (x, y) \neq (0, 0) \\ \infty & \text{for } (x, y) = (0, 0) \end{cases}$$

which means that as $\sigma \to 0$, the Gaussian kernel acts likes a $\delta-$function, namely

$$u(x, y, 0) = \delta(x, y) \otimes u_0(x, y) = u_0(x, y)$$

We can also examine the Gaussian kernel in the frequency domain by taking 2-D Fourier transform of $G_\sigma(x, y)$. This gives

$$2-\text{D Fourier transform of } G_\sigma(x, y) = e^{-\sigma(\omega_1^2+\omega_2^2)}$$

This means that $G_\sigma(x, y)$ is the impulse response of a 2-D circularly symmetric lowpass filter whose

passband width is inversely proportional to the diffusion parameter σ.

An important observation made by Witkin (1983) and Koenderink (1984) is that that if we interpret parameter σ as "time" *t*, performing the convolution of an image with the Gaussian kernel at each scale is then equivalent to solving the heat equation

$$u_t = u_{xx} + u_{yy} \tag{3.95a}$$

subject to initial condition

$$u(x, y, 0) = u_0(x, y) \tag{3.95b}$$

In other words, $u(x, y, t) = G_t(x, y) \otimes u_0(x, y)$ satisfies the equation in (3.95a) and the initial condition in (3.95b). This fact can be verified as follows.

$$u(x, y, t) = G_t(x, y) * u_0(x, y) = \frac{1}{4\pi t} \iint_{R^2} e^{-[(x-x')^2+(y-y')^2]/4t} u_0(x', y')\, dx'\, dy'$$

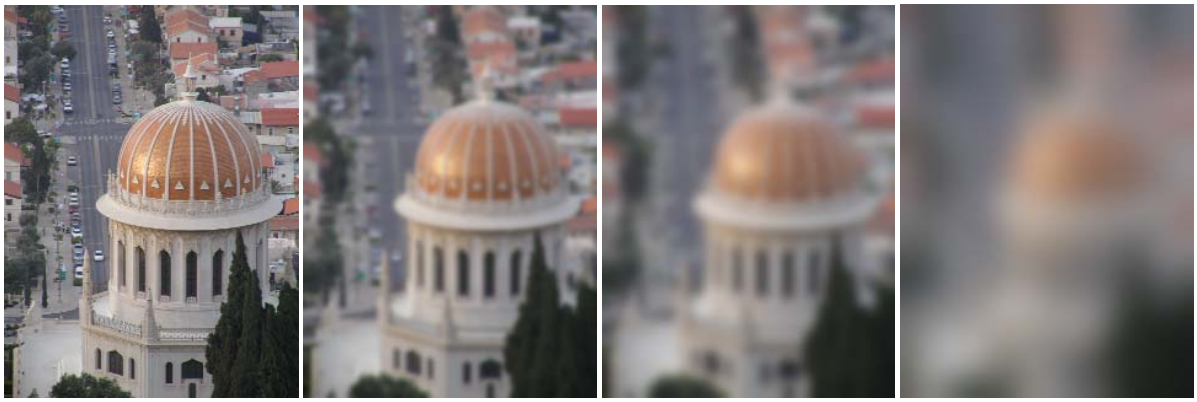
$$\frac{\partial u(x, y, t)}{\partial t} = \frac{1}{4\pi} \iint_{R^2} \left[\frac{(x-x')^2 + (y-y')^2}{4t^3} - \frac{1}{t^2} \right] \cdot e^{-[(x-x')^2+(y-y')^2]/4t} u_0(x', y')\, dx'\, dy'$$

$$\frac{\partial^2 u(x, y, t)}{\partial x^2} = \frac{1}{4\pi} \iint_{R^2} \left[\frac{(x-x')^2}{4t^3} - \frac{1}{2t^2} \right] \cdot e^{-[(x-x')^2+(y-y')^2]/4t} u_0(x', y')\, dx'\, dy'$$

$$\frac{\partial^2 u(x, y, t)}{\partial y^2} = \frac{1}{4\pi} \iint_{R^2} \left[\frac{(y-y')^2}{4t^3} - \frac{1}{2t^2} \right] \cdot e^{-[(x-x')^2+(y-y')^2]/4t} u_0(x', y')\, dx'\, dy'$$

$$\Rightarrow \qquad \frac{\partial u(x, y, t)}{\partial t} = \frac{\partial^2 u(x, y, t)}{\partial x^2} + \frac{\partial^2 u(x, y, t)}{\partial y^2} \quad !!!$$

See Figure 3.12 for an illustrative example of isotropic diffusion by (3.95).



(a) (b) (c) (d)

Figure 3.12 (a) Original image and its isotropic diffusion with (b) *t* = 2, (c) *t* = 10, and (d) *t* = 100.

The connection of Gaussian lowpass filtering to the heat equation gives a good reason for calling the lowpass filtering a diffusion process. More importantly, the PDE interpretation of image diffusion opens a door to the development of a very effective image restoration/edge detection/segmentation method – the Perona-Malik (PM) anisotropic diffusion.

3.2.2 The Perona-Malik Anisotropic Diffusion

In 1990, Perona and Malik [11] proposed a method for edge detection by using *anisotropic diffusion*. The term anisotropic diffusion was used to stress the use a kind of image-selective lowpass filtering technique compared to the isotropic diffusion via the heat equation (3.95). To analyze and understand the method quantitatively, note that the heat equation in (3.95) can be expressed as

$$u_t = \mathrm{div}\left[\nabla u(x, y)\right] \tag{3.96a}$$

where

$$\nabla u(x, y) = \begin{bmatrix} u_x \\ u_y \end{bmatrix}, \quad \mathrm{div}\begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \frac{\partial a_1}{\partial x} + \frac{\partial a_2}{\partial y} \tag{3.96b}$$

In the P-M anisotropic diffusion, images are diffused by nonlinear diffusion which can be described as

$$u_t = \mathrm{div}\left[g\left(|\nabla u(x, y)| / K\right) \nabla u(x, y) \right] \tag{3.97a}$$

subject to the initial condition

$$u(x, y, 0) = u_0(x, y) \tag{3.97b}$$

where $g\left(|\nabla u(x, y)| / K\right)$, known as an *edge detector*, is a smooth function of scaled magnitude of image gradient, $|\nabla u| / K$. Obviously, the P-M model (3.97) includes the isotropic diffusion as a special case where the diffusion coefficient $g\left(|\nabla u(x, y)| / K\right) \equiv 1$. By incorporating an image-dependent $g\left(|\nabla u(x, y)| / K\right)$ into the diffusion process, the diffusion can be made *selective to image features*. For denoising, edge detection, and image segmentation, the P-M model uses

$$g\left(\nabla u | / K\right) = e^{-\left(|\nabla u| / K\right)^2} \quad \text{or} \quad g\left(\nabla u | / K\right) = \frac{1}{1 + \left(|\nabla u| / K\right)^2} \tag{3.98}$$

where $K > 0$ is a threshold to deal with noise. The above $g\left(|\nabla u(x,y)|/K\right)$ have one thing in common: at each edge point, the magnitude of the gradient $|\nabla u|$ is very large, leading to a near zero g which means that *no* (*or very little*) *diffusion is performed along edges*. On the other hand, in the interior of each region, $|\nabla u|$ is small, hence $g\left(|\nabla u(x,y)|/K\right)$ is close to one and diffusion is performed as usual.

3.2.3 A Semi-Implicit Scheme for (3.97a), (3.98)

For notation simplicity, in what follows we write the edge detector g in (3.97a) as a smooth function of $|\nabla u|^2 = u_x^2 + u_y^2$, thus (3.97a) can be written as

$$\frac{\partial u}{\partial t} = \frac{\partial}{\partial x}\left(g\left(u_x^2+u_y^2\right)\frac{\partial u}{\partial x}\right)+\frac{\partial}{\partial y}\left(g\left(u_x^2+u_y^2\right)\frac{\partial u}{\partial y}\right) \tag{3.99}$$

Based on this, an implicit discrete approximation of the P-M model can be deduced as

$$\begin{aligned}\frac{u_{i,j}^{n+1}-u_{i,j}^{n}}{\tau} &= \delta_x^-\left(g\left[\left(\delta_x^+u_{i,j}^n\right)^2+\left(u_{i,j+1}^n-u_{i,j-1}^n\right)^2/2^2\right]\cdot\delta_x^+u_{i,j}^{n+1}\right)\\ &+\delta_y^-\left(g\left[\left(u_{i+1,j}^n-u_{i-1,j}^n\right)^2/2^2+\left(\delta_y^+u_{i,j}^n\right)^2\right]\cdot\delta_y^+u_{i,j}^{n+1}\right)\end{aligned} \tag{3.100a}$$

where

$$\begin{aligned}&\delta_x^-\left(g\left[\left(\delta_x^+u_{i,j}^n\right)^2+\left(u_{i,j+1}^n-u_{i,j-1}^n\right)^2/2^2\right]\cdot\delta_x^+u_{i,j}^{n+1}\right)\\ &= g\left[\left(u_{i+1,j}^n-u_{i,j}^n\right)^2+\left(u_{i,j+1}^n-u_{i,j-1}^n\right)^2/2^2\right]\cdot\left(u_{i+1,j}^{n+1}-u_{i,j}^{n+1}\right)\\ &-g\left[\left(u_{i,j}^n-u_{i-1,j}^n\right)^2+\left(u_{i-1,j+1}^n-u_{i-1,j-1}^n\right)^2/2^2\right]\cdot\left(u_{i,j}^{n+1}-u_{i-1,j}^{n+1}\right)\end{aligned} \tag{3.100b}$$

and

$$\begin{aligned}&\delta_y^-\left(g\left[\left(u_{i+1,j}^n-u_{i-1,j}^n\right)^2/2^2+\left(\delta_y^+u_{i,j}^n\right)^2\right]\cdot\delta_y^+u_{i,j}^{n+1}\right)\\ &= g\left[\left(u_{i+1,j}^n-u_{i-1,j}^n\right)^2/2^2+\left(u_{i,j+1}^n-u_{i,j}^n\right)^2\right]\cdot\left(u_{i,j+1}^{n+1}-u_{i,j}^{n+1}\right)\\ &-g\left[\left(u_{i+1,j-1}^n-u_{i-1,j-1}^n\right)^2/2^2+\left(u_{i,j}^n-u_{i,j-1}^n\right)^2\right]\cdot\left(u_{i,j}^{n+1}-u_{i,j-1}^{n+1}\right)\end{aligned} \tag{3.100c}$$

If we define

$$G_1 = g\left[\left(u^n_{i+1,j} - u^n_{i,j}\right)^2 + \left(u^n_{i,j+1} - u^n_{i,j-1}\right)^2 / 2^2\right]$$

$$G_2 = g\left[\left(u^n_{i,j} - u^n_{i-1,j}\right)^2 + \left(u^n_{i-1,j+1} - u^n_{i-1,j-1}\right)^2 / 2^2\right]$$

$$G_3 = g\left[\left(u^n_{i+1,j} - u^n_{i-1,j}\right)^2 / 2^2 + \left(u^n_{i,j+1} - u^n_{i,j}\right)^2\right]$$

$$G_4 = g\left[\left(u^n_{i+1,j-1} - u^n_{i-1,j-1}\right)^2 / 2^2 + \left(u^n_{i,j} - u^n_{i,j-1}\right)^2\right]$$

then (3.100a) becomes

$$\frac{u^{n+1}_{i,j} - u^n_{i,j}}{\tau} = G_1 \cdot u^{n+1}_{i+1,j} + G_2 \cdot u^{n+1}_{i-1,j} + G_3 \cdot u^{n+1}_{i,j+1} + G_4 \cdot u^{n+1}_{i,j-1} - (G_1 + G_2 + G_3 + G_4) \cdot u^{n+1}_{i,j} \qquad (3.101)$$

At this point a relaxation is made to modify (3.101) to

$$\frac{u^{n+1}_{i,j} - u^n_{i,j}}{\tau} = G_1 \cdot u^n_{i+1,j} + G_2 \cdot u^n_{i-1,j} + G_3 \cdot u^n_{i,j+1} + G_4 \cdot u^n_{i,j-1} - (G_1 + G_2 + G_3 + G_4) \cdot u^{n+1}_{i,j} \qquad (3.102)$$

which leads to a semi-implicit scheme for (3.97a) as

$$u^{n+1}_{i,j} = \frac{1}{G} \cdot \left(u^n_{i,j} + \tau \cdot \left(G_1 \cdot u^n_{i+1,j} + G_2 \cdot u^n_{i-1,j} + G_3 \cdot u^n_{i,j+1} + G_4 \cdot u^n_{i,j-1}\right)\right) \qquad (3.103a)$$

where

$$G = 1 + \tau \cdot (G_1 + G_2 + G_3 + G_4) \qquad (3.103b)$$

3.2.4 MATLAB Code and Examples for the P-M Nonlinear Diffusion

The MATLAB code as a function pm_diffusion.m that implements the implicit scheme (3.103) subject to the initial condition (3.97b) is given below. The function requires another function ex_sym2.m which was also used in the codes given in Sec. 3.1.

1. Function pm_diffusion.m

```
% Perona-Malik nonlinear diffusion where the edge
% detector is given by exp(-abs(gradient(u))/K)^2)
% or by 1/(1 + (abs(gradient(u))/K)^2).
% Reference:P. Perona and J. Malik, “Scale-space and
% edge detection using anisotropic diffusion,” IEEE Trans.
% Pattern Analysis and Machine Intelligence, vol. 12,
% pp. 629-639, July 1990.
% A semi-implicit scheme is used in the MATLAB implemention
% of the algorithm. The scheme is in spirit similar to the
% scheme used for the Chan-Vese segmentation algorithms (2-phase
% as well as 4-phase formulations) and its details are given
% in the ELEC 639B notes.
```

```
% Inputs:
% f       -- test image
% dt      -- stepsize in time
% st      -- state for Gaussian white noise generation
% sig     -- sigma, standard deviation of the noise
% iter    -- number of iterations
% K       -- parameter in diffusion coefficient c(u)
% ver     -- ver = 1 corresponds to using an exponential c(u)
%            ver = 2 corresponds to using a fractional c(u).
% Outputs:
% u           -- restored image
% fn          -- noise-corrupted image
% snr0        -- SNR before de-noising
% snr1        -- SNR after denoising
% snr1_rof -- SNR after denoising by using ROF model
% cpt         -- CPU time in seconds consumed by PM model
% Examples:
% [u,snr1,snr1_rof,snr0,cpt] = pm_diffusion(circles256,5,19,25,150,27,1);
% [u,snr1,snr1_rof,snr0,cpt] = pm_diffusion(circles256,0.5,19,25,150,5,2);
% [u,snr1,snr1_rof,snr0,cpt] = pm_diffusion(camera256,0.1,19,25,30,40,1);
% [u,snr1,snr1_rof,snr0,cpt] = pm_diffusion(camera256,0.1,19,25,32,19,2);
% [u,snr1,snr1_rof,snr0,cpt] = pm_diffusion(boat512,0.1,19,25,30,20,2);
% Written by W.-S. Lu, University of Victoria.
% Last modified: March 30, 2008.
function [u,snr1,snr1_rof,snr0,cpt] = pm_diffusion(f,dt,st,sig,iter,K,ver)
[M,N] = size(f);
randn('state',st);
n = randn(M,N);
n = n - mean(mean(n));
sig0 = norm(n,'fro')/sqrt(M*N);
n = (sig/sig0)*n;
fn = f + n;
snr0 = 20*log10(norm(f,'fro')/norm(n,'fro'));
map = gray(256);
K2 = K^2;
pw = fn;
it = 0;
t0 = cputime;
while it < iter,
   pww = ex_sym2(pw);
   pa = pww(2:(M+1),3:(N+2)); % phi_i+1,j
   pb = pww(3:(M+2),2:(N+1)); % phi_i,j+1
   pc = pww(2:(M+1),1:N);     % phi_i-1,j
   pd = pww(3:(M+2),1:N);     % phi_i-1,j+1
   pe = pww(1:M,2:(N+1));     % phi_i,j-1
   pf = pww(1:M,3:(N+2));     % phi_i+1,j-1
   pg = pww(1:M,1:N);         % phi_i-1,j-1
   E1 = (pa - pw).^2 + ((pb - pe)/2).^2;
   E2 = (pw - pc).^2 + ((pd - pg)/2).^2;
   E3 = (pb - pw).^2 + ((pa - pc)/2).^2;
   E4 = (pw - pe).^2 + ((pf - pg)/2).^2;
   Cw1 = E1/K2;
   Cw2 = E2/K2;
   Cw3 = E3/K2;
```

```
    Cw4 = E4/K2;
    if ver == 1,
       G1 = exp(-Cw1);
       G2 = exp(-Cw2);
       G3 = exp(-Cw3);
       G4 = exp(-Cw4);
    elseif ver == 2,
       G1 = 1./(1+Cw1);
       G2 = 1./(1+Cw2);
       G3 = 1./(1+Cw3);
       G4 = 1./(1+Cw4);
    end
    F1 = G1.*pa;
    F2 = G2.*pc;
    F3 = G3.*pb;
    F4 = G4.*pe;
    G = 1 + dt*(G1 + G2 + G3 + G4);
    pw_new = (pw + dt*(F1 + F2 + F3 + F4))./G;
    pw = pw_new;
    it = it + 1;
end
cpt = cputime - t0;
u = pw;
snr1 = 20*log10(norm(f,'fro')/norm(u-f,'fro'));
disp('SNR for Perona-Malik model')
[snr0 snr1]
disp('CPU time for PM model:')
cpt
% for circles256:
% [u_rof,fn_rof,snr0_rof,snr1_rof,cpt_rof] = tv_rof_m(f,0.25,19,25,0.02,140);
% for camera256:
% [u_rof,fn_rof,snr0_rof,snr1_rof,cpt_rof] = tv_rof_m(f,0.25,19,25,0.01,67);
% for boat512:
[u_rof,fn_rof,snr0_rof,snr1_rof,cpt_rof] = tv_rof_m(f,0.25,19,25,0.006,70);
disp('SNR for ROF model:')
[snr0_rof snr1_rof]
disp('CPU time for ROF model:')
cpt_rof
w1 = 128*ones(M,10);
w2 = 128*ones(10,2*N+10);
R = [f w1 fn; w2; u_rof w1 u];
figure(1)
imshow(R,map)
```

Example 3.14

```
[u,snr1,snr1_rof,snr0,cpt] = pm_diffusion(circles256,0.5,19,25,150,5,2);
```

for [u,snr1,snr1_rof,snr0,cpt] = pm_diffusion(f,dt,st,sig,iter,K,ver)
Input image: circles256
Diffusion: Perona-Malik with a fractional edge detector
Standard deviation: 25
randn state: 19
$\tau = 0.5$
value of K: 5

number of iterations: 150
PSNR of the noisy image: 14.2072 dB
PSNR of the restored image: 33.6983 dB
CPU time: 6.4531 s

For comparison purposes, The ROF algorithm with a constant λ implemented by function tv_rof_m.m was also applied to the same noisy image as given above, and the results obtained are as follows:

[u_rof,fn_rof,snr0_rof,snr1_rof,cpt_rof] = tv_rof_m(circles256,0.25,19,25,0.02,140);
Input image: circles256
Standard deviation: 25
randn state: 19
τ = 0.5
value of λ: 0.02
number of iterations: 140
PSNR of the noisy image: 14.2072 dB
PSNR of the restored image: 29.8583 dB
CPU time: 19.2969 s

The restored images are depicted in Figure 3. 13.

Example 3.15
[u,snr1,snr1_rof,snr0,cpt] = pm_diffusion(camera256,0.1,19,25,32,19,2);
for [u,snr1,snr1_rof,snr0,cpt] = pm_diffusion(f,dt,st,sig,iter,K,ver)
Input image: camera256
Diffusion: Perona-Malik with a fractional edge detector
Standard deviation: 25
randn state: 19
τ = 0.1
value of *K*: 19
number of iterations: 32
PSNR of the noisy image: 14.5896 dB
PSNR of the restored image: 22.1826 dB
CPU time: 1.3906 s

For comparison purposes, The ROF algorithm with a constant λ implemented by function tv_rof_m.m was also applied to the same noisy image as given above, and the results obtained are as follows:

[u_rof,fn_rof,snr0_rof,snr1_rof,cpt_rof] = tv_rof_m(camera256,0.25,19,25,0.01,67);
Input image: camera256
Standard deviation: 25
randn state: 19
τ = 0.25
value of λ: 0.01
number of iterations: 67
PSNR of the noisy image: 14.5896 dB
PSNR of the restored image: 22.2069 dB
CPU time: 9.3906 s

The restored images are depicted in Figure 3. 14.

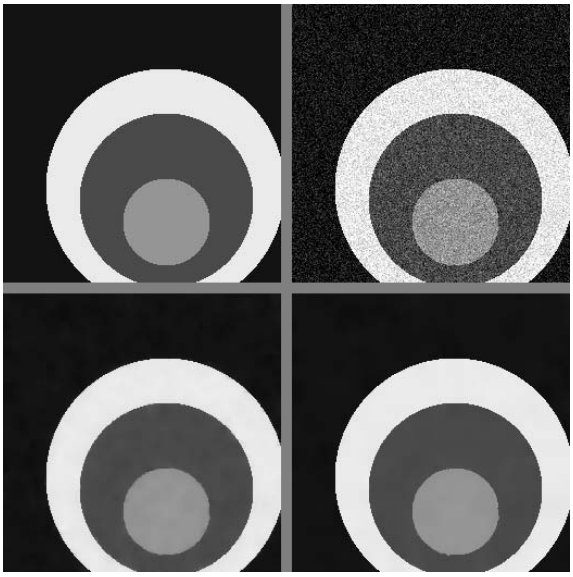
Example 3.16
[u,snr1,snr1_rof,snr0,cpt] = pm_diffusion(boat512,0.1,19,25,30,20,2);
for [u,snr1,snr1_rof,snr0,cpt] = pm_diffusion(f,dt,st,sig,iter,K,ver)
Input image: boat512
Diffusion: Perona-Malik with a fractional edge detector
Standard deviation: 25
randn state: 19
τ = 0.1
value of *K*: 20
number of iterations: 30
PSNR of the noisy image: 15.3185 dB
PSNR of the restored image: 23.7650 dB
CPU time: 5.0625 s

For comparison purposes, The ROF algorithm with a constant λ implemented by function tv_rof_m.m was also applied to the same noisy image as given above, and the results obtained are as follows:

[u_rof,fn_rof,snr0_rof,snr1_rof,cpt_rof] = tv_rof_m(f,0.25,19,25,0.006,70);
Input image: boat512
Standard deviation: 25
randn state: 19
τ = 0.25
value of λ: 0.006
number of iterations: 70
PSNR of the noisy image: 15.3185 dB
PSNR of the restored image: 24.0456 dB
CPU time: 38.7969 s

The restored images are depicted in Figure 3. 15.

(a) (b)



(c) (d)

Figure 3.13: (a) Original circles256; (b) noisy image; (c) denoised by ROF; (d) denoised by P-M model.

(a) (b)

(c) (d)

Figure 3.14: (a) Original camera256; (b) noisy image; (c) denoised by ROF; (d) denoised by P-M model.

(a) (b)

(c) (d)

Figure 3.15: (a) Original boat512; (b) noisy image; (c) denoised by ROF; (d) denoised by P-M model.

3.2.5 The Catte-Lions-Morel-Coll Nonlinear Diffusion

There are several problems encountered in the Perona-Malik model. First, the noise introduces very large,

in theory unbounded, oscillations of the gradient ∇u . The conditional smoothing of the P-M algorithm will not help since these noise "edges" will stay. Second, for the "edge detectors" *g*(*s*) used in the P-M model, i.e.,

$$g(s) = e^{-s^2} \quad \text{or} \quad g(s) = \frac{1}{1+s^2} \tag{3.104}$$

where $s = |\nabla u| / K$, a correct theory is not available to ensure that the P-M anisotropic diffusion is a stable process [12]. In Catte, Lions, Morel, and Coll [12], the general P-M model, i.e.,

$$u_t = \text{div}\left[g\left(|\nabla u(x,y)| / K\right) \nabla u(x,y) \right] \tag{3.105a}$$

where

$$g\left(|\nabla u(x,y)| / K\right) = g(s)\Big|_{s=|\nabla u|/K} \tag{3.105b}$$

is also adopted. The difference between the P-M and CLMC models is that the P-M model uses $s = |\nabla u| / K$ while the CLMC model employs $s = |\nabla G_\sigma * u|$ where G_σ is a Gaussian kernel given by

$$G_\sigma(x,y) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{(x^2+y^2)}{4\sigma}} \tag{3.106}$$

Note that

$$s = |\nabla G_\sigma * u| = |G_\sigma * \nabla u|$$

for any function *u*(*x*, *y*) with zero boundary value. Therefore, when a Gaussian kernel with a very small variance σ (i.e. as $\sigma \to 0$) is used, we have

$$G_\sigma(x,y) \to \delta(x,y) \quad \text{and} \quad s = |G_\sigma * \nabla u| = |\delta * \nabla u| = |\nabla u| \tag{3.107}$$

We see that the CLMC model essentially uses a lowpass-filtered version of the gradient in edge detector *g*(s) and in this way includes the P-M model as a special case where the passband of the filter expands to the entire frequency band. Reference [12] also establishes a theory for their model, that shows the existence of a unique smooth solution for the nonlinear parabolic diffusion equation

$$\begin{aligned} &\partial u / \partial t = \text{div}[g(|\nabla G_\sigma * u|)\nabla u] \quad \text{in} \quad (0,T]\times\Omega \\ &u(0,x,y) = u_0(x,y) \end{aligned} \tag{3.108}$$

3.2.6 A Semi-Implicit Scheme for (3.108)

A semi-implicit scheme for (3.108) may be deduced in a way similar to what was done in Sec. 3.2.3. The scheme is given by

$$u_{i,j}^{n+1} = \frac{1}{G} \cdot \left(u_{i,j}^{n} + \tau \cdot \left(G_1 \cdot u_{i+1,j}^{n} + G_2 \cdot u_{i-1,j}^{n} + G_3 \cdot u_{i,j+1}^{n} + G_4 \cdot u_{i,j-1}^{n} \right) \right) \qquad (3.109a)$$

$$G = 1 + \tau \cdot (G_1 + G_2 + G_3 + G_4) \qquad (3.109b)$$

where $G_i = g(s_i)$ for $i = 1, .., 4$ with $g(s)$ assuming one of the forms in (3.104), and

$$s_1 = \sqrt{\left[G_\sigma * \left(u_{i+1,j}^{n} - u_{i,j}^{n} \right) \right]^2 + \left[G_\sigma * \left(u_{i,j+1}^{n} - u_{i,j-1}^{n} \right) / 2 \right]^2} \Big/ K$$

$$s_2 = \sqrt{\left[G_\sigma * \left(u_{i,j}^{n} - u_{i-1,j}^{n} \right) \right]^2 + \left[G_\sigma * \left(u_{i-1,j+1}^{n} - u_{i-1,j-1}^{n} \right) / 2 \right]^2} \Big/ K$$

$$s_3 = \sqrt{\left[G_\sigma * \left(u_{i+1,j}^{n} - u_{i-1,j}^{n} \right) / 2 \right]^2 + \left[G_\sigma * \left(u_{i,j+1}^{n} - u_{i,j}^{n} \right) \right]^2} \Big/ K$$

$$s_4 = \sqrt{\left[G_\sigma * \left(u_{i+1,j-1}^{n} - u_{i-1,j-1}^{n} \right) / 2 \right]^2 + \left[G_\sigma * \left(u_{i,j}^{n} - u_{i,j-1}^{n} \right) \right]^2} \Big/ K$$

3.2.7 MATLAB Code and Examples for the CLMC Nonlinear Diffusion

The MATLAB code as a function CLMC_diffusion.m that implements the implicit scheme (3.109) subject to the initial condition in (3.108) is given below. The function requires another function ex_sym2.m which was also used in the codes given in Sec. 3.1.

1. Function CLMC_diffusion.m

```
% Catte-Lions-Morel-Coll nonlinear diffusion that calculates
% gradients of diffusion coefficient convolved with gaussian
% of variance =  sigma2.
% Reference: F. Catte, P. L. Lions, J. M. Morel and T. Coll,
% "Image selective smoothing and edge detection by nonlinear
% diffusion", SIAM J. Num. Anal., vol. 29, no. 1, pp. 182-193,
% 1992.
% A semi-implicit scheme is used in the MATLAB implemention
% of the algorithm. The scheme is in spirit similar to the
% scheme used for the Chan-Vese segmentation algorithms (2-phase
% as well as 4-phase formulations) and its details are given
% in the ELEC 639B notes.
% Inputs:
% f       -- test image
% dt      -- stepsize in time
% st      -- state for Gaussian white noise generation
% sg_n -- sigma, standard deviation of the noise
% sg_h -- sigma for Gaussian kernel
% iter    -- number of iterations
% K       -- parameter in diffusion coefficient c(u)
```

```
% ver  -- ver = 1 corresponds to using an exponential c(u)
%          ver = 2 corresponds to using a fractional c(u).
% Outputs:
% u           -- restored image
% fn          -- noise-corrupted image
% snr0        -- SNR before de-noising
% snr1        -- SNR after denoising
% snr1_rof  -- SNR after denoising by using ROF model
% cpt          -- CPU time in seconds consumed by CLMC model
% Examples:
% [u,snr1,snr1_rof,snr0,cpt] = CLMC_diffusion(circles256,5,19,25,1,130,9,1);
% [u,snr1,snr1_rof,snr0,cpt] = CLMC_diffusion(circles256,5,19,25,0.525,150,1,2);
% [u,snr1,snr1_rof,snr0,cpt] = CLMC_diffusion(camera256,0.1,19,25,1,30,40,1);
% [u,snr1,snr1_rof,snr0,cpt] = CLMC_diffusion(camera256,0.6,19,25,0.65,40,3,2);
% [u,snr1,snr1_rof,snr0,cpt] = CLMC_diffusion(boat512,0.5,19,25,0.7,32,4,2);
% Written by W.-S. Lu, University of Victoria.
% Last modified: March 30, 2008.
function [u,snr1,snr1_rof,snr0,cpt] = CLMC_diffusion(f,dt,st,sg_n,sg_h,iter,K,ver)
[M,N] = size(f);
randn('state',st);
n = randn(M,N);
n = n - mean(mean(n));
sg0 = norm(n,'fro')/sqrt(M*N);
n = (sg_n/sg0)*n;
fn = f + n;
snr0 = 20*log10(norm(f,'fro')/norm(n,'fro'));
map = gray(256);
K2 = K^2;
pw = fn;
H = fspecial('gaussian',7,sg_h);
it = 0;
t0 = cputime;
while it < iter,
   pww = ex_sym2(pw);
   pa = pww(2:(M+1),3:(N+2)); % phi_i+1,j
   pb = pww(3:(M+2),2:(N+1)); % phi_i,j+1
   pc = pww(2:(M+1),1:N);     % phi_i-1,j
   pd = pww(3:(M+2),1:N);     % phi_i-1,j+1
   pe = pww(1:M,2:(N+1));     % phi_i,j-1
   pf = pww(1:M,3:(N+2));     % phi_i+1,j-1
   pg = pww(1:M,1:N);         % phi_i-1,j-1
   gw1 = pa - pw;
   gw2 = (pb - pe)/2;
   gh1 = imfilter(gw1,H,'symmetric');
   gh2 = imfilter(gw2,H,'symmetric');
   Cw1 = (gh1.^2 + gh2.^2)/K2;
   gw1 = pw - pc;
   gw2 = (pd - pg)/2;
   gh1 = imfilter(gw1,H,'symmetric');
   gh2 = imfilter(gw2,H,'symmetric');
   Cw2 = (gh1.^2 + gh2.^2)/K2;
   gw1 = pb - pw;
   gw2 = (pa - pc)/2;
   gh1 = imfilter(gw1,H,'symmetric');
```

```
    gh2 = imfilter(gw2,H,'symmetric');
    Cw3 = (gh1.^2 + gh2.^2)/K2;
    gw1 = pw - pe;
    gw2 = (pf - pg)/2;
    gh1 = imfilter(gw1,H,'symmetric');
    gh2 = imfilter(gw2,H,'symmetric');
    Cw4 = (gh1.^2 + gh2.^2)/K2;
    if ver == 1,
       G1 = exp(-Cw1);
       G2 = exp(-Cw2);
       G3 = exp(-Cw3);
       G4 = exp(-Cw4);
    elseif ver == 2,
       G1 = 1./(1+Cw1);
       G2 = 1./(1+Cw2);
       G3 = 1./(1+Cw3);
       G4 = 1./(1+Cw4);
    end
    F1 = G1.*pa;
    F2 = G2.*pc;
    F3 = G3.*pb;
    F4 = G4.*pe;
    G = 1 + dt*(G1 + G2 + G3 + G4);
    pw_new = (pw + dt*(F1 + F2 + F3 + F4))./G;
    pw = pw_new;
    it = it + 1;
end
cpt = cputime - t0;
u = pw;
snr1 = 20*log10(norm(f,'fro')/norm(u-f,'fro'));
disp('SNR for CLMC model')
[snr0 snr1]
disp('CPU time for CLMC model:')
cpt
% for circles256:
[u_rof,fn_rof,snr0_rof,snr1_rof,cpt_rof] = tv_rof_m(f,0.25,10,25,0.02,140);
% for camera256:
% [u_rof,fn_rof,snr0_rof,snr1_rof,cpt_rof] = tv_rof_m(f,0.25,19,25,0.01,67);
% for boat512:
% [u_rof,fn_rof,snr0_rof,snr1_rof,cpt_rof] = tv_rof_m(f,0.25,19,25,0.006,70);
disp('SNR for ROF model:')
[snr0_rof snr1_rof]
disp('CPU time for ROF model:')
cpt_rof
w1 = 128*ones(M,10);
w2 = 128*ones(10,2*N+10);
R = [f w1 fn; w2; u_rof w1 u];
figure(1)
imshow(R,map)
```

Example 3.17

```
[u,snr1,snr1_rof,snr0,cpt] = CLMC_diffusion(circles256,5,19,25,0.525,150,1,2);
```

for [u,snr1,snr1_rof,snr0,cpt] = CLMC_diffusion(f,dt,st,sg_n,sg_h,iter,K,ver)

Input image: circles256
Diffusion: CLMC model with a fractional edge detector
Standard deviation: 25
randn state: 19
τ = 5
value of σ for the Gaussian kernel: 0.525
value of *K*: 1
number of iterations: 150
PSNR of the noisy image: 14.2072 dB
PSNR of the restored image: 30.9260 dB
CPU time: 20.4219 s

For comparison purposes, The ROF algorithm with a constant λ implemented by function tv_rof_m.m was also applied to the same noisy image as given above, and the results obtained are as follows:

[u_rof,fn_rof,snr0_rof,snr1_rof,cpt_rof] = tv_rof_m(circles256,0.25,19,25,0.02,140);
Input image: circles256
Standard deviation: 25
randn state: 19
τ = 0.5
value of λ : 0.02
number of iterations: 140
PSNR of the noisy image: 14.2072 dB
PSNR of the restored image: 29.8583 dB
CPU time: 19.2969 s

The restored images are depicted in Figure 3. 16.

Example 3.18
[u,snr1,snr1_rof,snr0,cpt] = CLMC_diffusion(camera256,0.6,19,25,0.65,40,3,2);
for [u,snr1,snr1_rof,snr0,cpt] = CLMC_diffusion(f,dt,st,sg_n,sg_h,iter,K,ver)
Input image: camera256
Diffusion: CLMC model with a fractional edge detector
Standard deviation: 25
randn state: 19
τ = 0.6
value of σ for the Gaussian kernel: 0.65
value of *K*: 3
number of iterations: 40
PSNR of the noisy image: 14.5896 dB
PSNR of the restored image: 22.2885 dB
CPU time: 5.4844 s

For comparison purposes, The ROF algorithm with a constant λ implemented by function tv_rof_m.m was also applied to the same noisy image as given above, and the results obtained are as follows:

[u_rof,fn_rof,snr0_rof,snr1_rof,cpt_rof] = tv_rof_m(camera256,0.25,19,25,0.01,67);
Input image: camera256
Standard deviation: 25

randn state: 19
τ = 0.25
value of λ : 0.01
number of iterations: 67
PSNR of the noisy image: 14.5896 dB
PSNR of the restored image: 22.2069 dB
CPU time: 9.2656 s

The restored images are depicted in Figure 3. 17.

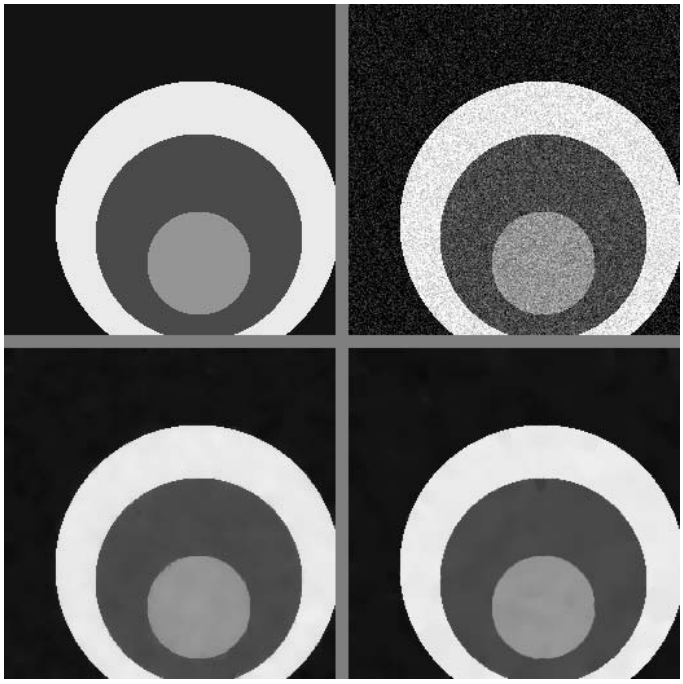
Example 3.19
[u,snr1,snr1_rof,snr0,cpt] = CLMC_diffusion(boat512,0.5,19,25,0.7,32,4,2);
for [u,snr1,snr1_rof,snr0,cpt] = CLMC_diffusion(f,dt,st,sg_n,sg_h,iter,K,ver)
Input image: boat512
Diffusion: CLMC model with a fractional edge detector
Standard deviation: 25
randn state: 19
τ = 0.5
value of σ for the Gaussian kernel: 0.7
value of *K*: 4
number of iterations: 32
PSNR of the noisy image: 15.3185 dB
PSNR of the restored image: 24.2796 dB
CPU time: 15.4375 s

For comparison purposes, The ROF algorithm with a constant λ implemented by function tv_rof_m.m was also applied to the same noisy image as given above, and the results obtained are as follows:

[u_rof,fn_rof,snr0_rof,snr1_rof,cpt_rof] = tv_rof_m(f,0.25,19,25,0.006,70);
Input image: boat512
Standard deviation: 25
randn state: 19
τ = 0.25
value of λ : 0.006
number of iterations: 70
PSNR of the noisy image: 15.3185 dB
PSNR of the restored image: 24.0456 dB
CPU time: 38.0313 s

The restored images are depicted in Figure 3. 18.

(a) (b)



(c) (d)

Figure 3.16 (a) Original circles256; (b) noisy image; (c) denoised by ROF; (d) denoised by CLMC model.

(a) (b)

(c) (d)

Figure 3.17 (a) Original camera256; (b) noisy image; (c) denoised by ROF; (d) denoised by CLMC model.

(a) (b)

(c) (d)

Figure 3.18 (a) Original boat512; (b) noisy image; (c) denoised by ROF; (d) denoised by CLMC model.

Problems

3.1 Derive formula (3.16).
3.2 Prepare two separate sets of MATLAB codes for the denoising algorithms highlighted in (3.23) and (3.24) respectively.
3.3 Use the codes from Problem 3.2 to run Examples 3.4, 3.5, and 3.6.
3.4 Derive formula (3.33).
3.5 (a) Write formulas that can be used to perform discrete implementation of the deblurring algorithm highlighted in (3.32), (3.33).
(a) Prepare MATLAB code for implementing the formulas in part (a).
(b) Apply the code in part (b) to de-blurring the images in Figs. 3.5(b), 3.6(b), 3.7(d), and 3.8(d).
3.6 Prove formulas in (3.36).
3.7 Show that function $\varphi(s)$ in (3.44) satisfies assumptions A1 and A2.
3.8 Verify formula (3.63).
3.9 Verify Steps 1, 2, and 3 of the Thomas algorithm.
3.10 Develop an AOS-based algorithm for image deblurring.
3.11 Apply the AOS-based algorithm obtained in Problem (3.10) to images in Figs. 3.5(b), 3.6(b), 3.7(d), and 3.8(d).
3.12 Prove that the functional $J_0(u)$ in (3.11) is convex.
3.13 Prove that the functional $J(u)$ in (3.71) is convex.
3.14 Generalize Chambolle's projection algorithm for image deblurring.

Chapter 4 Image Segmentation

4.1 Mumford-Shah Functional

4.1.1 *Introduction*

The goal of image segmentation is to find a visually meaningful edge set that leads to a complete partition of a given image into its constituent parts. As such, image segmentation serves as a bridge linking low-level image processing (such as edge detection) and high-level vision (such as object detection, recognition, and tracking). The main technical difficulty is that one needs to manipulate objects of different kinds, such as functions and curves.

In what follows we introduce some representative segmentation methods that are based on the concept of *active contours* (nicknamed *snakes*) and involve the minimization of the *Mumford-Shah functional* via *level-set methods*.

4.1.2 *The Mumford-Shah Functional*

A well-known image segmentation method was proposed by D. Mumford and J. Shah [16] in 1989. The Mumford-Shah segmentation problem can be formulated as follows: *Given an observed image u_0 defined in region Ω, find a decomposition $\{\Omega_i\}$ of Ω and an optimal piecewise smooth approximation of u_0, denoted by u, such that u is smooth within each connected component Ω_i, and varies rapidly or discontinuously across the boundaries of Ω_i.*

To solve the problem, Mumford and Shah [16] proposes to minimize the following functional which is now known as the *Mumford-Shah* (MS) *functional*:

$$J_{MS}(u,\Gamma) = \int_{\Omega} |u_0 - u|^2 \, dx\,dy + \mu \cdot \int_{\Omega\setminus\Gamma} |\nabla u|^2 \, dx\,dy + \nu \cdot |\Gamma| \qquad (4.1)$$

where μ and ν are weights, and $|\Gamma|$ denotes the length of curve Γ. The MS model appropriately combines the effects of edge set Γ and its segmented regions $\{\Omega_i, i = 1,2,...\}$. Technically, however, the MS functional is difficult to minimize because the boundaries of the sub-regions, Γ, and function *u* are all unknown, and it is *non-convex*.

Example 4.1 We illustrate how minimizing the MS functional leads to a solution for segmentation problems by considering a special case where function $u(x, y)$ is *piecewise-constant*, i.e., u = constant a_i inside each connected sub-region Ω_i. The MS functional in (4.1) is then reduced to

$$J_{MS}(u,\Gamma) = \sum_i \int_{\Omega_i} (u_0 - a_i)^2 \, dx\,dy + \nu\,|\Gamma| \qquad (4.2)$$

Note that for a fixed curve set Γ (hence regions $\{\Omega_i\}$), each constant a_i in F_{MS} appears only in *one* term:

$$\int_{\Omega_i} (u_0 - a_i)^2 \, dx\,dy = |\Omega_i| \, a_i^2 - 2a_i \int_{\Omega_i} u_0 \, dx\,dy + \int_{\Omega_i} u_0^2 \, dx\,dy$$

whose minimum is reached when a_i assumes the value

$$a_i = \frac{1}{|\Omega_i|}\int_{\Omega_i} u_0\, dx\, dy = \text{mean}(u_0) \quad \text{in} \quad \Omega_i$$

Below we consider a synthetic image with one object of interest as shown in Figure 4.1. Using the MS model, we seek to find a closed curve Γ that partitions the entire region Ω into two sub-regions:

$$\Omega_{\text{int}} = \text{inside}(\Gamma) \text{ and } \Omega_{\text{ext}} = \text{outside}(\Gamma)$$

and the MS functional in this case becomes

$$J_{MS}(u,\Gamma) = \int_{\Omega_{\text{int}}} (u_0 - a_1)^2\, dx\, dy + \int_{\Omega_{\text{ext}}} (u_0 - a_2)^2\, dx\, dy + \nu\,|\Gamma|$$

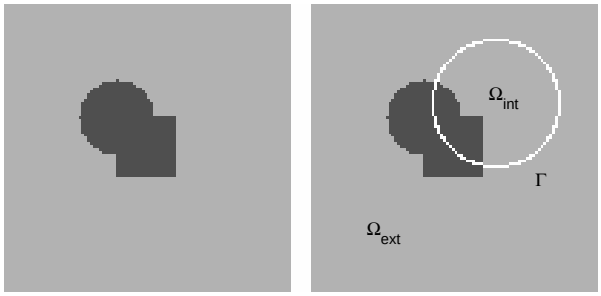


Figure 4.1

We now examine four cases. For simplicity, we assume weight ν is zero thus the MS functional is reduced to

$$J_{MS}(u,\Gamma) = \int_{\Omega_{\text{int}}} (u_0 - a_1)^2\, dx\, dy + \int_{\Omega_{\text{ext}}} (u_0 - a_2)^2\, dx\, dy$$

where u_0 denotes 8-bit light intensity of the image, it was found that $u_0 = 80$ for pixels of the object body and $u_0 = 180$ for pixels in the background.

Case 1. Curve Γ is outsize of the object, see Fig. 4.2a. Recall that $a_{\text{int}} = \text{mean}(u_0)$ in Ω_{int}. It was found that in this case $a_{\text{int}} = 158$. We also evaluated $a_{\text{ext}} = \text{mean}(u_0)$ in $\Omega_{\text{ext}} = 180$. This leads to

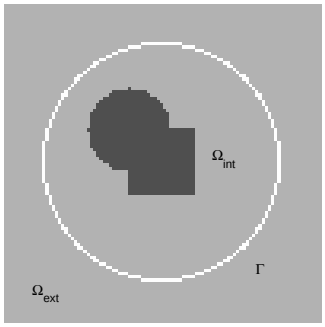


Figure 4.2 (a)

$$J_{MS} = \int_{\Omega_{\text{int}}} (u_0 - 158)^2 \, dx\, dy >> 0$$

Case 2. Curve Γ is inside of the object. In this case it was found that $a_{\text{int}} = \text{mean}(u_0)$ in $\Omega_{\text{int}} = 80$, and $a_{\text{ext}} = \text{mean}(u_0)$ in $\Omega_{\text{ext}} = 174$. Hence

$$J_{MS} = \int_{\Omega_{\text{ext}}} (u_0 - 174)^2 \, dx\, dy >> 0$$

See Figure 4.2b.

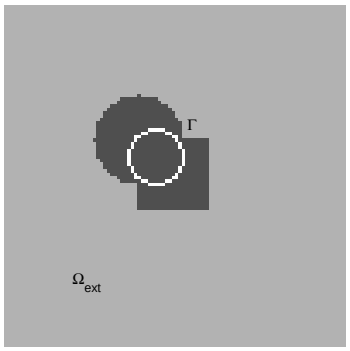


Figure 4.2 (b)

Case 3. Curve Γ intersects with both the object and background. We have $a_{\text{nt}} = \text{mean}(u_0)$ in $\Omega_{\text{int}} = 121$, and $a_{\text{ext}} = \text{mean}(u_0)$ in $\Omega_{\text{ext}} = 166$. Hence

$$J_{MS} = \int_{\Omega_{\text{int}}} (u_0 - 121)^2 \, dx\, dy + \int_{\Omega_{\text{ext}}} (u_0 - 166)^2 \, dx\, dy >> 0$$

See Figure 4.2c.

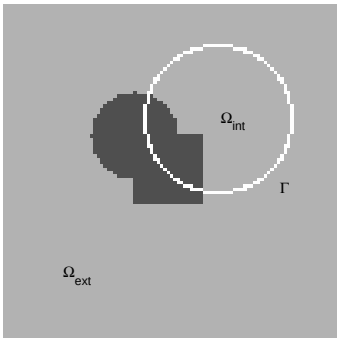


Figure 4.2 (c)

Case 4. Curve Γ coincides with the boundary of the object. In this case we have $a_{\text{int}} = \text{mean}(u_0)$ in $\Omega_{\text{int}} = 80$, and $a_{\text{ext}} = \text{mean}(u_0)$ in $\Omega_{\text{ext}} = 180$. This gives

$$\int_{\Omega_{\text{int}}} (u_0 - 80)^2 \, dx\, dy = 0 \quad \text{and} \quad \int_{\Omega_{\text{ext}}} (u_0 - 180)^2 \, dx\, dy = 0$$

$$J_{MS}(u, \Gamma) = \int_{\Omega_{\text{int}}} (u_0 - a_1)^2 \, dx\, dy + \int_{\Omega_{\text{ext}}} (u_0 - a_2)^2 \, dx\, dy = 0$$

See Figure 4.2d. we see that the curve obtained in this case is the unique global minimizer of the MS functional.

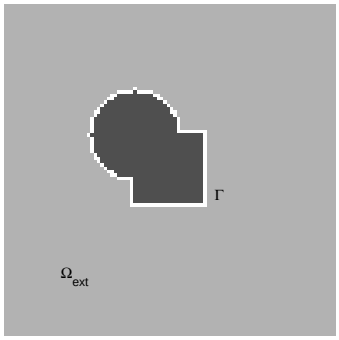


Figure 4.2 (d)

■

4.2 Geodesic Active Contours and the Level-Set Method

4.2.1 *The Kass-Witkin-Terzopoulos Model*

The basic idea in active contour models is to evolve a curve so that it eventually stops along the object edges of a given image $u_0(x, y)$. Let $\Gamma = \bigcup_i c_i$ be a set of curves with each c_i a piecewise smooth closed curve in R^2 which is parameterized as a differentiable function $c_i(q)$ with q varying from a to b and $c_i(a) = c_i(b)$. In a typical active contour model, one introduces an *edge detection function* (EDF) $g(s)$ which, when variable s is set to a scaled magnitude of the gradient of an image i.e. $s = |\nabla u_0(x, y)| / k$ with k a positive constant, characterizes image edges by *zero* values rather than *infinite* values. In general, an edge detection function possesses several general properties like this:

(i) $g(s)$ is monotonically decreasing
(ii) $g(0) = 1$, and $\lim_{s\to\infty} g(s) = 0$

It can be easily verified that the function

$$g(s) = \frac{1}{1+s^2} \tag{4.3}$$

is qualified as an edge detection function. Figure 4.3b shows the effect of applying the edge detection function in (4.3) to the image of size 360 × 460 in Fig. 4.3a.

Listed below is the MATLAB code that produces Fig. 4.3, where we set $h = 1$ and $k = 30$. Since the EDF has a range between 0 and 1, the edge map in Fig. 4.3b is produced by multiplying $g(s)$ by 255.

```
% Examples:
% g = eval_g(objects,30);
% g = eval_g(building256,50);
% g = eval_g(goldhill512,20);
function g = eval_g(u,k)
```

```
[M,N] = size(u);
k2 = k^2;
uw = ex_sym1(u);
uwa = uw(2:(M+1),3:(N+2));
uwb = uw(3:(M+2),2:(N+1));
gp1 = uwa - u;
gp2 = uwb - u;
s2 = (gp1.^2 + gp2.^2)/k2;
g = 1./(1 + s2);
gn = 255*g;
map = gray(255);
figure(1)
subplot(121)
imshow(u,map)
title('Original image')
xlabel('(a)')
subplot(122)
imshow(gn,map)
title('Output of the EDF (4.3) with k = 30')
xlabel('(b)')
```

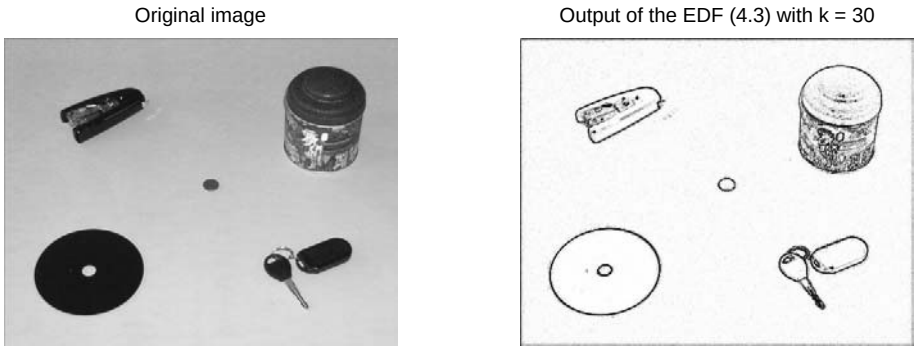


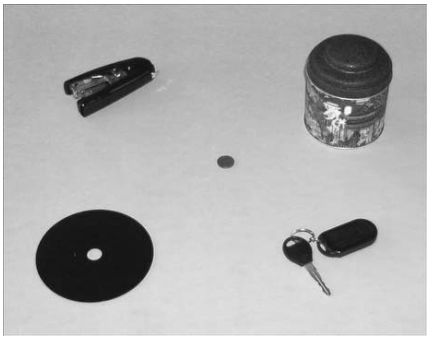
Figure 4.3

Another popular edge detection function also assume the form of (4.3), but the variable *s* there is now set to $s = |G_\sigma * \nabla u_0| / k$ where G_σ is a Gaussian kernel given by

$$G_\sigma(x,y) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{(x^2+y^2)}{4\sigma}} \qquad (4.4)$$

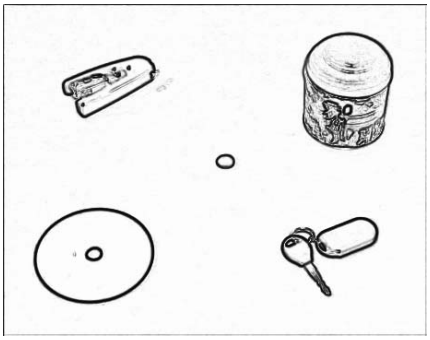
and * denotes the convolution of G_σ with each component of the gradient. The introduction of G_σ in EDF is the same as performing lowpass filtering of the image gradient which helps deal with noisy images. On the other hand it also blurs an image. For this reason the value of σ in (4.4) needs to be selected carefully. Fig. 4.4b depicts the result of applying such an edge detection function to the image in Fig. 4.4a with $\sigma = 0.8$ and $k = 15$.

Original image



(a)

Effect of Gaussian EDF with σ = 0.8 and k = 15



(b)

Figure 4.4

One of the earliest efforts in the direction of segmenting images by active contours is the use of the snake model proposed by Kass, Witkin and Terzopoulos [14] where the following functional is minimized:

$$J(c) = \int_a^b |c'(q)|^2 \, dq + \beta \int_a^b |c''(q)|^2 \, dq + \lambda \int_a^b g^2 \left(|\nabla u_0 (c(q))| \right) dq \qquad (4.5)$$

The function $c(q)$ in (4.5) is a parametric expression for a closed smooth curse on the plane region with $c(q) = [c_1(q) \;\; c_2(q)]$, so $|c'(q)| = \sqrt{\left(dc_1/dq\right)^2 + \left(dc_2/dq\right)^2}$ and $|c''(q)|$ is similarly defined. The first two terms on the right-hand side of (4.5) are called internal energy terms that impose smoothness of the curve c. The third term is called the external energy term that, when being minimized, attracts the curve toward the edges of the objects. In the above model, the second term (also called the elasticity term) is included to minimize the squared curvature. It turns out that minimizing $J(c)$ with $\beta = 0$ also decreases the curvature, hence it is rather natural to consider a simplified functional

$$J_{KWT}(c) = \int_a^b |c'(q)|^2 \, dq + \lambda \int_a^b g^2 \left(|\nabla u_0 (c(q))| \right) dq \qquad (4.6)$$

The functional in (4.6) is still not satisfactory to handle for several reasons. First, it is not intrinsic since it depends on the parameterization of curse c and one may obtain different solutions by changing the parameterization. Second, because of the regularity constraint, the model does not handle changes of topology, i.e., it is impossible to detect more than one object and the object has to be convex. Third, it is difficult to chose a set of marker points for discretizing a parameterized curve which evolving with time as the position of the marker points have to be updated in time according to the approximation of the time-embedded Euler-Lagrange equation.

These difficulties have been largely overcome in the geodesic active contours model proposed by Caselles, Kimmel, and Sapiro [15].

4.2.2 *A Geodesic Active Contours Model*

In [15], Caselles, Kimmel, and Sapiro proposed an edge detector of type

$$g(|\nabla u_0|) = \frac{1}{1 + (|G_\sigma * \nabla u_0| / k)^p} \quad \text{with} \;\; p \geq 1 \qquad (4.7)$$

and investigated the problem of minimizing the following functional

$$J_{CKS}(c) = \int_a^b g(|\nabla u_0(c(q))|)\,|c'(q)|\,dq \tag{4.8}$$

Note that with $s = |G_\sigma * \nabla u_0|/k$ and $p = 2$ the EDF in (4.7) becomes that in (4.3). Also notice that the functional in (4.8) reaches its global minimum (of value zero) when curve $c(q)$ coincides with the contours of the objectives. Moreover, the functional in (4.8) is intrinsic in the sense that for a new parameterization of the curve via $q = \varphi(r), \varphi:[c,d] \to [a,b], \varphi' > 0$, we have

$$J_{CKS}(c) = \int_c^d g(|\nabla u_0(\bar{c}(r))|)\,|\bar{c}'(r)|\,dr$$

where $\bar{c}(r) = c(\varphi(r))$ meaning that there is no change in energy.

Let $c(t, q)$ be a family of curves where $t \geq 0$ is an exterior parameter (time) such that $c(0, q) = c(q)$. We examine how the functional in (4.8) changes with the evolving curve $c(t, q)$. To this end we compute dJ_{CKS}/dt as follows where, for the sake of simplicity, $g(s)$ assumes the form of (4.3) with $s = |\nabla u_0(x, y)|/k$.

$$\begin{aligned}
&\frac{dJ_{CKS}(c(t,q))}{dt} = \frac{d}{dt}\int_a^b g(|\nabla u_0(c(t,q))|)\,|c'(t,q)|\,dq \\
&= \int_a^b g \cdot \frac{d\sqrt{(c_1(t,q)')^2 + (c_2(t,q)')^2}}{dt}\,dq + \int_a^b \left|\frac{\partial c(t,q)}{\partial q}\right| \frac{dg(|\nabla u_0(c(t,q))|)}{dt}\,dq \\
&= \int_a^b g \cdot \left\langle \frac{\frac{\partial c(t,q)}{\partial q}}{\left|\frac{\partial c(t,q)}{\partial q}\right|},\ \frac{\partial^2 c(t,q)}{\partial t \partial q} \right\rangle dq + \int_a^b \left|\frac{\partial c(t,q)}{\partial q}\right| \cdot \left\langle \nabla g, \frac{\partial c(t,q)}{\partial t} \right\rangle dq
\end{aligned}$$

By using integration by parts for the first integral, we have

$$\frac{dJ_{CKS}(c(t,q))}{dt} = -\int_a^b \left\langle g \cdot \frac{\partial}{\partial q}\left(\frac{\frac{\partial c}{\partial q}}{\left|\frac{\partial c}{\partial q}\right|}\right) + \left\langle \nabla g, \frac{\partial c}{\partial q}\right\rangle \frac{\frac{\partial c}{\partial q}}{\left|\frac{\partial c}{\partial q}\right|},\ \frac{\partial c}{\partial t} \right\rangle dq + \int_a^b \left|\frac{\partial c}{\partial q}\right| \cdot \left\langle \nabla g, \frac{\partial c}{\partial t}\right\rangle dq$$

which leads to

$$\frac{dJ_{CKS}(c(t,q))}{dt} = \int_a^b \left|\frac{\partial c}{\partial q}\right| \left\langle \frac{\partial c}{\partial t},\ \nabla g - \frac{1}{\left|\frac{\partial c}{\partial q}\right|}\frac{\partial}{\partial q}\left(\frac{\frac{\partial c}{\partial q}}{\left|\frac{\partial c}{\partial q}\right|}\right) g - \frac{\frac{\partial c}{\partial q}}{\left|\frac{\partial c}{\partial q}\right|}\left\langle \nabla g, \frac{\frac{\partial c}{\partial q}}{\left|\frac{\partial c}{\partial q}\right|}\right\rangle \right\rangle dq$$

Now recalling the definitions of tangent T, normal N, and curvature κ (see Chapter 2, Sections 2.5.1 and 2.5.2), we obtain

$$\frac{dJ_{CKS}\left(c(t,q)\right)}{dt} = \int_a^b \left|\frac{\partial c}{\partial q}\right| \left\langle \frac{\partial c}{\partial t},\ \nabla g - \kappa g N - \left\langle \nabla g, T \right\rangle T \right\rangle dq$$

By using the decomposition $\nabla g = \left\langle \nabla g, T \right\rangle T + \left\langle \nabla g, N \right\rangle N$ in the above expression, we have

$$\frac{dJ_{CKS}\left(c(t,q)\right)}{dt} = \int_a^b \left|\frac{\partial c}{\partial q}\right| \left\langle \frac{\partial c}{\partial t},\ \left(\left\langle \nabla g, N \right\rangle - \kappa g\right) N \right\rangle dq \tag{4.9}$$

which implies that functional J_{CKS} decreases most rapidly when $c(t, q)$ satisfies the equation

$$\frac{\partial c}{\partial t} = \left(\kappa g - \left\langle \nabla g, N \right\rangle\right) N \tag{4.10}$$

Eq. (4.10) is known as the Caselles-Kimmel-Sapiro (CKS) model.

Remarks

(i) If $g \equiv 1$, then (4.10) becomes

$$\frac{\partial c}{\partial t} = \kappa N \tag{4.11}$$

which is the well-known mean curvature motion (also known as shortening flow). Because this flow decreases the total curvature as well as the number of zero crossings and the value of maxima/minima curvature, it has the properties of *shortening* (an initial curve shrinks to a point in finite time with asymptotically circular shape) and *smoothing* (points with large curvature evolve faster and disappear asymptotically).

(ii) The CKS model may be improved by adding a supplementary tem to the right-hand side of (4.10):

$$\frac{\partial c}{\partial t} = \left(\kappa g - \left\langle \nabla g, N \right\rangle + \alpha g\right) N$$

with $\alpha \geq 0$. The additional term just modifies the coefficient of the first term from κ to $\kappa + \alpha$:

$$\frac{\partial c}{\partial t} = \left((\kappa + \alpha)\, g - \left\langle \nabla g, N \right\rangle\right) N \tag{4.12}$$

If the value of α is chosen to be sufficiently positive, then the modified curvature shall remain of constant and positive sign. In other words, the curvature κ itself is allowed to have non-constant sign, thus *nonconvex objects* can be detected.

4.2.3 *The Level-Set Method*

4.2.3.1 *Level-Set Functions*

The level-set method was proposed by Osher and Sethian [18] in a seminal paper in 1988. The method has since been studied by many researchers and the method is now found useful in many problems, see the recent book by Osher and Fedkiw [5]. We introduce the method here mainly for solving Eq. (4.12) as well as other PDEs derived from other variational models for segmentation and other problems in image processing. The level-set approach is computationally superior to other curve representations because it lets one to directly work on a fixed rectangular grid and allows automatic topological changes such as merging and breaking.

Let Γ be the boundary of an open subset S in region Ω. In the level-set method Γ is represented by the *zero level set* of a smooth real-valued function $\varphi(x, y)$ (called level-set function) such that

$$\begin{cases} \Gamma = \partial S = \{(x, y) \in \Omega : \varphi(x, y) = 0\} \\ \text{inside}(\Gamma) = S = \{(x, y) \in \Omega : \varphi(x, y) > 0\} \\ \text{outside}(\Gamma) = \Omega \setminus \overline{S} = \{(x, y) \in \Omega : \varphi(x, y) < 0\} \end{cases} \tag{4.13}$$

In a sense, a level-set function is like a water buffalo in river (Fig. 4.5).

Figure 4.5

For example, curve Γ in the right plot of Fig. 4.6 (of size 120 × 120) can be described by the level-set function $\varphi(x, y) = 37 - \sqrt{(x-50)^2 + (y-50)^2}$ because $\Gamma = \{(x, y) \in \Omega : \varphi(x, y) = 0\}$ satisfies

$$\text{inside}(\Gamma) = \{(x, y) \in \Omega : \varphi(x, y) > 0\}$$
$$\text{outside}(\Gamma) = \{(x, y) \in \Omega : \varphi(x, y) < 0\}$$

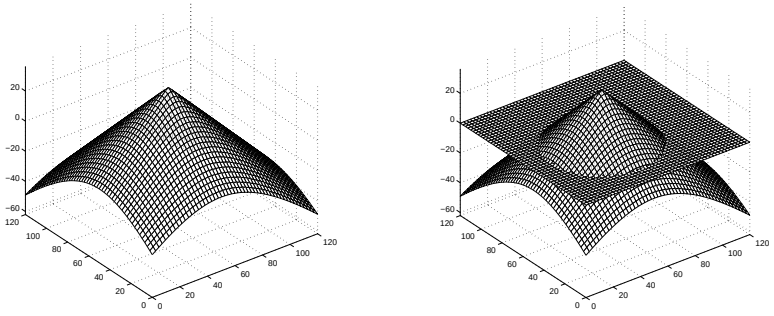


Figure 4.6

4.2.3.2 *The Level-Set Method for Image Segmentation*

For the purpose of segmentation by active contours, we are interested in solving the following type of equations:

$$\begin{cases} \dfrac{\partial c}{\partial t} = FN \\ c(0,q) = c_0(q) \end{cases} \tag{4.14}$$

where the time-varying curve Γ is described by a family of smooth vector-valued functions $c(t, q) = (c_1(t, q), c_2(t, q))$, and $c_0(q)$ is the initial curve from which the curve evolution stars, see for example (4.10), (4.11) and (4.12). As might be expected, the level-set functions in segmentation applications are time-dependent., the level-set function assumes the form $\varphi(t, x, y)$ for $t \geq 0$ and all (x, y), and satisfies

$$\varphi(t, c(t,q)) = \varphi(t, c_1(t,q), c_2(t,q)) = 0 \quad for\ \forall q,\ \forall t \geq 0 \tag{4.15}$$

Differentiating the above expression with respect to time t yields

$$\frac{\partial \varphi}{\partial t} + \frac{\partial \varphi}{\partial x}\frac{\partial c_1}{\partial t} + \frac{\partial \varphi}{\partial y}\frac{\partial c_2}{\partial t} = \frac{\partial \varphi}{\partial t} + \left\langle \nabla \varphi, \frac{\partial c}{\partial t} \right\rangle = 0 \tag{4.16}$$

By replacing the evolution speed in (4.16) using (4.14), we obtain

$$\frac{\partial \varphi}{\partial t} + \langle \nabla \varphi, FN \rangle = 0 \tag{4.17}$$

By (4.13), the unit *inward* normal of the curve is given by $\nabla\varphi / |\nabla\varphi|$, hence

$$\langle \nabla \varphi, FN \rangle = \langle \nabla \varphi, F \nabla \varphi / |\nabla \varphi| \rangle = F\,|\nabla \varphi|$$

which leads (4.17) to

$$\frac{\partial \varphi(t, c(t,q))}{\partial t} = -F\,|\nabla \varphi(t, c(t,q))| \tag{4.18}$$

The above equation is supposed to be valid only for the zero level-set of $\varphi(t, x, y)$. One of the advantages of the level set method is that function $\varphi(t, x, y)$ in (4.18) may be regarded as that defined in the entire domain $R^+ \times \Omega$. This gives

$$\frac{\partial \varphi(t, x, y)}{\partial t} = -F\,|\nabla \varphi(t, x, y)| \quad \text{for } t \geq 0, (x, y) \in \Omega \tag{4.19a}$$

Once the solution $\varphi(t, x, y)$ of (4.19) is computed, then curve $c(t, q)$ can be obtained as the zero level-set of $\varphi(t, x, y)$, see (4.15). Since (4.19) is defined in the entire domain Ω, a Neumann boundary condition is imposed as

$$\frac{\partial \varphi}{\partial N} = 0 \quad \text{on } \partial\Omega \tag{4.19b}$$

At time $t = 0$, the initial curve $c_0(q)$ needs to be expressed as the zero level-set of a smooth function, and that function will then be taken as the initial condition of $\varphi(t, x, y)$ at $t = 0$. A good candidate for this is the so-called *signed distance function* which is defined by

$$\varphi(0, x, y) = \bar{d}(x, y, c_0) = \begin{cases} -d(x, y, c_0) & \text{if } (x, y) \text{ is outside } c_0 \\ d(x, y, c_0) & \text{if } (x, y) \text{ is inside } c_0 \end{cases} \tag{4.19c}$$

where $d(x, y, c_0)$ denotes the Euclidean distance from (x, y) to c_0. The equation in (4.19a) is called a *Hamilton-Jacobi* equation.

Working with a level-set function offers several advantages compared working with equations of parameterized curves, as remarked below.

Remarks

(i) Equations for level-set functions take into account the topology changes such as merge or break automatically because an evolving smooth level-set function can have topologically arbitrarily different zero level-sets.
(ii) Numerical methods for level-set associated equations can use a fixed discrete grid, where finite-difference approximations are suitable for spatial as well as temporal derivatives.
(iii) In a level-set method, intrinsic geometric quantities of the front (i.e. object boundary) such as its normal and curvature can be readily expressed in terms of the level-set function $\varphi(t, x, y)$.
(iv) High-dimensional extension of level-set based methods is possible.

For image segmentation, the term F in (4.19a) is given in (4.12) as $F = (\kappa + \alpha)\, g - \langle \nabla g,\, N \rangle$, thus (4.19) assumes the form

$$\frac{\partial \varphi(t, x, y)}{\partial t} = \left(\langle \nabla g, \nabla \varphi / |\nabla \varphi| \rangle - (\kappa + \alpha)\, g \right) |\nabla \varphi(t, x, y)|$$

Since curvature $\kappa = -\text{div}\left(\dfrac{\nabla \varphi}{|\nabla \varphi|} \right)$ (see Eq. (2.55) in Chapter 2), the above equation can be written as

$$\frac{\partial \varphi}{\partial t} = g(|\nabla u|) \left(\text{div}\left(\frac{\nabla \varphi}{|\nabla \varphi|} \right) - \alpha \right) |\nabla \varphi| + \langle \nabla g, \nabla \varphi \rangle \tag{4.20}$$

with the same boundary and initial conditions as in (4.19b) and (4.19c). It is obvious that the first term on the right-hand side of (4.20) allows one to stop the evolving curve when it arrives at the object boundaries. It is however less intuitive that second term increases the attraction of the deforming contour towards the object boundaries, see [1, Sec. 4.3.3] for an illustrative example for the role the second term plays.

Remarks

(i) Using the fact that

$$\operatorname{div}\left(g \cdot \frac{\nabla \varphi}{|\nabla \varphi|} \right) = g \cdot \operatorname{div}\left(\frac{\nabla \varphi}{|\nabla \varphi|} \right) + \left\langle \nabla g, \nabla \varphi / |\nabla \varphi| \right\rangle$$

Eq. (4.20) may be expressed as

$$\frac{\partial \varphi}{\partial t} = |\nabla \varphi| \left(\operatorname{div}\left(g \cdot \frac{\nabla \varphi}{|\nabla \varphi|} \right) - \alpha g\left(|\nabla u|\right) \right) \tag{4.21}$$

(ii) In some literature, the level-set function is defined as a smooth real-valued function $\varphi(x, y)$ such that

$$\begin{cases} \Gamma = \partial S = \{(x, y) \in \Omega : \varphi(x, y) = 0\} \\ \text{inside}(\Gamma) = S = \{(x, y) \in \Omega : \varphi(x, y) < 0\} \\ \text{outside}(\Gamma) = \Omega \setminus \overline{S} = \{(x, y) \in \Omega : \varphi(x, y) > 0\} \end{cases} \tag{4.22}$$

Obviously if $\varphi(x, y)$ is a level-set function under the definition in (4.13), then $-\varphi(x, y)$ is a level-set function under the definition in (4.22). Consequently, the level-set based PDE according to (4.22) is given by

$$\frac{\partial \varphi}{\partial t} = g\left(|\nabla u|\right) \left(\operatorname{div}\left(\frac{\nabla \varphi}{|\nabla \varphi|} \right) + \alpha \right) |\nabla \varphi| + \left\langle \nabla g, \nabla \varphi \right\rangle \tag{4.23}$$

or alternatively

$$\frac{\partial \varphi}{\partial t} = |\nabla \varphi| \left(\operatorname{div}\left(g \cdot \frac{\nabla \varphi}{|\nabla \varphi|} \right) + \alpha g\left(|\nabla u|\right) \right) \tag{4.24}$$

Of course, the signed distance function in the initial condition (4.19c) needs to be modified accordingly as

$$\varphi(0, x, y) = \overline{d}(x, y, c_0) = \begin{cases} d(x, y, c_0) & \text{if } (x, y) \text{ is outside } c_0 \\ -d(x, y, c_0) & \text{if } (x, y) \text{ is inside } c_0 \end{cases} \tag{4.25}$$

4.2.3.3 Implementation Issues

Re-Initialization

An important issue to deal with for the implementation of a level-set-function based image segmentation is how to construct an adequate initial level-set function $\varphi(0, x, y)$. Furthermore, as the contour (also known as *interface* or *front* [5]) evolves, the level-set function $\varphi(t, x, y)$ will drift away from its initial value as signed distance. From a computational perspective, this drift means the magnitude of the gradient gets smaller and smaller (note that the magnitude of the gradient of a signed distance function is equal to

one), i.e. the level-set function gets flatter and flatter and eventually disappears. For example [1], consider a simplified case of Eq. (4.20) with $g \equiv 1$ and $\alpha = 0$:

$$\frac{\partial \varphi}{\partial t} = |\nabla \varphi| \operatorname{div}\left(\frac{\nabla \varphi}{|\nabla \varphi|}\right) \tag{4.26}$$

(which describes *mean curvature motion* [1]) with the unit circle centered at the origin as its initial condition. The initial level-set function is given by the signed distance function $\varphi(0, x, y) = 1 - \sqrt{x^2 + y^2}$. It can be easily verified that $\varphi(t, x, y) = 1 - \sqrt{x^2 + y^2 + 2t}$ solves (4.26) and satisfies the above initial condition. This solution gives

$$\nabla \varphi = \frac{-1}{\sqrt{x^2 + y^2 + 2t}} \begin{bmatrix} x \\ y \end{bmatrix}, \quad |\nabla \varphi| = \frac{\sqrt{x^2 + y^2}}{\sqrt{x^2 + y^2 + 2t}}$$

$$\frac{\nabla \varphi}{|\nabla \varphi|} = \frac{-1}{\sqrt{x^2 + y^2}} \begin{bmatrix} x \\ y \end{bmatrix}, \qquad \operatorname{div}\left(\frac{\nabla \varphi}{|\nabla \varphi|}\right) = \frac{-1}{\sqrt{x^2 + y^2}}$$

The last two quantities are not defined at the origin, so practically a spike occurs at the origin. Moreover the zero-level of $\varphi(t, x, y) = 1 - \sqrt{x^2 + y^2 + 2t}$ is the circle centered at the origin with radius $1 - 2t$ so at the front we have $|\nabla \varphi| = \sqrt{1 - 2t}$ -- we see that as the front evolves the magnitude of the gradient gets smaller and the front disappears at $t = ½$.

To prevent the front from drifting and disappearing, one needs to re-initiate the level-set function sufficiently frequently so that the level-set function maintains remains to be approximately a signed distance function whose zero-level is the current front. Chapter 7 of reference [5] presents several methods for constructing signed distance functions and in-initialization of a level-set function. Some of these methods are sketched below.

Let $\varphi_0(x, y)$ be a level-set function whose zero-level defines a closed curve Γ_0, and $S(\varphi_0)$ be a function in R^2 that assumes value 1 at point (x, y) if $\varphi_0(x, y) > 0$, assumes value -1 at (x, y) if $\varphi_0(x, y) < 0$, and assumes value 0 at (x, y) if $\varphi_0(x, y) = 0$. One of the re-initialization equations is given by

$$\begin{aligned} &\frac{\partial \varphi}{\partial t} + S(\varphi_0)\left(|\nabla \varphi| - 1\right) = 0 \\ &\varphi(0, x, y) = \varphi_0(x, y) \end{aligned} \tag{4.27}$$

In discretizing (4.27), numerical tests indicate that better results are obtained when $S(\varphi_0)$ is smeared out as

$$S(\varphi_0) = \frac{\varphi_0}{\sqrt{\varphi_0^2 + (\Delta x)^2}} \tag{4.28}$$

When the initial $S(\varphi_0)$ is a poor estimate of signed distance function, namely when $|\nabla\varphi_0|$ is far from 1, a choice better than (4.28) is given by

$$S(\varphi)=\frac{\varphi}{\sqrt{\varphi^2+|\nabla\varphi|^2\,(\Delta x)^2}} \tag{4.29}$$

Difference Schemes

(a) The Main Equation If we write (4.20) as

$$\frac{\partial\varphi}{\partial t}=g(|\nabla u|)\,|\nabla\varphi|\,\mathrm{div}\left(\frac{\nabla\varphi}{|\nabla\varphi|}\right)-\alpha\cdot g(|\nabla u|)\,|\nabla\varphi|+\langle\nabla g,\nabla\varphi\rangle \tag{4.30}$$

then the first term (on the right-hand side) is a parabolic term while the last two terms are hyperbolic terms, hence for the discretization of the equation we need to treat different types of terms differently (See. Sections 2.6 and 2.7). A difference scheme of (4.30) is given by [1] as

$$\begin{aligned}\varphi_{i,j}^{n+1}&=\varphi_{i,j}^{n}+\tau\Big[g_{i,j}^{n}K_{i,j}^{n}[(\delta_x\varphi_{i,j}^{n})^2+(\delta_y\varphi_{i,j}^{n})^2]^{1/2}\\&-\alpha\Big[\max(g_{i,j}^{n},0)\nabla^{+}+\min(g_{i,j}^{n},0)\nabla^{-}\Big]\varphi_{i,j}^{n}\\&+\max((g_x)_{i,j}^{n},0)\delta_x^{-}\varphi_{i,j}^{n}+\min((g_x)_{i,j}^{n},0)\varphi_x^{+}u_{i,j}^{n}\\&+\max((g_y)_{i,j}^{n},0)\delta_y^{-}\varphi_{i,j}^{n}+\min((g_y)_{i,j}^{n},0)\delta_y^{+}\varphi_{i,j}^{n}\Big]\end{aligned} \tag{4.31a}$$

where

$$K=\mathrm{div}\left(\frac{\nabla\varphi}{|\nabla\varphi|}\right)=\frac{\varphi_{xx}\varphi_y^2-2\varphi_x\varphi_y\varphi_{xy}+\varphi_{yy}\varphi_x^2}{\left(\varphi_x^2+\varphi_y^2\right)^{3/2}} \tag{4.31b}$$

$$\nabla^{+}\varphi_{i,j}^{n}=\Big[\max(\delta_x^{-}\varphi_{i,j}^{n},0)^2+\min(\delta_x^{+}\varphi_{i,j}^{n},0)^2+\max(\delta_y^{-}\varphi_{i,j}^{n},0)^2+\min(\delta_y^{+}\varphi_{i,j}^{n},0)^2\Big]^{1/2} \tag{4.31c}$$

$$\nabla^{-}\varphi_{i,j}^{n}=\Big[\max(\delta_x^{+}\varphi_{i,j}^{n},0)^2+\min(\delta_x^{-}\varphi_{i,j}^{n},0)^2+\max(\delta_y^{+}\varphi_{i,j}^{n},0)^2+\min(\delta_y^{-}\varphi_{i,j}^{n},0)^2\Big]^{1/2} \tag{4.31d}$$

and τ denotes the time step-size.

Now some details on the scheme in (4.31a). If follows from (4.30) and (4.31b) that if we denote

$$\bar{K}\equiv|\nabla\varphi|\,\mathrm{div}\left(\frac{\nabla\varphi}{|\nabla\varphi|}\right)=\frac{\varphi_{xx}\varphi_y^2-2\varphi_x\varphi_y\varphi_{xy}+\varphi_{yy}\varphi_x^2}{\varphi_x^2+\varphi_y^2} \tag{4.32}$$

then (4.31a) can be simplified to

$$\begin{aligned}\varphi_{i,j}^{n+1}&=\varphi_{i,j}^{n}+\tau\Big[g_{i,j}^{n}\bar{K}_{i,j}^{n}-\alpha\Big[\max(g_{i,j}^{n},0)\nabla^{+}+\min(g_{i,j}^{n},0)\nabla^{-}\Big]\varphi_{i,j}^{n}\\&+\max((g_x)_{i,j}^{n},0)\delta_x^{-}\varphi_{i,j}^{n}+\min((g_x)_{i,j}^{n},0)\varphi_x^{+}u_{i,j}^{n}\\&+\max((g_y)_{i,j}^{n},0)\delta_y^{-}\varphi_{i,j}^{n}+\min((g_y)_{i,j}^{n},0)\delta_y^{+}\varphi_{i,j}^{n}\Big]\end{aligned} \tag{4.33}$$

where $\bar{K}_{i,j}^n$ is obtained using (4.32) with central finite-difference approximation, and a small positive number ε is usually added to the denominator to prevent numerical difficulties. In addition we remark that the edge detection function g is determined entirely by the image data hence is *fixed* during the iteration process. Terms $\left(g_x\right)_{i,j}^n$ and $\left(g_y\right)_{i,j}^n$ can be obtained using central-difference approximation and they are also *fixed* during in the iterations.

Alternatively, one may use (4.21) as the main equation, a difference scheme of (4.21) is given by

$$\varphi_{i,j}^{n+1} = \varphi_{i,j}^n + \tau\left(|D\varphi_{i,j}^n| \left(\delta_x^- \left(g_{i,j}^n \frac{\delta_x^+ \varphi_{i,j}^n}{|D\varphi_{i,j}^n|} \right) + \delta_y^- \left(g_{i,j}^n \frac{\delta_y^+ \varphi_{i,j}^n}{|D\varphi_{i,j}^n|} \right) \right) - \alpha g_{i,j}^n \nabla_{i,j}^+ \varphi_{i,j}^n \right) \tag{4.34a}$$

where

$$|D\varphi_{i,j}^n| = \sqrt{\left(\delta_x^+ \varphi_{i,j}^n\right)^2 + \left(\delta_y^+ \varphi_{i,j}^n\right)^2 + \varepsilon} \tag{4.34b}$$

and $\nabla^+ \varphi_{i,j}^n$ is given by (4.13c).

(b) Equation (4.27) for Re-Initialization

We write (4.27) as

$$\frac{\partial \varphi}{\partial t} = -S(\varphi_0)|\nabla\varphi| + S(\varphi_0) \tag{4.35}$$

which suggests a difference scheme as

$$\varphi_{i,j}^{n+1} = \varphi_{i,j}^n - \tau\left(S\left((\varphi_0)_{i,j}\right)\nabla^+ \varphi_{i,j}^n + S\left((\varphi_0)_{i,j}\right)\right) \tag{4.36}$$

where $\nabla^+ \varphi_{i,j}^n$ is given by (4.13c).

4.2.3.4 MATLAB *Code and Examples*

MATLAB implementation of the CKS algorithm is performed by means of a main function cks_ak.m which requires two functions: re_initial.m and ex_sym2.m.

1. Function cks_ak.m

```
% To perform image segmentation by using the active contour method proposed
% in V. Caselles, R. Kimmel and G. Sapiro, "Geodesic active contours," Int.
% J. Computer Vision, vol. 22, no. 1, pp. 61-79, 1997.
% For the difference scheme employed here, please refer to Appendix A.3.4
% of the book: Gilles Aubert and Pierre Kornprobst, Mathematical Problems
% in Image Processing, Springer, New York, 2002.
% Inputs:   u -- image to be segmented
%         phi0 -- initial level-set function
%            dt -- step size in time
%           dt0 -- step size in time for re-initialization
```

```
%           a -- parameter alpha
%         sig -- parameter sigma in Gaussian kernel of the edge detection
%                function (EDF)
%           k -- parameter k in EDF
%       K_0 -- number of iterations for re-initialization
%      K_in -- number of iterations before next re-initialization
%                (b=number of internal iterations)
%    K_out -- number of internal iteration rounds
% Output: phi -- final level-set function
% Written by W.-S. Lu, University of Victoria.
% last modified: March 2, 2008.
% Example: phi = cks_ak(obj2,p02,0.1,0.5,5,1,6,7,20,10);
%          phi = cks_ak(obj3,p03,0.1,0.5,5,1.2,3,10,20,10);
function phi = cks_ak(u,phi0,dt,dt0,a,sig,k,K_0,K_in,K_out)

[M,N] = size(u);
yt = M:-1:1;
xt = 1:1:N;
k2 = k^2;
epsi = 1e-4;
v = -1e-6:5e-7:1e-6;
uw = ex_sym2(u);
uwa = uw(2:(M+1),3:(N+2));
uwb = uw(3:(M+2),2:(N+1));
gp1 = uwa - u;
gp1 = ex_sym2(gp1);
gp2 = uwb - u;
gp2 = ex_sym2(gp2);
H = fspecial('gaussian',7,sig);
gp1h = imfilter(gp1,H,'symmetric');
gp2h = imfilter(gp2,H,'symmetric');
s2 = (gp1h.^2 + gp2h.^2)/k2;
gex = 1./(1 + s2);         % expanded g
g = gex(2:(M+1),2:(N+1));  % g_i,j
ga = gex(2:(M+1),3:(N+2)); % g_i+1,j
gb = gex(3:(M+2),2:(N+1)); % g_i,j+1
gc = gex(2:(M+1),1:N);     % g_i-1,j
ge = gex(1:M,2:(N+1));     % g_i,j-1
gx = (ga - gc)/2;
gy = (gb - ge)/2;
gmax = max(g,0);
gmin = min(g,0);
gxmax = max(gx,0);
gxmin = min(gx,0);
gymax = max(gy,0);
gymin = min(gy,0);
map = gray(256);
figure(1)
imshow(u,map)
pw = phi0;
it = 0;
figure(2)
contour(xt,yt,pw,v)
axis square
title('Zero-level of initial level-set function')
pause(0.5)
while it < K_out,
```

```
    itt = 0;
    while itt < K_in,
       pww = ex_sym2(pw);
       pa = pww(2:(M+1),3:(N+2)); % phi_i+1,j
       pb = pww(3:(M+2),2:(N+1)); % phi_i,j+1
       pc = pww(2:(M+1),1:N);     % phi_i-1,j
       pd = pww(3:(M+2),1:N);     % phi_i-1,j+1
       pe = pww(1:M,2:(N+1));     % phi_i,j-1
       pf = pww(1:M,3:(N+2));     % phi_i+1,j-1
       pxx = pa - 2*pw + pc;
       pyy = pb - 2*pw + pe;
       pxy = 0.5*(pa -2*pw - pf + pe + pb - pd + pc);
       px = (pa - pc)/2;
       py = (pb - pe)/2;
       px2 = px.^2;
       py2 = py.^2;
       rr = px2 + py2 + epsi;
       Kw = pxx.*py2 + pyy.*px2 - 2*(px.*py).*pxy;
       Kav = Kw./rr;
       paw = pa - pw;
       pbw = pb - pw;
       pwc = pw - pc;
       pwe = pw - pe;
       gpp1 = max(pwc,0).^2;
       gpp2 = min(paw,0).^2;
       gpp3 = max(pwe,0).^2;
       gpp4 = min(pbw,0).^2;
       gpn1 = min(pwc,0).^2;
       gpn2 = max(paw,0).^2;
       gpn3 = min(pwe,0).^2;
       gpn4 = max(pbw,0).^2;
       gpm = sqrt(gpp1+gpp2+gpp3+gpp4);
       gnm = sqrt(gpn1+gpn2+gpn3+gpn4);
       T1 = g.*Kav;
       T2 = a*(gmax.*gpm + gmin.*gnm);
       T3 = gxmax.*pwc + gxmin.*paw;
       T4 = gymax.*pwe + gymin.*pbw;
       pw = pw + dt*(T1 - T2 + T3 + T4);
       itt = itt + 1;
    end
    pw_new = re_initial(pw,dt0,K_0);
    pw = pw_new;
    figure(2)
    contour(xt,yt,pw,v)
    axis square
    it = it + 1;
    title(sprintf('Zero-level after %.5g rounds of iterations',it'))
end
phi = pw;
```

2. Function re_initial.m

```
% Perform re-initialization of a given level-set function
% to produce a level-set function with the same zero-level
% set as the initial level-set function and non-vanishing
% gradient magnitude. For detaols refer to Chapter 7 of the
```

```
% book: S. Osher and R. Fedkiw, Level Set Methods and Dynamic
% Implicit Surfaces, Springer, New York, 2003.
% Written by W.-S. Lu, University of Victoria.
% Last modified: March 1, 2008.

function phi = re_initial(vn,dt,iter)

[M,N] = size(vn);
uw = vn;
uwe = ex_sym2(uw);
ua = uwe(2:(M+1),3:(N+2)); % u_i+1_j
ub = uwe(2:(M+1),1:N);     % u_i-1,j
uc = uwe(3:(M+2),2:(N+1)); % u_i_j+1
ud = uwe(1:M,2:(N+1));     % u_i_j-1
dxp = ua - uw;
dxn = uw - ub;
dyp = uc - uw;
dyn = uw - ud;
gp = max(dxn,0).^2 + min(dxp,0).^2 + max(dyn,0).^2 + min(dyp,0).^2;
S = vn./sqrt(vn.^2 + gp);
it = 0;
while it < iter,
   uwe = ex_sym2(uw);
   ua = uwe(2:(M+1),3:(N+2)); % u_i+1_j
   ub = uwe(2:(M+1),1:N);     % u_i-1,j
   uc = uwe(3:(M+2),2:(N+1)); % u_i_j+1
   ud = uwe(1:M,2:(N+1));     % u_i_j-1
   dxp = ua - uw;
   dxn = uw - ub;
   dyp = uc - uw;
   dyn = uw - ud;
   gp = sqrt(max(dxn,0).^2 + min(dxp,0).^2 + max(dyn,0).^2 + min(dyp,0).^2);
   gn = sqrt(max(dxp,0).^2 + min(dxn,0).^2 + max(dyp,0).^2 + min(dyn,0).^2);
   sp = max(S,0).*gp + min(S,0).*gn;
   uw_new = uw + dt*(S - sp);
   uw = uw_new;
   it = it + 1;
end
phi = uw;
```

3. Function ex_sym2.m

```
% To expand the input 2-D signal, uw, of size
% M x N to signal uwe of size (M+2) x (N+2) by
% symmetriical reflection so that the "main part"
% of the 1st column of uwe equals the 2nd column
% of uw, and the "main part" of the last column of
% uwe equals the (N-1)th column of uw, and the
% 1st and last rows of uwe are generated similarly.
% Written by W.-S. Lu, University of Victoria.
% Last modified: Jan. 3, 2008.
function uwe = ex_sym2(uw)
[M,N] = size(uw);
w = zeros(M+2,N+2);
w(2:(M+1),2:(N+1)) = uw;
w(1,2:(N+1)) = uw(2,:);
```

```
w((M+2),2:(N+1)) = uw((M-1),:);
w(2:(M+1),1) = uw(:,2);
w(2:(M+1),(N+2)) = uw(:,(N-1));
w(1,1) = uw(2,2);
w(1,(N+2)) = uw(2,(N-1));
w((M+2),1) = uw((M-1),2);
w((M+2),(N+2)) = uw((M-1),(N-1));
uwe = w;
```

Example 4.2

MATLAB function: phi = cks_ak(obj2,p02,0.1,0.5,5,1,6,7,20,10);
for phi = cks_ak(u,phi0,dt,dt0,a,sig,k,K_0,K_in,K_out)

Image: obj2.mat (size: 114×114)
Initial level-set function: $\varphi(0,x,y) = 49 - \sqrt{(x-56)^2 + (y-56)^2}$ (a discrete version is given by p02.mat)
Step size in time $\tau = 0.1$
Step size in time for re-initialization $\tau_0 = 0.5$
Parameter α : 5
σ for the Gaussian kernel: 1
k for the Gaussian kernel: 6
Number of iterations for re-initialization: 7
Number of iterations after each initialization (internal iteration): 20
Number of internal iteration rounds: 10

See Fig. 4.7 for the image for segmentation and Fig. 4.8 for the zero-level of the level-set function after various number of iterations.

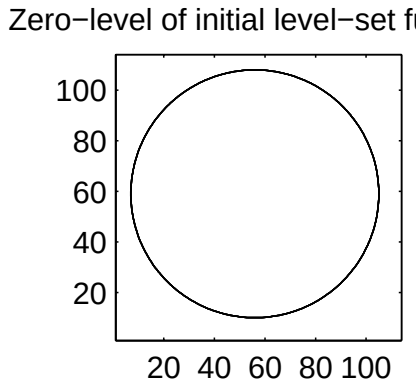
Example 4.3

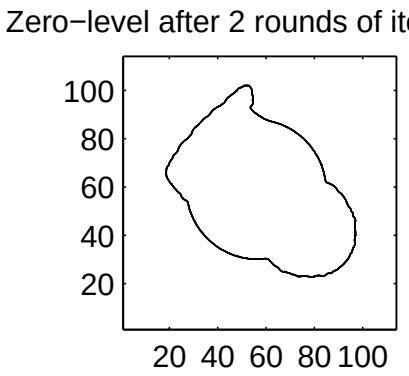
MATLAB function: phi = cks_ak(obj3,p03,0.1,0.5,5,1.2,3,10,20,10);
for phi = cks_ak(u,phi0,dt,dt0,a,sig,k,K_0,K_in,K_out)
Image: obj3.mat (size: 192×192)
Initial level-set function: $\varphi(0,x,y) = 82 - \sqrt{(x-95)^2 + (y-95)^2}$ (a discrete version is given by p03.mat)
Step size in time $\tau = 0.1$
Step size in time for re-initialization $\tau_0 = 0.5$
Parameter α : 5
σ for the Gaussian kernel: 1.2
k for the Gaussian kernel: 3
Number of iterations for re-initialization: 10
Number of iterations after each initialization (internal iteration): 20
Number of internal iteration rounds: 10

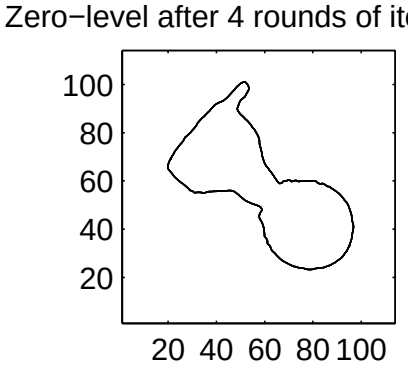
See Fig. 4.9 for the image for segmentation and Fig. 4.10 for the zero-level of the level-set function after various number of iterations.

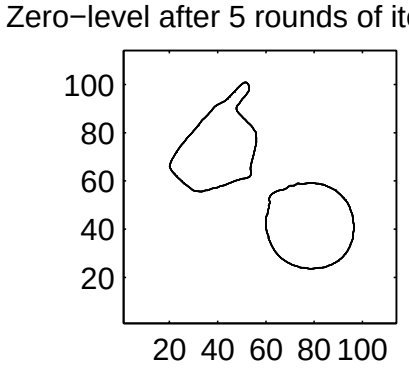
Figure 4.7 for Example 4.2.

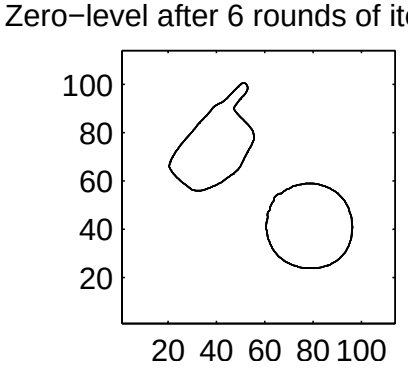
Figure 4.9 for Example 4.3.











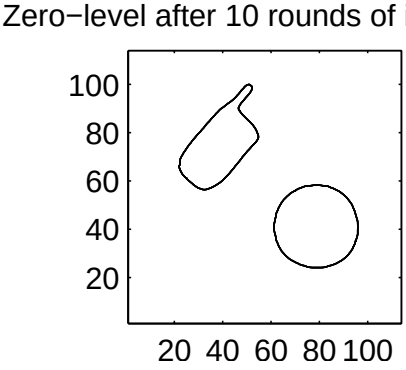


Figure 4.8 for Example 4.2.

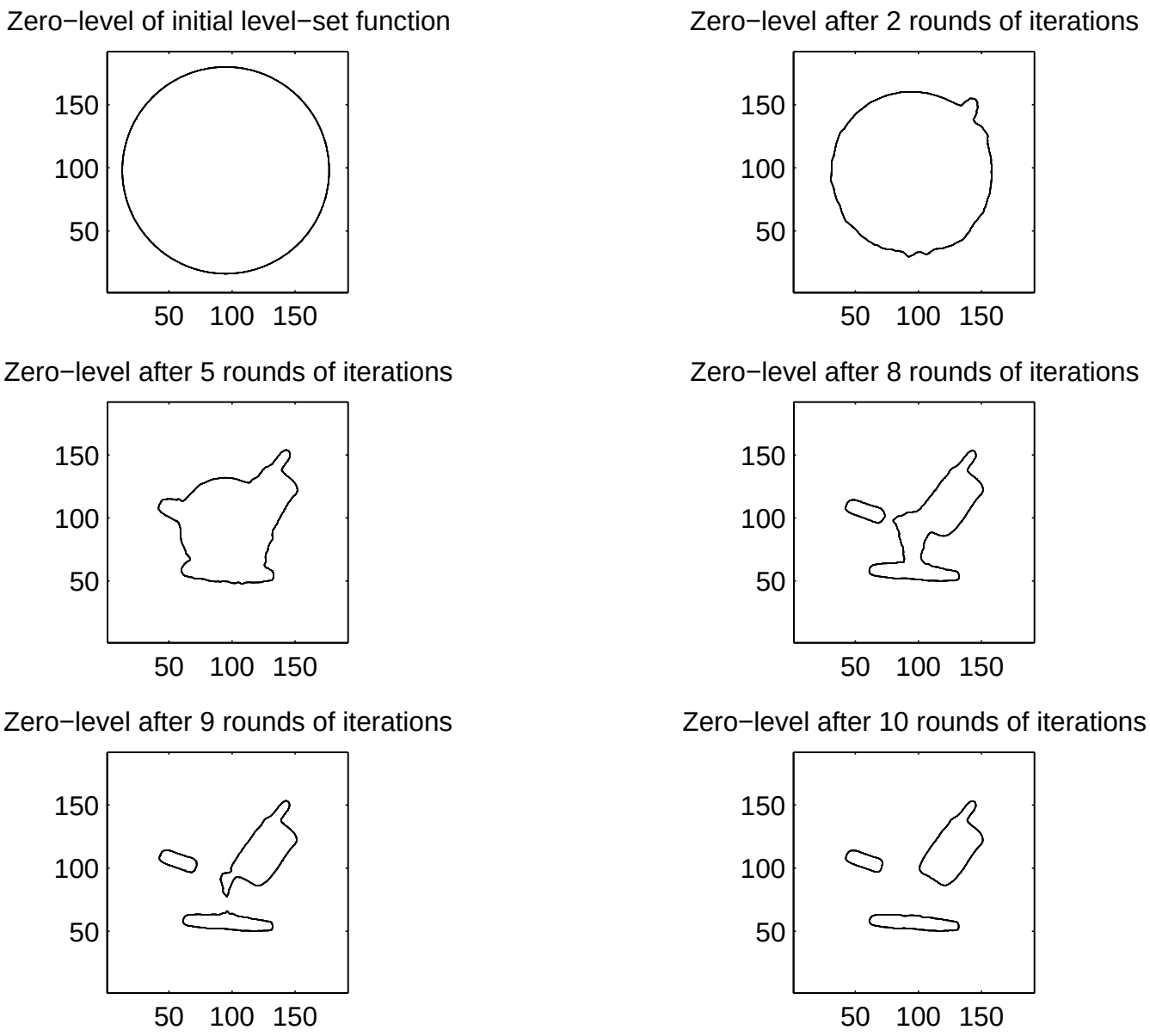


Figure 4.10 for Example 4.3.

4.3 The Chan-Vese Model

4.3.1 *Introduction*

This section devotes to a relative recent development in image segmentation described in reference [17] where Chan and Vese proposed a new model for active contours. The method is based on curve evolution, Mumford-Shah functional, and level sets. The model can detect objects whose boundaries are not necessarily defined by gradient because the stopping term does not depend on the gradient of the image. For this reason the authors call their method "*active contours without edges* [17]". Advantages the Chan-Vese model offers include its ability to detect interior contours and relaxation of initial curve to practically anywhere in the image.

4.3.2 *The Model*

4.3.2.1 The Chan-Vese Functional

The Chan-Vese (CV) model considers a two-value (as known as two-phase) approximation of a given image, and the corresponding functional is given by

$$J_{CV}(c_1, c_2, \Gamma) = \lambda_1 \int_{\text{inside}(\Gamma)} \left(u_0(x,y) - c_1\right)^2 dx\,dy + \lambda_2 \int_{\text{outside}(\Gamma)} \left(u_0(x,y) - c_2\right)^2 dx\,dy + \nu \cdot |\Gamma| \qquad (4.37)$$

where c_1 and c_2 are constants and $|\Gamma|$ denotes the length of curve Γ. Recall the Mumford-Shah functional which is given by

$$J_{MS}(u, \Gamma) = \int_{\Omega} |u_0 - u|^2 \, dx\,dy + \mu \cdot \int_{\Omega \setminus \Gamma} |\nabla u|^2 \, dx\,dy + \nu \cdot |\Gamma| \qquad (4.38)$$

One can regard (4.37) as a special case of (4.38) where one assumes $\lambda_1 = \lambda_2 = 1$ and seeks to find a piecewise constant approximation $u(x, y)$ of image $u_0(x, y)$ with $u(x, y)$ taking only two values. From Example 4.1, the best choices of these values are the mean values of the image $u_0(x, y)$ inside and outside Γ respectively, namely,

$$u(x,y) = \begin{cases} c_1 = \text{mean}(u_0) \text{ inside } \Gamma \\ c_2 = \text{mean}(u_0) \text{ outside } \Gamma \end{cases} \qquad (4.39)$$

Like in the CKS model, curve Γ is represented implicitly by means of a level-set function $\varphi(x, y)$ which is defined by

$$\begin{cases} \Gamma = \partial S = \{(x,y) \in \Omega : \varphi(x,y) = 0\} \\ \text{inside}(\Gamma) = S = \{(x,y) \in \Omega : \varphi(x,y) > 0\} \\ \text{outside}(\Gamma) = \Omega \setminus \overline{S} = \{(x,y) \in \Omega : \varphi(x,y) < 0\} \end{cases} \qquad (4.40)$$

The three terms in the CV functional are treated in terms of the corresponding level-set function $\varphi(x, y)$ as follows (for simplicity below $\lambda_1 = \lambda_2 = 1$ have been assumed).

(a) The curve length term in (4.37). By using the one-variable Heaviside function defined by

$$H(z)=\begin{cases}1 & \text{if } z\geq 0\\ 0 & \text{if } z<0\end{cases} \tag{4.41}$$

and Dirac's delta function defined by

$$\delta(z)=\frac{dH(z)}{dz} \tag{4.42}$$

the curve length $|\Gamma|$ can be expressed as

$$\begin{aligned}|\Gamma| &= \text{length}\left(\varphi(x,y)=0\right)\\ &= \int_{\Omega}|\nabla H\left(\varphi(x,y)\right)|\,dx\,dy\\ &= \int_{\Omega}\delta\left(\varphi(x,y)\right)|\nabla\varphi(x,y)|\,dx\,dy\end{aligned} \tag{4.43}$$

(b) From (4.40), the region inside curve Γ is characterized by $\varphi(x,y) > 0$, hence the first integral in (4.37) can be expressed in terms of level-set function $\varphi(x,y)$ as

$$\int_{\varphi>0}\left(u_0(x,y)-c_1\right)^2 dx\,dy=\int_{\Omega}\left(u_0(x,y)-c_1\right)^2 H\left(\varphi(x,y)\right)dx\,dy \tag{4.44}$$

(c) Similarly, since the region outsize Γ is characterized by $\varphi(x,y) < 0$, we can write the second integral as

$$\int_{\varphi<0}\left(u_0(x,y)-c_2\right)^2 dx\,dy=\int_{\Omega}\left(u_0(x,y)-c_2\right)^2\left(1-H\left(\varphi(x,y)\right)\right)dx\,dy \tag{4.45}$$

For a *fixed* level-set function $\varphi(x,y)$ whose zero level set defines the boundary Γ, it follows from (4.39) that the optimal constants c_1 and c_2 can be computed (assuming curve Γ has a nonempty interior in Ω)

$$\begin{aligned}c_1 &= \text{mean}(u_0) \text{ inside } \Gamma\\ &= \frac{\int_{\text{inside}(\Gamma)}u_0(x,y)\,dx\,dy}{\text{area}(\text{inside}(\Gamma))}\\ &= \frac{\int_{\varphi>0}u_0(x,y)\,dx\,dy}{\int_{\varphi>0}1\,dx\,dy}\\ &= \frac{\int_{\Omega}u_0(x,y)\,H\left(\varphi(x,y)\right)dx\,dy}{\int_{\Omega}H\left(\varphi(x,y)\right)dx\,dy}\end{aligned} \tag{4.46}$$

and (assuming curve Γ has a nonempty exterior in Ω).

$$
\begin{aligned}
c_2 &= \text{mean}(u_0) \text{ outside } \Gamma \\
&= \frac{\int_{\text{outside}(\Gamma)} u_0(x,y)\,dx\,dy}{\text{area}(\text{outside}(\Gamma))} \\
&= \frac{\int_{\varphi<0} u_0(x,y)\,dx\,dy}{\int_{\varphi<0} 1\,dx\,dy} \\
&= \frac{\int_{\Omega} u_0(x,y)\big(1-H\left(\varphi(x,y)\right)\big)\,dx\,dy}{\int_{\Omega}\big(1-H\left(\varphi(x,y)\right)\big)\,dx\,dy}
\end{aligned}
\tag{4.47}
$$

and the two-phase (2-value) piecewise constant approximation of image $u_0(x, y)$ can be characterized in terms of level-set function as

$$
u(x,y) = c_1 H\left(\varphi(x,y)\right) + c_2\left(1-H\left(\varphi(x,y)\right)\right) \quad \text{for } (x,y) \in \bar{\Omega} \tag{4.48}
$$

4.3.2.2 A Regularized Chan-Vese Functional

In order to compute the associated Euler-Lagrange equation for $\varphi(x, y)$, functions $H(z)$ in (4.41) and $\delta(z)$ in (4.42) need to be slightly modified (regularized). One such regularization that has found beneficial [17] is given by

$$
H_\varepsilon(z) = \frac{1}{2}\left(1+\frac{2}{\pi}\arctan\left(\frac{z}{\varepsilon}\right)\right) \tag{4.49}
$$

for some small positive ε. Accordingly the delta function is modified to the first-order derivative of $H_\varepsilon(z)$, i.e.,

$$
\delta_\varepsilon = \frac{d}{dz} H_\varepsilon(z) = \frac{\varepsilon}{\pi(z^2+\varepsilon^2)} \tag{4.50}
$$

Figures 4.11 and 4.12 show an $H_\varepsilon(z)$ and an $\delta_\varepsilon(z)$ with $\varepsilon = 1/\pi$, respectively.

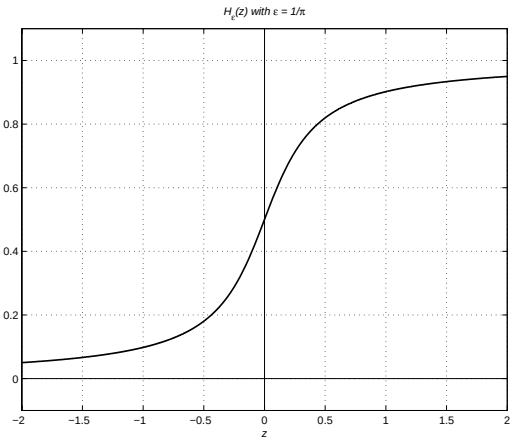


Figure 4.11: Function $H_\varepsilon(z)$.

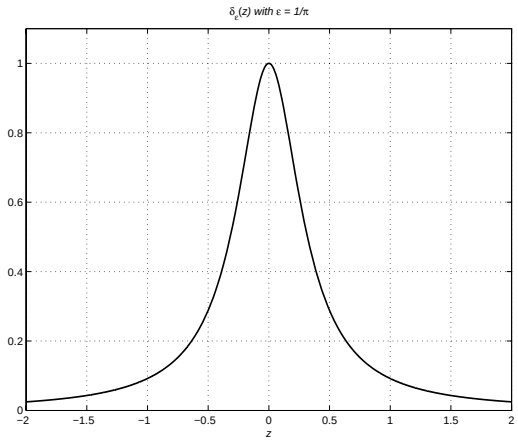


Figure 4.12: Function $\delta_\varepsilon(z)$

Using (4.43), (4.44) and (4.45), the CV functional can be expressed in terms of level-set function $\varphi(x, y)$ as

$$\begin{aligned} &J_{CV}(c_1, c_2, \varphi) = \\ &\nu \int_\Omega \delta\left(\varphi(x,y)\right) |\nabla \varphi(x,y)|\, dx\, dy \\ &+\lambda_1 \int_\Omega \left(u_0(x,y) - c_1\right)^2 H\left(\varphi(x,y)\right) dx\, dy \\ &+\lambda_2 \int_\Omega \left(u_0(x,y) - c_2\right)^2 \left(1 - H\left(\varphi(x,y)\right)\right) dx\, dy \end{aligned} \tag{4.51}$$

By assuming $\lambda_1 = \lambda_2 = 1$ and replacing $H(z)$ and $\delta(z)$ with $H_\varepsilon(z)$ and $\delta_\varepsilon(z)$, respectively, in (4.51), we obtain a regularized CV functional as

$$\begin{aligned} &J_{CV}(c_1, c_2, \varphi) = \\ &\nu \int_\Omega \delta_\varepsilon\left(\varphi(x,y)\right) |\nabla \varphi(x,y)|\, dx\, dy \\ &+\int_\Omega \left(u_0(x,y) - c_1\right)^2 H_\varepsilon\left(\varphi(x,y)\right) dx\, dy \\ &+\int_\Omega \left(u_0(x,y) - c_2\right)^2 \left(1 - H_\varepsilon\left(\varphi(x,y)\right)\right) dx\, dy \end{aligned} \tag{4.52}$$

If we *fix* c_1 and c_2 and minimize the regularized CV functional (4.52) with respect to level-set function $\varphi(x, y)$, then we obtain a Euler-Lagrange equation for $\varphi(x, y)$. This EL equation is then imbedded into a time-evolution equation as follows:

$$\begin{aligned} &\frac{\partial \varphi}{\partial t} = \delta_\varepsilon(\varphi)\left[\nu \operatorname{div}\left(\frac{\nabla \varphi}{|\nabla \varphi|}\right) - \left(u_0 - c_1\right)^2 + \left(u_0 - c_2\right)^2\right] \\ &\varphi(0, x, y) = \varphi_0(x, y) \quad \text{in} \quad \Omega \\ &\frac{\delta_\varepsilon(\varphi)}{|\nabla \varphi|} \frac{\partial \varphi}{\partial N} = 0 \quad \text{on} \quad \partial\Omega \end{aligned} \tag{4.53}$$

4.3.2.3 Difference Schemes for (4.53)

An implicit difference scheme for EL equation in (4.53) is also proposed in [17] and it is given by

$$\begin{aligned} &\frac{\varphi_{i,j}^{n+1} - \varphi_{i,j}^{n}}{\tau} = \delta_h\left(\varphi_{i,j}^{n}\right)\left[\frac{\nu}{h^2}\Delta_x^-\left(\frac{\Delta_x^+ \varphi_{i,j}^{n+1}}{\sqrt{(\Delta_x^+ \varphi_{i,j}^{n})^2 / h^2 + (\varphi_{i,j+1}^{n} - \varphi_{i,j-1}^{n})^2 / (2h)^2}}\right)\right. \\ &+\frac{\nu}{h^2}\Delta_y^-\left(\frac{\Delta_y^+ \varphi_{i,j}^{n+1}}{\sqrt{(\varphi_{i+1,j}^{n} - \varphi_{i-1,j}^{n})^2 / (2h)^2 + (\Delta_y^+ \varphi_{i,j}^{n})^2 / h^2}}\right) \\ &\left.-\left((u_0)_{i,j} - c_1(\varphi^n)\right)^2 + \left((u_0)_{i,j} - c_2(\varphi^n)\right)^2\right] \end{aligned} \tag{4.54}$$

The implicit nature of the scheme implies that one needs to solve a system of linear equations in each iteration that advances the solution in a time step of size τ.

Alternatively, there are explicit schemes for the equation in (4.53). By writing

$$\kappa = \operatorname{div}\left(\frac{\nabla\varphi}{|\nabla\varphi|}\right)$$

the PDE in (4.53) can be expressed as

$$\frac{\partial\varphi}{\partial t} = \delta_\varepsilon(\varphi)\left[\nu\kappa - (u_0 - c_1)^2 + (u_0 - c_2)^2\right] \tag{4.55}$$

An explicit scheme for (4.55) is given by

$$\varphi_{i,j}^{n+1} = \varphi_{i,j}^n + \tau\delta_\varepsilon\left(\varphi_{i,j}^n\right)\left[\nu\kappa_{i,j}^n - \left((u_0)_{i,j} - c_1(\varphi_{i,j}^n)\right)^2 + \left((u_0)_{i,j} - c_2(\varphi_{i,j}^n)\right)^2\right] \tag{4.56a}$$

where

$$c_1\left(\varphi_{i,j}^n\right) = \frac{\sum_i\sum_j\left((u_0)_{i,j}\cdot H\left(\varphi_{i,j}^n\right)\right)}{\sum_i\sum_j H\left(\varphi_{i,j}^n\right)} \tag{4.56b}$$

$$c_2\left(\varphi_{i,j}^n\right) = \frac{\sum_i\sum_j\left((u_0)_{i,j}\cdot\left(1 - H\left(\varphi_{i,j}^n\right)\right)\right)}{\sum_i\sum_j\left(1 - H\left(\varphi_{i,j}^n\right)\right)} \tag{4.56c}$$

$$\delta_\varepsilon\left(\varphi_{i,j}^n\right) = \frac{\varepsilon}{\pi\left(\left(\varphi_{i,j}^n\right)^2 + \varepsilon^2\right)} \tag{4.56d}$$

and $\kappa_{i,j}^n$ approximates the curvature of the level-set function through

$$\kappa = \operatorname{div}\left(\frac{\nabla\varphi}{|\nabla\varphi|}\right) = \frac{\varphi_{xx}\varphi_y^2 - 2\varphi_x\varphi_y\varphi_{xy} + \varphi_{yy}\varphi_x^2}{\left(\varphi_x^2 + \varphi_y^2\right)^{3/2}} \tag{4.56e}$$

where the partial derivatives are approximated using central difference schemes.

Base on (4.56), we can summarize the Chan-Vese segmentation algorithm as follows.

Step 1: Prepare an initial level-set function $\varphi(0, x, y) = \varphi_0(x, y)$ and set $n = 0$.

Step 2: Compute $c_1\left(\varphi_{i,j}^n\right)$ and $c_2\left(\varphi_{i,j}^n\right)$ by (4.56b) and (4.56c).

Step 3: Update the PDE to obtain $\varphi_{i,j}^{n+1}$ using (4.56a).

Step 4: Reinitialize φ every 15 to 20 iterations to the signed distance function whose zero-level set is the same as that of the input level-set function.

Step 5: Check if the solution becomes stationary. If not, set $n = n + 1$ and repeat from Step 2.

4.3.2.4 MATLAB Code and Examples

MATLAB implementation of the CV algorithm is performed by means of a main function cv_ak.m which requires two functions: re_initial.m and ex_sym2.m. Since re_initial.m and ex_sym2.m are identical to those listed in Sec. 4.2, below we only list function cv_ak.m.

```
% To perform image segmentation by using the active contour method proposed
% in T. F. Chan and L. Vese , "Active contours without edges," IEEE Tarns.
% Image Processing, vol. 10, no. 2, pp. 266-277, Feb. 2001.
% For the difference scheme employed here, please refer to Appendix A.3.4
% of the book: Gilles Aubert and Pierre Kornprobst, Mathematical Problems
% in Image Processing, Springer, New York, 2002.
% Inputs:   u -- image to be segmented
%        phi0 -- initial level-set function
%          dt -- step size in time
%         dt0 -- step size in time for re-initialization
%          ep -- epsilon for regularized delta function
%          nu -- parameter nu
%         K_0 -- number of iterations for re-initialization
%        K_in -- number of iterations before next re-initialization
%               (number of internal iterations)
%       K_out -- number of internal iteration rounds
% Output: phi -- final level-set function
% Written by W.-S. Lu, University of Victoria.
% Last modified: March 9, 2008.
% Examples: [phi,c1,c2,ua] = cv_ak(obj2,p02,0.01,0.1,3,25,3,10,30);
%           [phi,c1,c2,ua] = cv_ak(obj3a_cv,p03a_cv,0.05,0.1,3,30,3,20,30);
%           [phi,c1,c2,ua] = cv_ak(obj3b_cv,p03b_cv,0.33,0.15,5,5,10,20,16);
%           [phi,c1,c2,ua] = cv_ak(curves,p0_curves,0.15,0.5,3,25,3,20,12);
function [phi,c1,c2,ua] = cv_ak(u,phi0,dt,dt0,ep,nu,K_0,K_in,K_out)

[M,N] = size(u);
yt = M:-1:1;
xt = 1:1:N;
epsi = 1e-6;
v = -1e-6:5e-7:1e-6;
won = ones(M,N);
wep = ep*won;
map = gray(256);
figure(1)
subplot(121)
imshow(u,map)
title('Image to be segmented')
pause(1)
pw = phi0;
it = 0;
figure(2)
contour(xt,yt,pw,v)
title('Zero-level of initial level-set function')
axis square
pause(0.5)
while it < K_out,
   itt = 0;
```

```
   while itt < K_in,
      H = 0.5*(won+sign(pw));
      H1 = 1 - H;
      c1 = sum(sum(u.*H))/sum(sum(H));
      c2 = sum(sum(u.*H1))/sum(sum(H1));
      delt = wep./(pi*(wep.^2+pw.^2));
      pww = ex_sym2(pw);
      pa = pww(2:(M+1),3:(N+2)); % phi_i+1,j
      pb = pww(3:(M+2),2:(N+1)); % phi_i,j+1
      pc = pww(2:(M+1),1:N);     % phi_i-1,j
      pd = pww(3:(M+2),1:N);     % phi_i-1,j+1
      pe = pww(1:M,2:(N+1));     % phi_i,j-1
      pf = pww(1:M,3:(N+2));     % phi_i+1,j-1
      pxx = pa - 2*pw + pc;
      pyy = pb - 2*pw + pe;
      pxy = 0.5*(pa -2*pw - pf + pe + pb - pd + pc);
      px = (pa - pc)/2;
      py = (pb - pe)/2;
      px2 = px.^2;
      py2 = py.^2;
      rr = (px2 + py2 + epsi).^1.5;
      Kw = pxx.*py2 + pyy.*px2 - 2*(px.*py).*pxy;
      Ku = nu*(Kw./rr);
      T1 = -(u-c1).^2 + (u-c2).^2;
      pw = pw + dt*(delt.*(Ku + T1));
      itt = itt + 1;
   end
   pw_new = re_initial(pw,dt0,K_0);
   pw = pw_new;
   it = it + 1;
   figure(2)
   contour(xt,yt,pw,v)
   title(sprintf('Zero-level after %.5g rounds of iterations',it'))
   axis square
end
phi = pw;
H = 0.5*(won+sign(phi));
H1 = 1 - H;
c1 = sum(sum(u.*H))/sum(sum(H));
c2 = sum(sum(u.*H1))/sum(sum(H1));
ua = c1*H + c2*H1;
disp('Optimal values of c1 and c2:')
[c1 c2]
figure(1)
subplot(122)
imshow(ua,map)
title('2-value piecewise constant approximation of the image')
```

Example 4.4

MATLAB function: [phi,c1,c2,ua] = cv_ak(obj2,p02,0.01,0.1,3,25,3,10,30);
for [phi,c1,c2,ua] = cv_ak(u,phi0,dt,dt0,ep,nu,K_0,K_in,K_out)
Image: obj2.mat (size: 114×114)

Initial level-set function: $\varphi(0,x,y) = 49 - \sqrt{(x-56)^2 + (y-56)^2}$ (a discrete version is given by p02.mat)
Step size in time τ = 0.01
Step size in time for re-initialization τ_0 = 0.1
Parameter ε : 3
Parameter ν : 25
Number of iterations for re-initialization: 3
Number of iterations after each initialization (internal iteration): 10
Number of internal iteration rounds: 30

The highest and lowest light intensity in the image were 255 and 0, respectively. The optimal values of the constants obtained from the algorithm (after 300 iterations) were c_1 = 50.1388 and c_2 = 252.4578. These values of c_1 and c_2 were used to construct the two-value piecewise-constant approximation of the image using (4.48).

Fig. 4.13 depicts the image for segmentation (on the left) and the 2-value piecewise-constant approximation of the image (on the right), and Fig. 4.14 shows the zero-level of the level-set function after various number of iterations.

These results demonstrate that the CV model can detect image's internal boundaries.

Example 4.5

MATLAB function: [phi,c1,c2,ua] = cv_ak(obj3a_cv,p03a_cv,0.05,0.1,3,30,3,20,30);
for [phi,c1,c2,ua] = cv_ak(u,phi0,dt,dt0,ep,nu,K_0,K_in,K_out)
Image: obj3a_cv.mat (size: 120 × 120)
Initial level-set function: $\varphi(0,x,y) = 57.5 - \sqrt{(x-60.5)^2 + (y-60.5)^2}$ (a discrete version of it is given by p03a_cv.mat)
Step size in time τ = 0.05
Step size in time for re-initialization τ_0 = 0.1
Parameter ε : 3
Parameter ν : 30
Number of iterations for re-initialization: 3
Number of iterations after each initialization (internal iteration): 20
Number of internal iteration rounds: 30

In the noise-free image, the light intensity for the objects was 40 and the light intensity in the background was 180. The optimal values of the constants obtained from the algorithm (after 600 iterations) were c_1 = 40.0674 and c_2 = 180.0461.

Fig. 4.15 depicts the image for segmentation (on the left) and the 2-value piecewise-constant approximation of the image (on the right), and Fig. 4.16 shows the zero-level of the level-set function after various number of iterations.

These results demonstrate that the CV model also works well for noisy images.

Example 4.6

MATLAB function: [phi,c1,c2,ua] = cv_ak(obj3b_cv,p03b_cv,0.33,0.15,5,5,10,20,16);

for [phi,c1,c2,ua] = cv_ak(u,phi0,dt,dt0,ep,nu,K_0,K_in,K_out)
Image: obj2.mat (size: 120 × 120)
Initial level-set function: $\varphi(0,x,y) = 27.5 - \sqrt{(x-30.5)^2 + (y-30.5)^2}$ (a discrete version of it is given by p03b_cv.mat)
Step size in time τ = 0.33
Step size in time for re-initialization τ_0 = 0.15
Parameter ε: 5
Parameter ν: 5
Number of iterations for re-initialization: 10
Number of iterations after each initialization (internal iteration): 20
Number of internal iteration rounds: 16

In the noise-free image, the light intensity was 0 for the triangle, 128 for the ring, and 180 for the third object, and the light intensity in the background was 255. The optimal values of the constants obtained from the algorithm (after 320 iterations) were c_1 = 250.8163 and c_2 = 125.5001.

Fig. 4.17 depicts the image for segmentation (on the left) and the 2-value piecewise-constant approximation of the image (on the right), and Fig. 4.18 shows the zero-level of the level-set function after various number of iterations.

These results demonstrate that the CV model can also deal with images with more-than-two light intensity levels.

Example 4.7

MATLAB function: [phi,c1,c2,ua] = cv_ak(curves,p0_curves,0.15,0.5,3,25,3,20,12);
for [phi,c1,c2,ua] = cv_ak(u,phi0,dt,dt0,ep,nu,K_0,K_in,K_out)
Image: curves.mat (size: 150 × 150)
Initial level-set function: $\varphi(0,x,y) = 50 - \sqrt{(x-74.5)^2 + (y-74.5)^2}$ (a discrete version of it is given by p0_curves.mat)
Step size in time τ = 0.15
Step size in time for re-initialization τ_0 = 0.5
Parameter ε: 3
Parameter ν: 25
Number of iterations for re-initialization: 3
Number of iterations after each initialization (internal iteration): 20
Number of internal iteration rounds: 12

The minimum light intensity of the thick curves was 25 and the light intensity in the background was 255. The optimal values of the constants obtained from the algorithm (after 240 iterations) were c_1 = 197.9518 and c_2 = 251.3805.

Fig. 4.19 depicts the image for segmentation (on the left) and the 2-value piecewise-constant approximation of the image (on the right), and Fig. 4.20 shows the zero-level of the level-set function after various number of iterations.

These results demonstrate that the CV model can also deal with images curves and open lines with conjunctions.

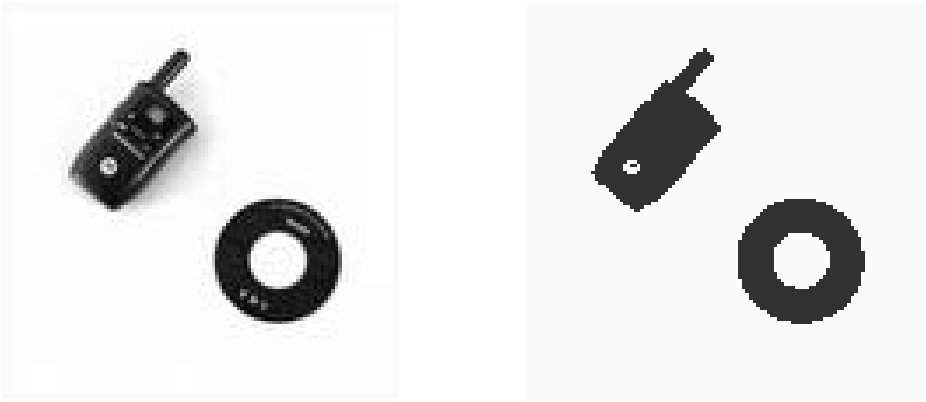


Figure 4.13 for Example 4.4

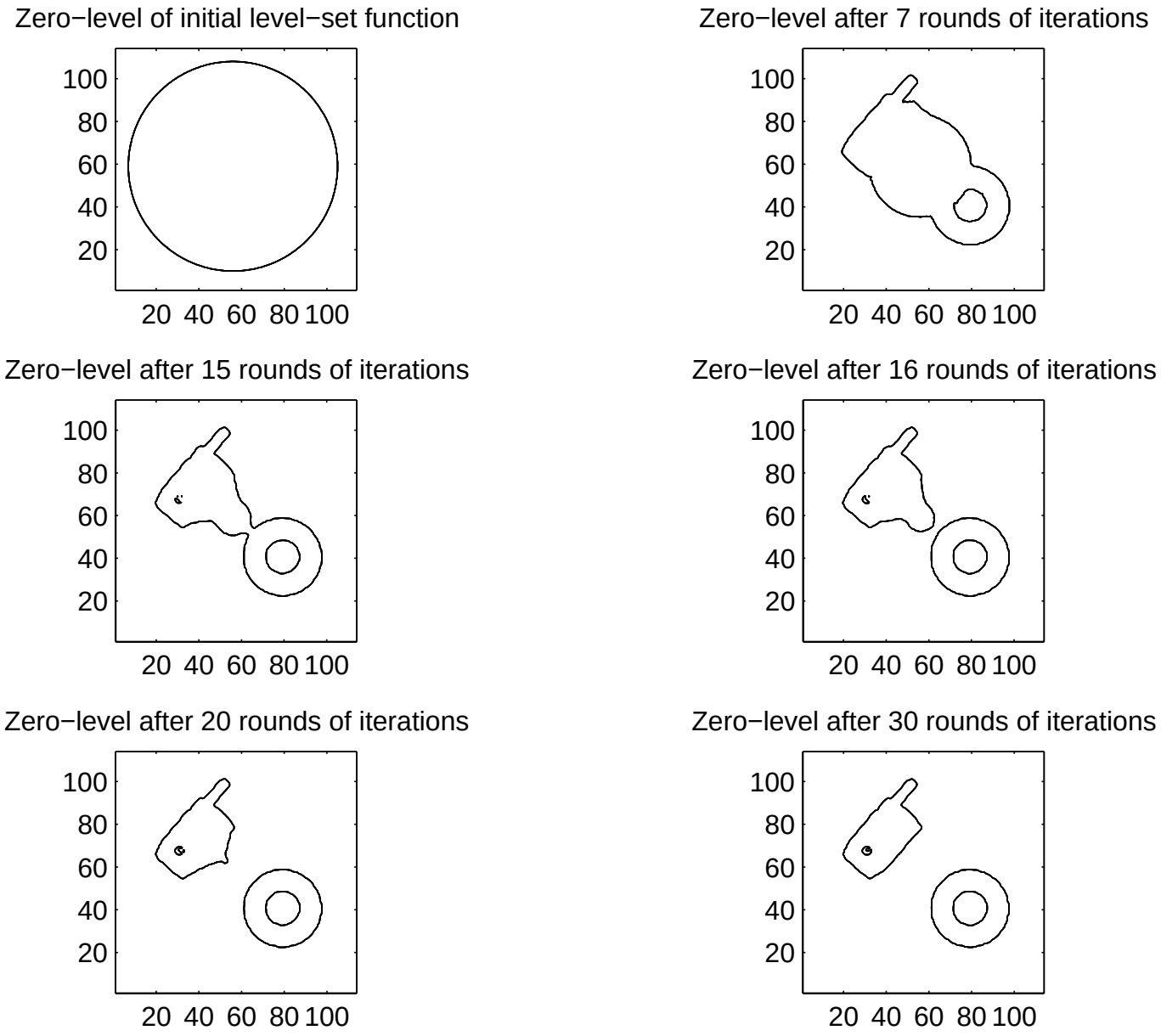


Figure 4.14 for Example 4.4

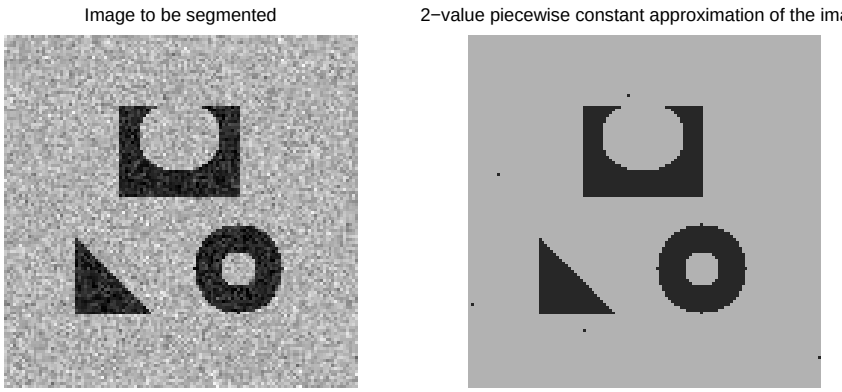


Figure 4.15 for Example 4.5.

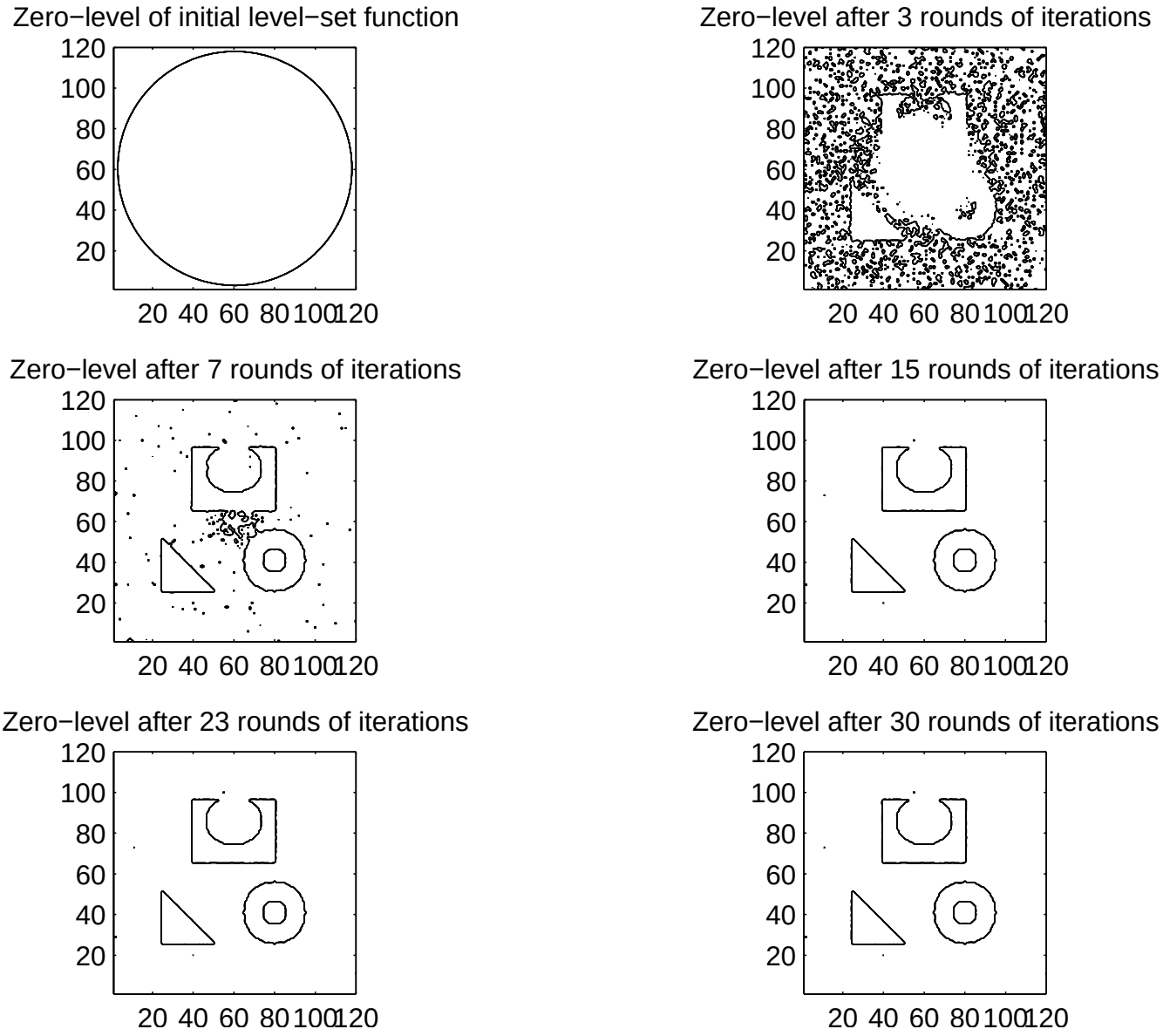


Figure 4.16 for Example 4.5.

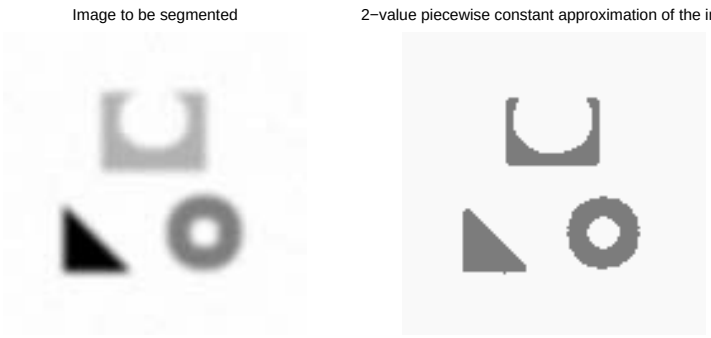


Figure 4.17 for Example 4.6.

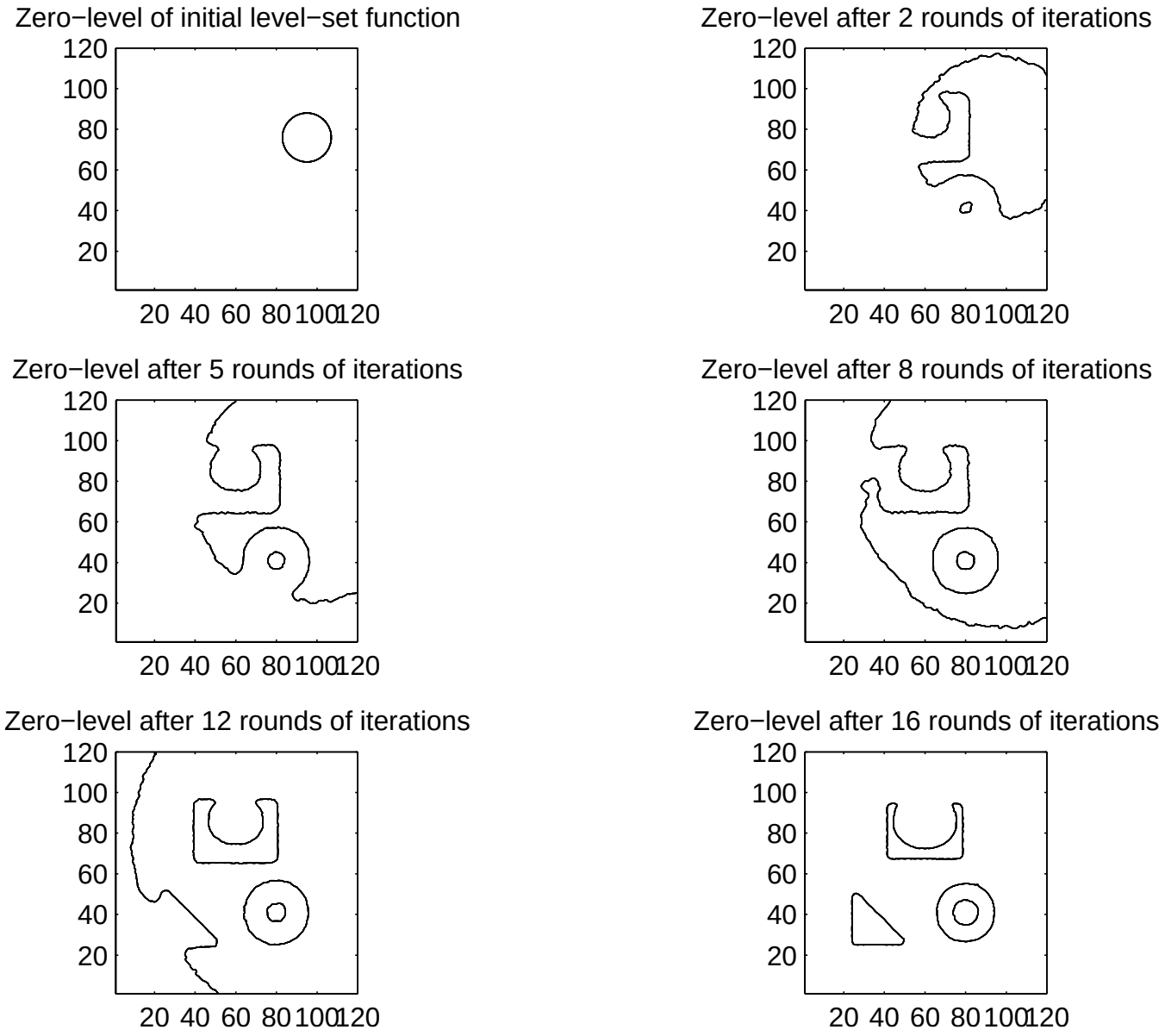


Figure 4.18 for Example 4.6.

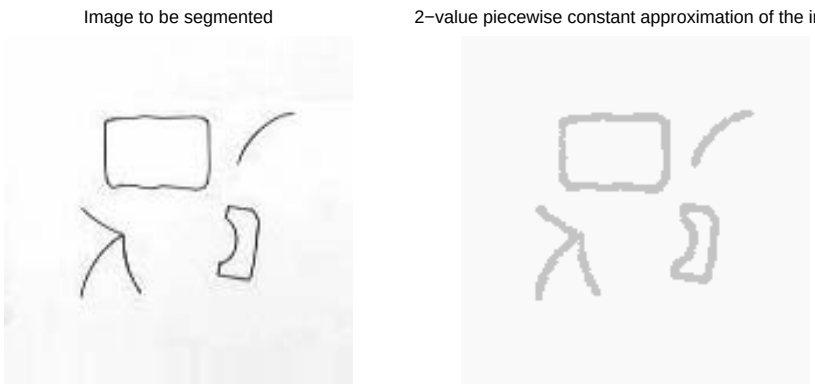


Figure 4.19 for Example 4.7.

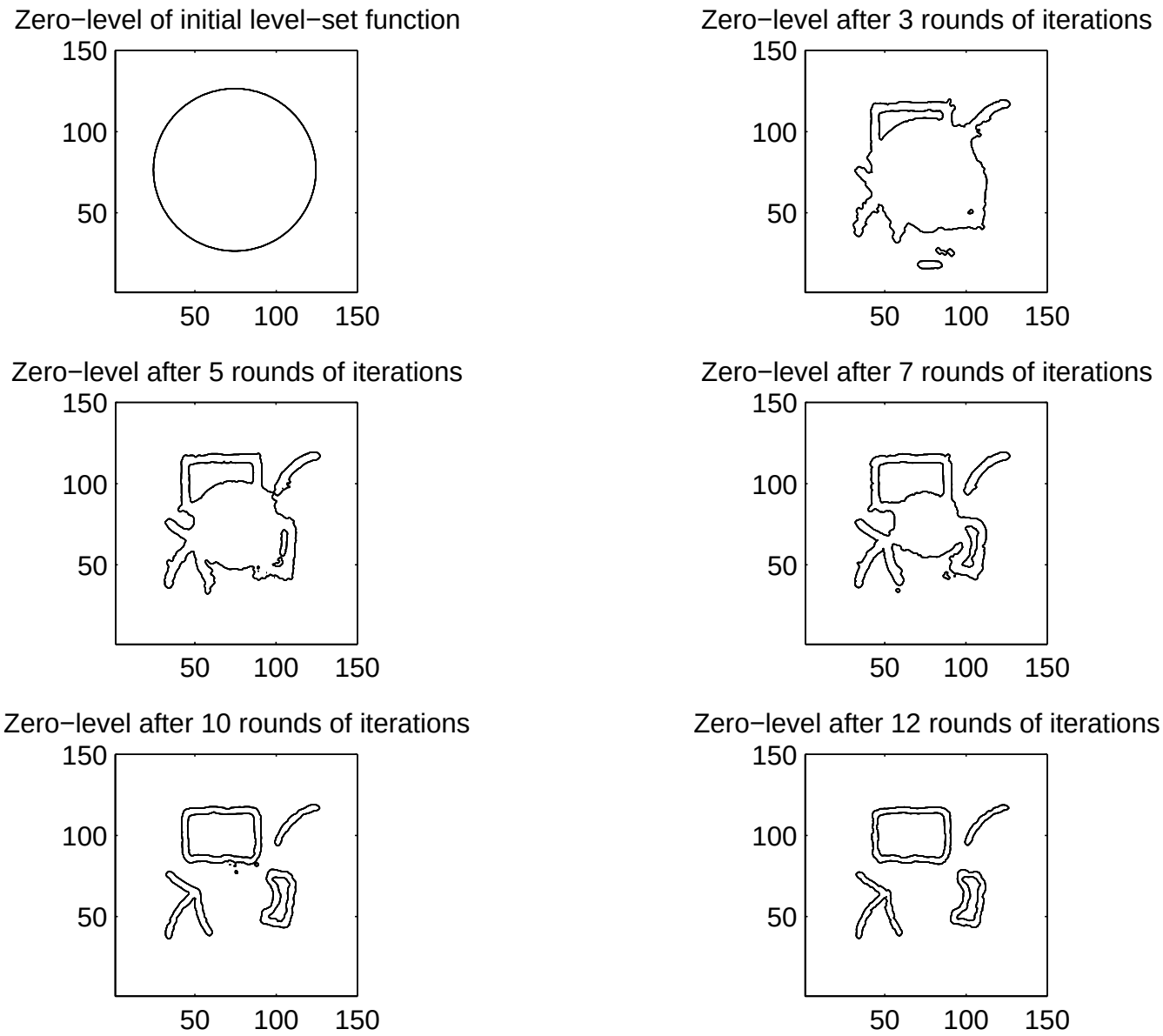


Figure 4.20 for Example 4.7.

4.3.2.5 Difference Schemes Revisited: A Semi-Implicit Scheme Based on (4.54)

The explicit scheme in (4. 56) is proven to work. In what follows we describe a seme-implicit scheme based on (4.54). To start, we recall the implicit scheme in (4.54):

$$\frac{\varphi_{i,j}^{n+1}-\varphi_{i,j}^{n}}{\tau}=\delta_{\varepsilon}\left(\varphi_{i,j}^{n}\right)\left[\frac{\nu}{h^2}\Delta_x^-\left(\frac{\Delta_x^+\varphi_{i,j}^{n+1}}{\sqrt{(\Delta_x^+\varphi_{i,j}^{n})^2/h^2+(\varphi_{i,j+1}^{n}-\varphi_{i,j-1}^{n})^2/(2h)^2}}\right)\right.$$
$$+\frac{\nu}{h^2}\Delta_y^-\left(\frac{\Delta_y^+\varphi_{i,j}^{n+1}}{\sqrt{(\varphi_{i+1,j}^{n}-\varphi_{i-1,j}^{n})^2/(2h)^2+(\Delta_y^+\varphi_{i,j}^{n})^2/h^2}}\right)$$
$$\left.-\left((u_0)_{i,j}-c_1(\varphi^n)\right)^2+\left((u_0)_{i,j}-c_2(\varphi^n)\right)^2\right]$$

For simplicity below we assume $h = 1$. The differences in the first term in the right-hand side (RHS) can be explicitly expressed as

$$\Delta_x^-\left(\frac{\Delta_x^+\varphi_{i,j}^{n+1}}{\sqrt{(\Delta_x^+\varphi_{i,j}^{n})^2+(\varphi_{i,j+1}^{n}-\varphi_{i,j-1}^{n})^2/2^2}}\right)$$
$$=\frac{\varphi_{i+1,j}^{n+1}-\varphi_{i,j}^{n+1}}{\sqrt{\left(\varphi_{i+1,j}^{n}-\varphi_{i,j}^{n}\right)^2+\left[\left(\varphi_{i,j+1}^{n}-\varphi_{i,j-1}^{n}\right)/2\right]^2}}-\frac{\varphi_{i,j}^{n+1}-\varphi_{i-1,j}^{n+1}}{\sqrt{\left(\varphi_{i,j}^{n}-\varphi_{i-1,j}^{n}\right)^2+\left[\left(\varphi_{i-1,j+1}^{n}-\varphi_{i-1,j-1}^{n}\right)/2\right]^2}} \tag{4.57}$$

Similarly, we write the differences in the second term in the RHS as

$$\Delta_y^-\left(\frac{\Delta_y^+\varphi_{i,j}^{n+1}}{\sqrt{(\varphi_{i+1,j}^{n}-\varphi_{i-1,j}^{n})^2/2^2+(\Delta_y^+\varphi_{i,j}^{n})^2}}\right)$$
$$=\frac{\varphi_{i,j+1}^{n+1}-\varphi_{i,j}^{n+1}}{\sqrt{\left[\left(\varphi_{i+1,j}^{n}-\varphi_{i-1,j}^{n}\right)/2\right]^2+\left(\varphi_{i,j+1}^{n}-\varphi_{i,j}^{n}\right)^2}}-\frac{\varphi_{i,j}^{n+1}-\varphi_{i,j-1}^{n+1}}{\sqrt{\left[\left(\varphi_{i+1,j-1}^{n}-\varphi_{i-1,j-1}^{n}\right)/2\right]^2+\left(\varphi_{i,j}^{n}-\varphi_{i,j-1}^{n}\right)^2}} \tag{4.58}$$

If we define

$$E_1=\frac{1}{\sqrt{\left(\varphi_{i+1,j}^{n}-\varphi_{i,j}^{n}\right)^2+\left[\left(\varphi_{i,j+1}^{n}-\varphi_{i,j-1}^{n}\right)/2\right]^2}}$$
$$E_2=\frac{1}{\sqrt{\left(\varphi_{i,j}^{n}-\varphi_{i-1,j}^{n}\right)^2+\left[\left(\varphi_{i-1,j+1}^{n}-\varphi_{i-1,j-1}^{n}\right)/2\right]^2}}$$
$$E_3=\frac{1}{\sqrt{\left[\left(\varphi_{i+1,j}^{n}-\varphi_{i-1,j}^{n}\right)/2\right]^2+\left(\varphi_{i,j+1}^{n}-\varphi_{i,j}^{n}\right)^2}} \tag{4.59}$$
$$E_4=\frac{1}{\sqrt{\left[\left(\varphi_{i+1,j-1}^{n}-\varphi_{i-1,j-1}^{n}\right)/2\right]^2+\left(\varphi_{i,j}^{n}-\varphi_{i,j-1}^{n}\right)^2}}$$

then the sum of the terms in (4.57) and (4.58) can be expressed as

$$\Delta_x^-\left(\frac{\Delta_x^+\varphi_{i,j}^{n+1}}{\sqrt{(\Delta_x^+\varphi_{i,j}^n)^2+(\varphi_{i,j+1}^n-\varphi_{i,j-1}^n)^2/2^2}}\right)+\Delta_y^-\left(\frac{\Delta_y^+\varphi_{i,j}^{n+1}}{\sqrt{(\varphi_{i+1,j}^n-\varphi_{i-1,j}^n)^2/2^2+(\Delta_y^+\varphi_{i,j}^n)^2}}\right)$$

$$= E_1\left(\varphi_{i+1,j}^{n+1}-\varphi_{i,j}^{n+1}\right)-E_2\left(\varphi_{i,j}^{n+1}-\varphi_{i-1,j}^{n+1}\right)+E_3\left(\varphi_{i,j+1}^{n+1}-\varphi_{i,j}^{n+1}\right)-E_4\left(\varphi_{i,j}^{n+1}-\varphi_{i,j-1}^{n+1}\right)$$

$$= E_1\varphi_{i+1,j}^{n+1}+E_2\varphi_{i-1,j}^{n+1}+E_3\varphi_{i,j+1}^{n+1}+E_4\varphi_{i,j-1}^{n+1}-\left(E_1+E_2+E_3+E_4\right)\varphi_{i,j}^{n+1}$$

Now if we let

$$F=\tau\nu\,\delta_\varepsilon\left(\varphi_{i,j}^n\right),\quad E=1+F\left(E_1+E_2+E_3+E_4\right) \tag{4.60}$$

then (4.54) becomes

$$\begin{aligned}E\,\varphi_{i,j}^{n+1} &= \varphi_{i,j}^n+F\left(E_1\varphi_{i+1,j}^{n+1}+E_2\varphi_{i-1,j}^{n+1}+E_3\varphi_{i,j+1}^{n+1}+E_4\varphi_{i,j-1}^{n+1}\right)\\ &+\tau\,\delta_\varepsilon\left(\varphi_{i,j}^n\right)\cdot\left[-\left(u_0-c_1\left(\varphi_{i,j}^n\right)\right)^2+\left(u_0-c_2\left(\varphi_{i,j}^n\right)\right)^2\right]\end{aligned} \tag{4.61}$$

The semi-implicit scheme is obtained from (4.61) by relaxing the terms with superscript $n+1$ in the RHS to n, namely,

$$\begin{aligned}\varphi_{i,j}^{n+1} &= \frac{1}{E}\Big\{\varphi_{i,j}^n+F\left(E_1\varphi_{i+1,j}^n+E_2\varphi_{i-1,j}^n+E_3\varphi_{i,j+1}^n+E_4\varphi_{i,j-1}^n\right)\\ &+\tau\,\delta_\varepsilon\left(\varphi_{i,j}^n\right)\cdot\left[-\left(u_0-c_1\left(\varphi_{i,j}^n\right)\right)^2+\left(u_0-c_2\left(\varphi_{i,j}^n\right)\right)^2\right]\Big\}\end{aligned} \tag{4.62}$$

It is important to stress that the quantities E_i, F, and E defined in (4.59) and (4.60) are matrices and multiplications and divisions in (4.62) involving these terms are *pointwise* operations.

A MATLAB function named cv.m is written to implement the Chan-Vese algorithm based on the scheme in (4.62). The code is listed as follows.

```
% To perform image segmentation by using the active contour method proposed
% in T.F.Chan and L.A.Vese , "Active contours without edges," IEEE Tarns.
% Image Processing, vol. 10, no. 2, pp. 266-277, Feb. 2001.
% The difference scheme employed here is a semi-implicit scheme obtained by a
% relaxation of the implicit scheme proposed in the above paper. The user
% is referred to ELEC 639B course notes for the details. It is noted that
% the scheme used can also be deduced by reducing the scheme proposed in the
% following paper to the one level-set function case:
% L.A.Vese and T.F.Chan,"A multiphase level set framework for image
% segmentation using the Mumforn and Shah model," Int. J. Computer Vision,
% vol. 50, no. 3, pp. 271-293, 2002.
% Inputs:   u -- image to be segmented
```

```
%       phi0 -- initial level-set function
%          dt -- step size in time
%         dt0 -- step size in time for re-initialization
%          ep -- epsilon for regularized delta function
%          nu -- parameter nu
%         K_0 -- number of iterations for re-initialization
%        K_in -- number of iterations before next re-initialization
%                (number of internal iterations)
%       K_out -- number of internal iteration rounds
%  Output: phi -- final level-set function
%  Written by W.-S. Lu, University of Victoria.
%  Last modified: March 11, 2008.
%  Examples: [phi,c1,c2,ua] = cv(obj2,p02,12,0.2,3,50,0,1,20);
%            [phi,c1,c2,ua] = cv(obj3a_cv,p03a_cv,110,0.1,4,3,0,1,100);
%            [phi,c1,c2,ua] = cv(obj3b_cv,p03b_cv,25,0.15,3,3,0,1,20);
%            [phi,c1,c2,ua] = cv(curves,p0_curves,25,0.5,4,3,0,1,20);

function [phi,c1,c2,ua] = cv(u,phi0,dt,dt0,ep,nu,K_0,K_in,K_out)

[M,N] = size(u);
yt = M:-1:1;
xt = 1:1:N;
epsi = 1e-6;
v = -1e-6:5e-7:1e-6;
won = ones(M,N);
wep = ep*won;
map = gray(256);
figure(1)
subplot(121)
imshow(u,map)
title('Image to be segmented')
pause(1)
pw = phi0;
it = 0;
figure(2)
contour(xt,yt,pw,v)
title('Zero-level of initial level-set function')
axis square
pause(0.5)
while it < K_out,
   itt = 0;
   while itt < K_in,
      H = 0.5*(won+sign(pw));
      H1 = 1 - H;
      c1 = sum(sum(u.*H))/sum(sum(H));
      c2 = sum(sum(u.*H1))/sum(sum(H1));
      delt = wep./(pi*(wep.^2+pw.^2));
      pww = ex_sym2(pw);
      pa = pww(2:(M+1),3:(N+2)); % phi_i+1,j
      pb = pww(3:(M+2),2:(N+1)); % phi_i,j+1
      pc = pww(2:(M+1),1:N);     % phi_i-1,j
      pd = pww(3:(M+2),1:N);     % phi_i-1,j+1
      pe = pww(1:M,2:(N+1));     % phi_i,j-1
      pf = pww(1:M,3:(N+2));     % phi_i+1,j-1
```

```
      pg = pww(1:M,1:N);        % phi_i-1,j-1
      E1 = 1./sqrt((pa - pw).^2 + ((pb - pe)/2).^2 + epsi);
      E2 = 1./sqrt((pw - pc).^2 + ((pd - pg)/2).^2 + epsi);
      E3 = 1./sqrt((pb - pw).^2 + ((pa - pc)/2).^2 + epsi);
      E4 = 1./sqrt((pw - pe).^2 + ((pf - pg)/2).^2 + epsi);
      fw = dt*delt;
      F = nu*fw;
      E = 1 + F.*(E1 + E2 + E3 + E4);
      T1 = E1.*pa + E2.*pc + E3.*pb + E4.*pe;
      T2 = -(u-c1).^2 + (u-c2).^2;
      T3 = pw + F.*T1 + fw.*T2;
      pw = T3./E;
      itt = itt + 1;
   end
   pw_new = re_initial(pw,dt0,K_0);
   pw = pw_new;
   it = it + 1;
   figure(2)
   contour(xt,yt,pw,v)
   title(sprintf('Zero-level after %.5g rounds of iterations',it'))
   axis square
end
phi = pw;
H = 0.5*(won+sign(phi));
H1 = 1 - H;
c1 = sum(sum(u.*H))/sum(sum(H));
c2 = sum(sum(u.*H1))/sum(sum(H1));
ua = c1*H + c2*H1;
disp('Optimal values of c1 and c2:')
[c1 c2]
figure(1)
subplot(122)
imshow(ua,map)
title('2-value piecewise constant approximation of the image')
```

Example 4.8

MATLAB function: [phi,c1,c2,ua] = cv(obj2,p02,12,0.2,3,50,0,1,20);
for function [phi,c1,c2,ua] = cv(u,phi0,dt,dt0,ep,nu,K_0,K_in,K_out)
Image: obj2.mat (size: 114×114)
Initial level-set function: $\varphi(0, x, y) = 49 - \sqrt{(x-56)^2 + (y-56)^2}$ (a discrete version is given by p02.mat)
Step size in time $\tau = 12$
No re-initialization was performed
Parameter ε: 3
Parameter ν: 50
Number of iterations after each initialization (internal iteration): 1
Number of internal iteration rounds: 20

The highest and lowest light intensity in the image were 255 and 0, respectively. The optimal values of the constants obtained from the algorithm (after 20 iterations) were $c_1 = 49.3786$ and $c_2 = 252.3867$. These values of c_1 and c_2 were used to construct the two-value piecewise-constant approximation of the image using (4.48).

Fig. 4.21 depicts the image for segmentation (on the left) and the 2-value piecewise-constant approximation of the image (on the right), and Fig. 4.22 shows the zero-level of the level-set function after various number of iterations.

On comparing with the results obtained by the CV algorithm implemented using the explicit scheme in (4.56), the implicit scheme offers considerable improvement in terms of computational efficiency. We note that this improvement in efficiency is due primarily to the fact that the semi-implicit schemes allows one to use large step size in time. As is well known, large time step size is not allowed in a typical explicit scheme as it can cause numerical divergence.

Example 4.9

MATLAB function: [phi,c1,c2,ua] = cv(obj3a_cv,p03a_cv,110,0.1,4,3,0,1,100);
for function [phi,c1,c2,ua] = cv(u,phi0,dt,dt0,ep,nu,K_0,K_in,K_out)
Image: obj3a_cv.mat (size: 120 × 120)
Initial level-set function: $\varphi(0, x, y) = 57.5 - \sqrt{(x-60.5)^2 + (y-60.5)^2}$ (a discrete version of it is given by p03a_cv.mat)
Step size in time τ = 110
No re-initialization was performed.
Parameter ε : 4
Parameter ν : 3
Number of iterations after each initialization (internal iteration): 1
Number of internal iteration rounds: 100

In the noise-free image, the light intensity for the objects was 40 and the light intensity in the background was 180. The optimal values of the constants obtained from the algorithm (after 100 iterations) were c_1 = 40.6320 and c_2 = 180.1016.

Fig. 4.23 depicts the image for segmentation (on the left) and the 2-value piecewise-constant approximation of the image (on the right), and Fig. 4.24 shows the zero-level of the level-set function after various number of iterations.

It is observed that the implicit scheme remains superior to the explicit scheme in terms of computational efficiency (100 iterations versus 600 iterations).

Example 4.10

MATLAB function: [phi,c1,c2,ua] = cv(obj3b_cv,p03b_cv,25,0.15,3,3,0,1,20);
for function [phi,c1,c2,ua] = cv(u,phi0,dt,dt0,ep,nu,K_0,K_in,K_out)
Image: obj2.mat (size: 120 × 120)
Initial level-set function: $\varphi(0, x, y) = 27.5 - \sqrt{(x-30.5)^2 + (y-30.5)^2}$ (a discrete version of it is given by p03b_cv.mat)
Step size in time τ = 25
No re-initialization was performed.
Parameter ε : 3
Parameter ν : 3
Number of iterations after each initialization (internal iteration): 1
Number of internal iteration rounds: 20

In the noise-free image, the light intensity was 0 for the triangle, 128 for the ring, and 180 for the third object, and the light intensity in the background was 255. The optimal values of the constants obtained from the algorithm (after 20 iterations) were c_1 = 253.8399 and c_2 = 151.3664.

Fig. 4.25 depicts the image for segmentation (on the left) and the 2-value piecewise-constant approximation of the image (on the right), and Fig. 4.26 shows the zero-level of the level-set function after various number of iterations.

Note that the implicit scheme demonstrates its efficiency again (20 iterations versus 320 iterations). In addition, the 2-value approximation of the given image has also been considerably improved relative to that obtained by the explicit scheme (see Figures 4. 17 and 4.25).

Example 4.11

MATLAB function: [phi,c1,c2,ua] = cv(curves,p0_curves,25,0.5,4,3,0,1,20);
for function [phi,c1,c2,ua] = cv(u,phi0,dt,dt0,ep,nu,K_0,K_in,K_out)
Image: curves.mat (size: 150 × 150)
Initial level-set function: $\varphi(0, x, y) = 50 - \sqrt{(x-74.5)^2 + (y-74.5)^2}$ (a discrete version of it is given by p0_curves.mat)
Step size in time τ = 25
No re-initialization was performed
Parameter ε : 4
Parameter ν : 3
Number of iterations after each initialization (internal iteration): 1
Number of internal iteration rounds: 20

The minimum light intensity of the thick curves was 25 and the light intensity in the background was 255. The optimal values of the constants obtained from the algorithm (after 20 iterations) were c_1 = 113.0831 and c_2 = 251.1306.

Fig. 4.27 depicts the image for segmentation (on the left) and the 2-value piecewise-constant approximation of the image (on the right), and Fig. 4.28 shows the zero-level of the level-set function after various number of iterations.

Again the implicit scheme demonstrates it improved efficiency over its explicit counterpart (20 iterations versus 240 iterations). Moreover, the 2-value approximation obtained here (see Fig. 4. 28) is much better than that obtained previously (see Fig. 4.19).

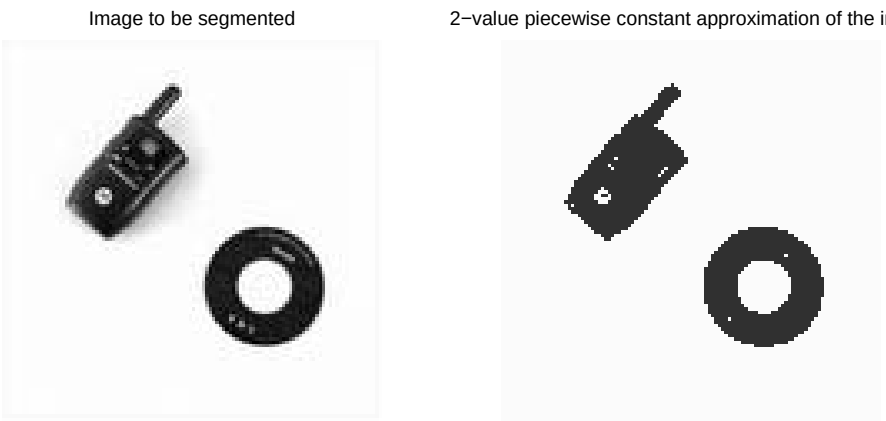


Figure 4.21 for Example 4.8.

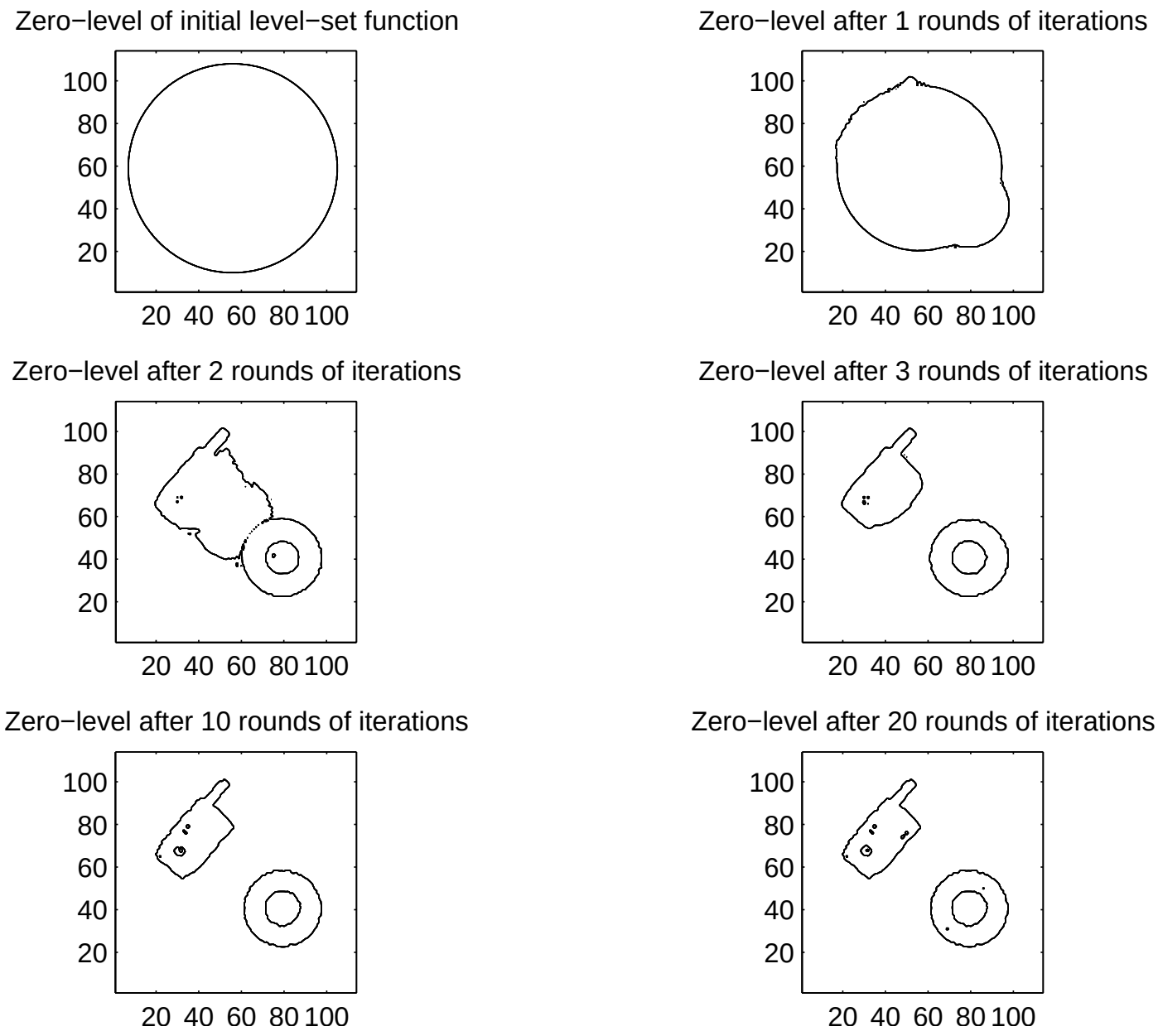


Figure 4.22 for Example 4.8.

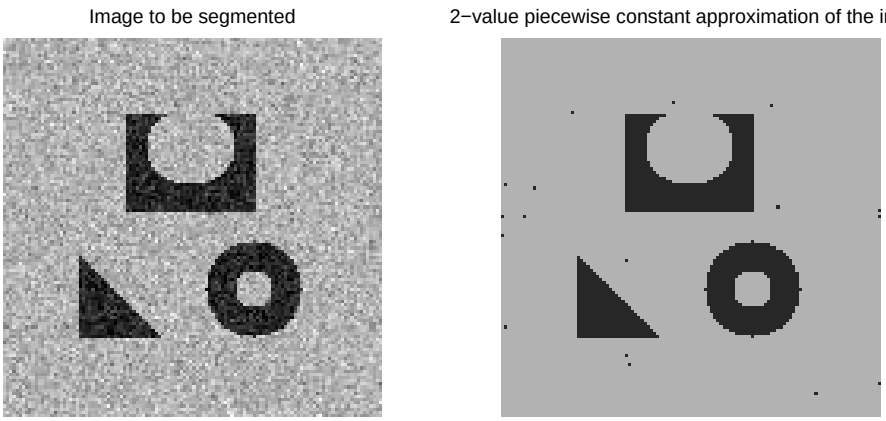


Figure 4.23 for Example 4.9.

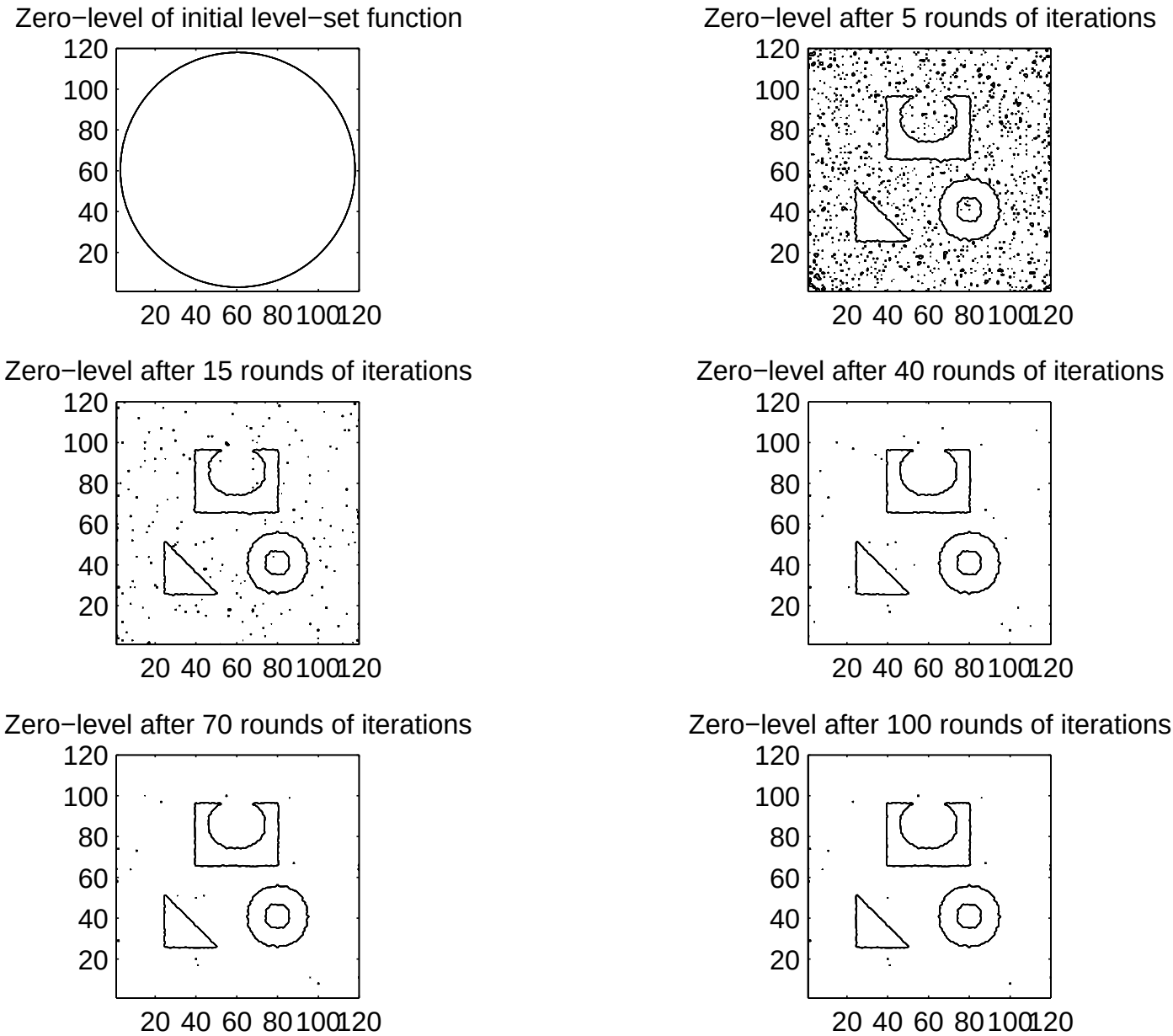


Figure 4.24 for Example 4.9.

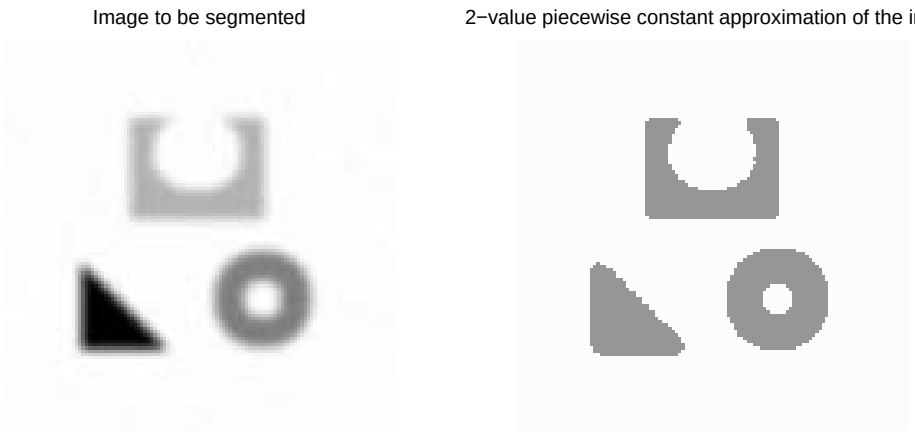


Figure 4.25 for Example 4.10.

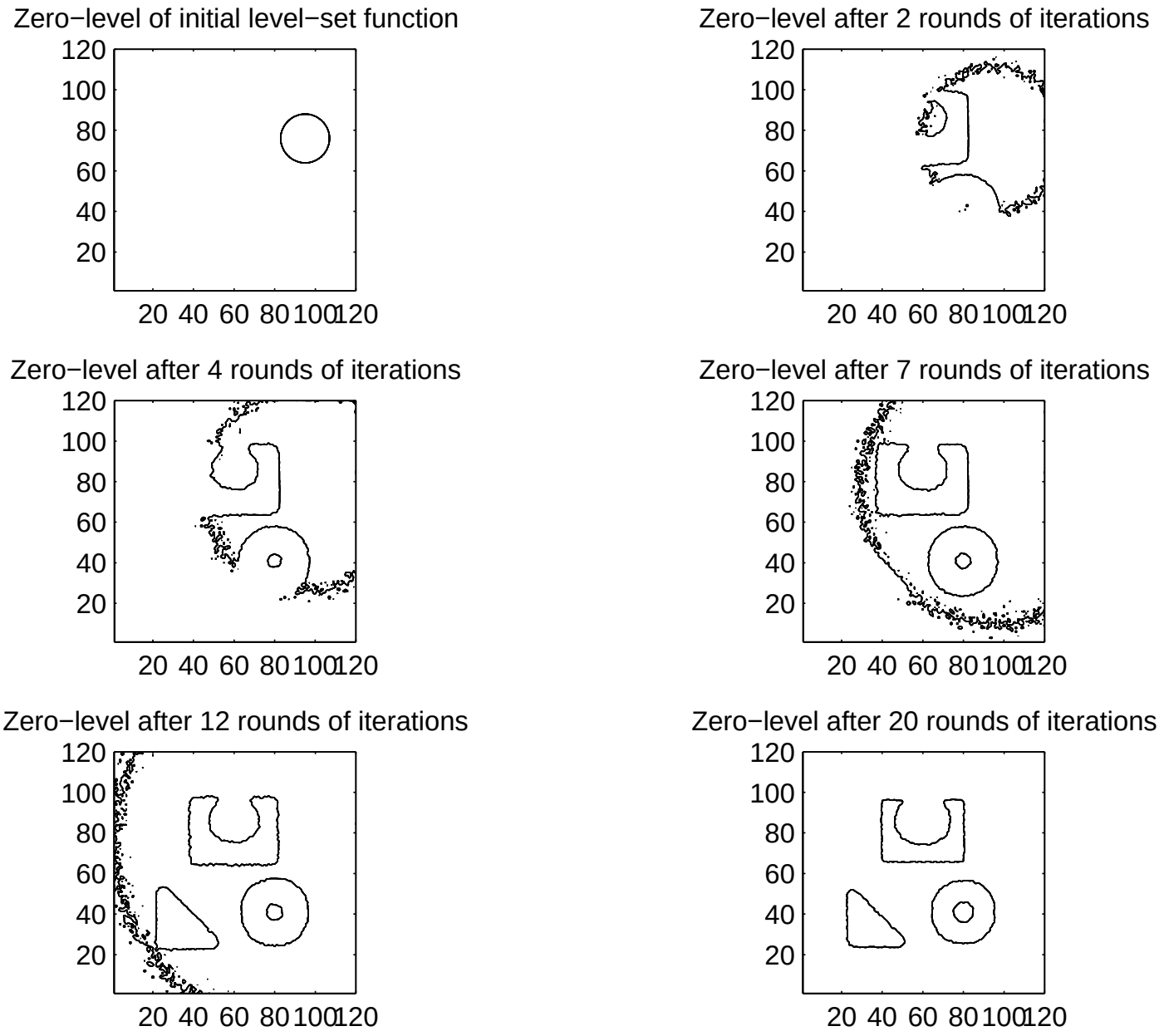


Figure 4.26 for Example 4.10.

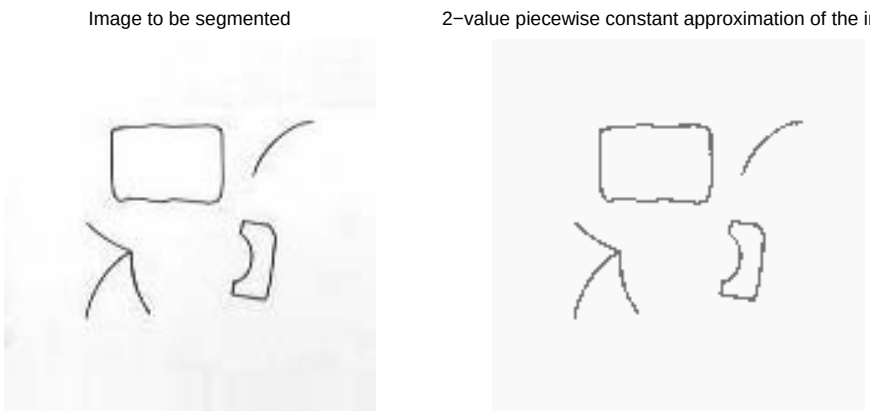


Figure 4.27 for Example 4.11.

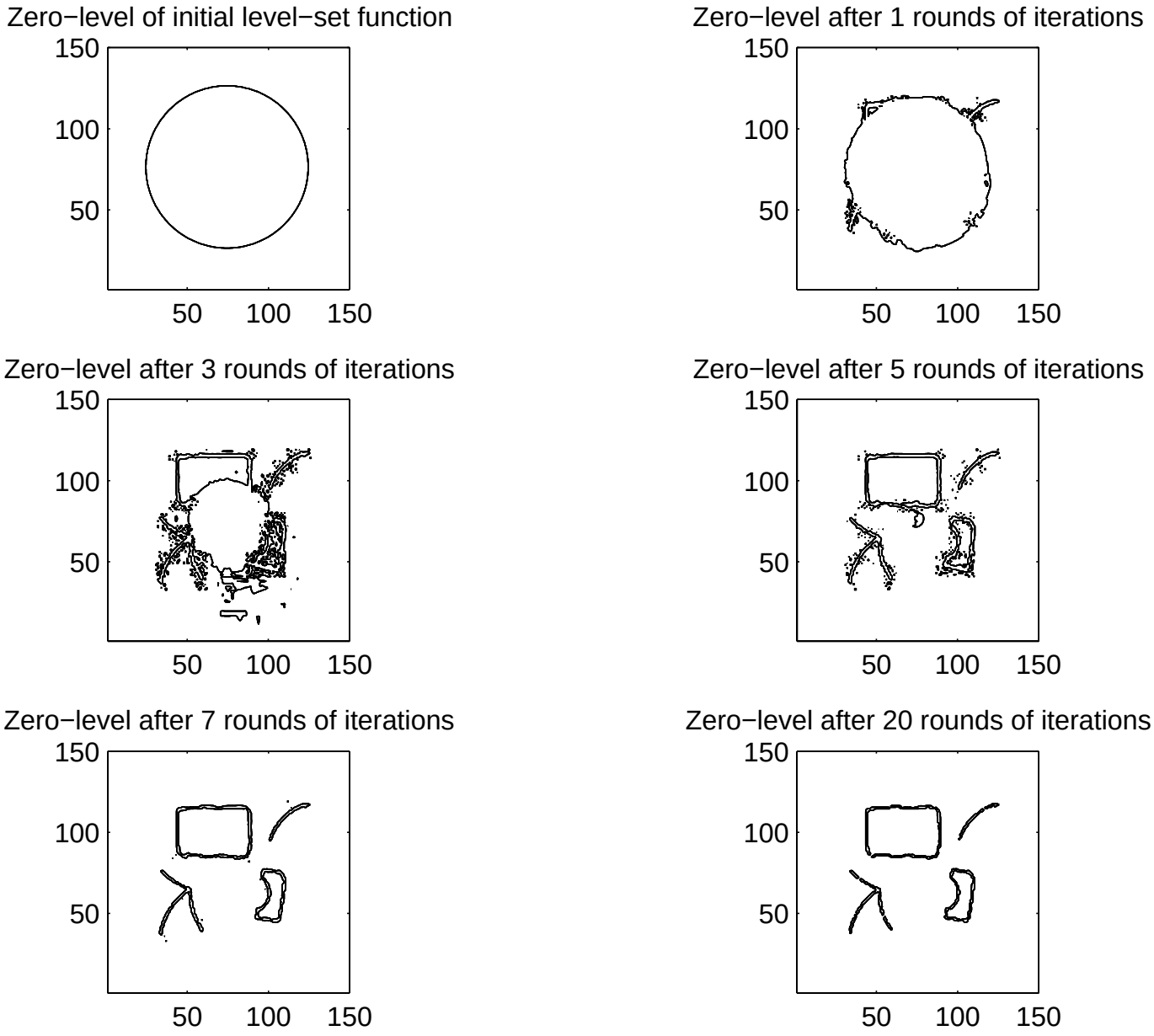


Figure 4.28 for Example 4.11.

4.4 The Chan-Vese Model in a Multiphase Level-Set Framework

4.4.1 *Introduction*

In Sec. 4.3, we introduced the Chan-Vese model where the Mumford-Shah functional is applied to find a particular partition of a given image into *two* regions with one representing the objects of interest and the other representing the background. In this way, the active contour is characterized by the boundaries of these two regions. In other words, the CV model in Sec. 4.3 is for *binary* segmentation. This model was generalized in the authors' 2002 paper

[R1] L. A. Vese and T. F. Chan, "A multiphase level set framework for image segmentation using the Mumford-Shah model," *Int. J. Computer Vision*, vol. 50, no. 3, pp. 271-293, 2002, to segment images with more than two regions. As we will see below, the main ingredients of the generalized model remain the same as in the previous CV model: the Mumford-Shah model and variational level sets. The difference is that in a multiphase framework one needs more than one level set functions for partitioning the image domain into more-than-two regions.

4.4.2 *A Multiphase Level Set Model*

For an *n*-phase piecewise constant active contour model, one needs $m = \log_2 n$ level-set functions. Indeed the case of $n = 2$ and $m = 1$ refers to the CV model studied in Sec. 4.3; for a 4-phase model, one needs $m = 2$ level-set functions; and for a 8-phase model, one needs $m = 3$ level-set functions an so on, see Fig. 4. 29 (borrowed from reference [R1]) that illustrates the case of 2-level-set-function and 3-level-set-function cases.

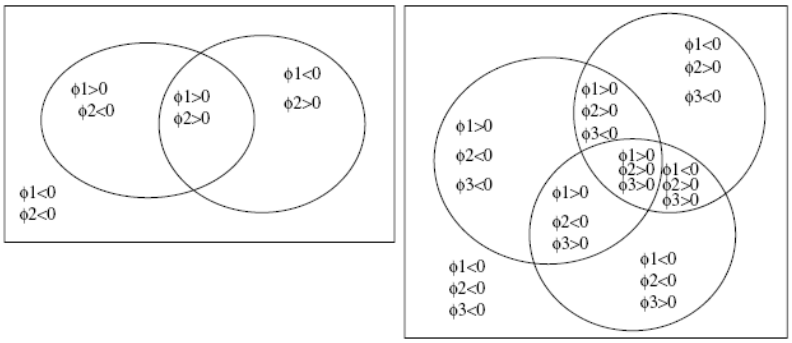


Figure 2. Left: 2 curves $\{\phi_1 = 0\} \cup \{\phi_2 = 0\}$ partition the domain into 4 regions: $\{\phi_1 > 0, \phi_2 > 0\}$, $\{\phi_1 > 0, \phi_2 < 0\}$, $\{\phi_1 < 0, \phi_2 > 0\}$, $\{\phi_1 < 0, \phi_2 < 0\}$. Right: 3 curves $\{\phi_1 = 0\} \cup \{\phi_2 = 0\} \cup \{\phi_3 = 0\}$ partition the domain into 8 regions: $\{\phi_1 > 0, \phi_2 > 0, \phi_3 > 0\}$, $\{\phi_1 > 0, \phi_2 > 0, \phi_3 < 0\}$, $\{\phi_1 > 0, \phi_2 < 0, \phi_3 > 0\}$, $\{\phi_1 > 0, \phi_2 < 0, \phi_3 < 0\}$, $\{\phi_1 < 0, \phi_2 > 0, \phi_3 > 0\}$, $\{\phi_1 < 0, \phi_2 > 0, \phi_3 < 0\}$, $\{\phi_1 < 0, \phi_2 < 0, \phi_3 > 0\}$, $\{\phi_1 < 0, \phi_2 < 0, \phi_3 < 0\}$.

Figure 4.29 (from reference [R1]).

Consider a general *m*-level-set-function case and denote the level-set functions by $\{\varphi_i(x, y) \;\; i = 1, 2, \ldots, m\}$. For the cases with $m > 1$, we introduce *vector level-set function* $\Phi = (\varphi_1, \varphi_2, \cdots, \varphi_m)$ and *vector Heaviside function* $H(\Phi) = (H(\varphi_1), H(\varphi_2), \cdots, H(\varphi_m))$ with each component assuming value 1 or 0. Given an image, a segment (or a phase) of the image according to level-set functions $\{\varphi_i(x, y) \;\; i = 1, 2, \ldots, m\}$ is defined as a region where the vector Heaviside function $\Phi = (\varphi_1, \varphi_2, \cdots, \varphi_m)$ is identical at every point (x, y) in the region. In other words, each segment is characterized by the set

$$\{(x, y) \mid H(\Phi(x, y)) = \text{constant vector} \in H(\Phi(\Omega))\} \tag{4.63}$$

For example, for the 2-phase case shown in the left plot in Fig. 4.29 we see four segments (phases) that correspond to the vector Heaviside function being equal to values (1, 1), (1, 0), (0, 1) and (0, 0), respectively.

In the present framework, the set of object boundaries, denoted by Γ, are represented by the union of the zero-level sets of functions $\{\varphi_i(x, y)\ \ i = 1, 2, \ldots, m\}$. As can be seen from Fig. 4.29, this representation has a problem because the union of the zero-level sets may count some (usually small) part of Γ more than once. However this should not be a serious concern because the term representing the curve length is not as critical as the fidelity terms.

We denote each segment (also called class or region) in (4.63) by I_i for i = 1, 2, …, $n = 2^m$ and denote the characteristic function of class I_i by χ_{I_i} which assumes value 1 for a point (x, y) inside I_i and value 0 for a point (x, y) outsize I_i. We also introduce a constant vector $c = (c_1, c_2, \ldots, c_n)$ where c_i = mean(u_0) in class I_i. Under these notation the Mumford-Shah functional (energy) can be expressed as

$$J_{MS}(c, \Phi) = \sum_{1 \le i \le n = 2^m} \int_\Omega (u_0 - c_i)^2 \chi_{I_i}\, dx\, dy + \frac{\nu}{2} \sum_{1 \le i \le n = 2^m} \int_\Omega |\nabla \chi_{I_i}|\, dx\, dy \qquad (4.64)$$

The above model may be simplified by replacing the length term by sum of the lengths of the zero-level sets of $\{\varphi_i(x, y)\ \ i = 1, 2, \ldots, m\}$ which, as mentioned before, may cause some inaccuracy but shall not be a serious concern in many application scenarios. In doing so, we have a simplified functional

$$J_n(c, \Phi) = \sum_{1 \le i \le n = 2^m} \int_\Omega (u_0 - c_i)^2 \chi_{I_i}\, dx\, dy + \frac{\nu}{2} \sum_{1 \le i \le n = 2^m} \int_\Omega |\nabla H(\varphi_i)|\, dx\, dy \qquad (4.65)$$

In what follows we examine the model in (4.65) for m = 2 (i.e. n = 4), a case that is particularly interesting because it is significantly broader than the 2-phase case considered in Sec. 4.3, yet still feasible to handle. Indeed we can then express each of the four characteristic functions in the fidelity terms in (4.65) in terms of the associated level-set functions:

$$\begin{aligned} J_4(c, \Phi) &= \int_\Omega (u_0 - c_{11})^2 H(\varphi_1) H(\varphi_2)\, dx\, dy \\ &+ \int_\Omega (u_0 - c_{10})^2 H(\varphi_1)(1 - H(\varphi_2))\, dx\, dy \\ &+ \int_\Omega (u_0 - c_{01})^2 (1 - H(\varphi_1)) H(\varphi_2)\, dx\, dy \\ &+ \int_\Omega (u_0 - c_{00})^2 (1 - H(\varphi_1))(1 - H(\varphi_2))\, dx\, dy \\ &+ \nu \int_\Omega |\nabla H(\varphi_1)|\, dx\, dy + \nu \int_\Omega |\nabla H(\varphi_2)|\, dx\, dy \end{aligned} \qquad (4.66)$$

Once the problem of minimizing the functional J_4 in (4.66) is solved, the solution vectors $c = (c_{11}, c_{10}, c_{01}, c_{00})$ and $\Phi = (\varphi_1, \varphi_2)$ are used to construct a 4-phase approximation $u(x, y)$ of image $u_0(x, y)$ as

$$\begin{aligned} u(x, y) &= c_{11} H(\varphi_1) H(\varphi_2) + c_{10} H(\varphi_1)(1 - H(\varphi_2)) \\ &+ c_{01}(1 - H(\varphi_1)) H(\varphi_2) + c_{00}(1 - H(\varphi_1))(1 - H(\varphi_2)) \end{aligned} \qquad (4.67)$$

4.4.3 *The Euler-Lagrange Equations*

Given initial level-set functions

$$\varphi_1(0,x,y)=\varphi_{1,0}(x,y) \quad \text{and} \quad \varphi_2(0,x,y)=\varphi_{2,0}(x,y) \tag{4.68a}$$

we evaluate constants

$$\begin{aligned} c_{11} &= \text{mean}(u_0) \text{ in } \{(x,y):\varphi_1(x,y)>0,\ \varphi_2(x,y)>0\} \\ c_{10} &= \text{mean}(u_0) \text{ in } \{(x,y):\varphi_1(x,y)>0,\ \varphi_2(x,y)<0\} \\ c_{01} &= \text{mean}(u_0) \text{ in } \{(x,y):\varphi_1(x,y)<0,\ \varphi_2(x,y)>0\} \\ c_{00} &= \text{mean}(u_0) \text{ in } \{(x,y):\varphi_1(x,y)<0,\ \varphi_2(x,y)<0\} \end{aligned} \tag{4.68b}$$

and solve the coupled Euler-Lagrange equations that are embedded in time *t*:

$$\begin{aligned} \frac{\partial\varphi_1}{\partial t} &= \delta_\varepsilon(\varphi_1)\left\{\nu\,\text{div}\left(\frac{\nabla\varphi_1}{|\nabla\varphi_1|}\right)-\left[((u_0-c_{11})^2-(u_0-c_{01})^2)H(\varphi_2)\right.\right. \\ &\left.\left.+((u_0-c_{10})^2-(u_0-c_{00})^2)(1-H(\varphi_2))\right]\right\} \\ \frac{\partial\varphi_2}{\partial t} &= \delta_\varepsilon(\varphi_2)\left\{\nu\,\text{div}\left(\frac{\nabla\varphi_2}{|\nabla\varphi_2|}\right)-\left[((u_0-c_{11})^2-(u_0-c_{10})^2)H(\varphi_1)\right.\right. \\ &\left.\left.+((u_0-c_{01})^2-(u_0-c_{00})^2)(1-H(\varphi_1))\right]\right\} \end{aligned} \tag{4.68c}$$

with Neumann boundary conditions

$$\left.\frac{\delta_\varepsilon(\varphi_1)}{|\nabla\varphi_1|}\frac{\partial\varphi_1}{\partial N}\right|_{\partial\Omega}=0 \quad \text{and} \quad \left.\frac{\delta_\varepsilon(\varphi_2)}{|\nabla\varphi_2|}\frac{\partial\varphi_2}{\partial N}\right|_{\partial\Omega}=0 \tag{4.68d}$$

4.4.4 *A Semi-Implicit Scheme for (4.68)*

A semi-implicit scheme for Eq. (4.68) is proposed by Vese and Chan in [R1]. The scheme is now given below, where for notation simplicity we have assumed $h = 1$.

The regularized functions $H_\varepsilon(z)$ and $\delta_\varepsilon(z)$ involved in the scheme were defined by (4.49) and (4.50) are given by

$$H_\varepsilon(z)=\frac{1}{2}\left(1+\frac{2}{\pi}\arctan\left(\frac{z}{\varepsilon}\right)\right)$$

$$\delta_\varepsilon=\frac{d}{dz}H_\varepsilon(z)=\frac{\varepsilon}{\pi(z^2+\varepsilon^2)}$$

The constants given in (4.86b) can be evaluated in terms of the regularized Heaviside functions as

$$
\begin{aligned}
c_{11} &= \frac{\int_\Omega u_0(x,y) H_\varepsilon(\varphi_1) H_\varepsilon(\varphi_2)\,dx\,dy}{\int_\Omega H_\varepsilon(\varphi_1) H_\varepsilon(\varphi_2)\,dx\,dy} \\
c_{10} &= \frac{\int_\Omega u_0(x,y) H_\varepsilon(\varphi_1)\big(1-H_\varepsilon(\varphi_2)\big)\,dx\,dy}{\int_\Omega H_\varepsilon(\varphi_1)\big(1-H_\varepsilon(\varphi_2)\big)\,dx\,dy} \\
c_{01} &= \frac{\int_\Omega u_0(x,y)\big(1-H_\varepsilon(\varphi_1)\big) H_\varepsilon(\varphi_2)\,dx\,dy}{\int_\Omega \big(1-H_\varepsilon(\varphi_1)\big) H_\varepsilon(\varphi_2)\,dx\,dy} \\
c_{00} &= \frac{\int_\Omega u_0(x,y)\big(1-H_\varepsilon(\varphi_1)\big)\big(1-H_\varepsilon(\varphi_2)\big)\,dx\,dy}{\int_\Omega \big(1-H_\varepsilon(\varphi_1)\big)\big(1-H_\varepsilon(\varphi_2)\big)\,dx\,dy}
\end{aligned}
\tag{4.69}
$$

The scheme for the first equation in (4.68c) is realized by first evaluating the four matrices

$$
\begin{aligned}
E_1 &= \frac{1}{\sqrt{\big(\varphi^n_{1,i+1,j}-\varphi^n_{1,i,j}\big)^2+\Big[\big(\varphi^n_{1,i,j+1}-\varphi^n_{1,i,j-1}\big)/2\Big]^2}} \\
E_2 &= \frac{1}{\sqrt{\big(\varphi^n_{1,i,j}-\varphi^n_{1,i-1,j}\big)^2+\Big[\big(\varphi^n_{1,i-1,j+1}-\varphi^n_{1,i-1,j-1}\big)/2\Big]^2}} \\
E_3 &= \frac{1}{\sqrt{\Big[\big(\varphi^n_{1,i+1,j}-\varphi^n_{1,i-1,j}\big)/2\Big]^2+\big(\varphi^n_{1,i,j+1}-\varphi^n_{1,i,j}\big)^2}} \\
E_4 &= \frac{1}{\sqrt{\Big[\big(\varphi^n_{1,i+1,j-1}-\varphi^n_{1,i-1,j-1}\big)/2\Big]^2+\big(\varphi^n_{1,i,j}-\varphi^n_{1,i,j-1}\big)^2}}
\end{aligned}
\tag{4.70a}
$$

then letting

$$
F=\tau\nu\,\delta_\varepsilon\big(\varphi^n_{1,i,j}\big),\quad E=1+F\,(E_1+E_2+E_3+E_4) \tag{4.70b}
$$

and computing

$$
\begin{aligned}
\varphi^{n+1}_{1,i,j} &= \frac{1}{E}\Big\{\varphi^n_{1,i,j}+F\Big(E_1\,\varphi^n_{1,i+1,j}+E_2\,\varphi^n_{1,i-1,j}+E_3\,\varphi^n_{1,i,j+1}+E_4\,\varphi^n_{1,i,j-1}\Big) \\
&+\tau\,\delta_\varepsilon\big(\varphi^n_{1,i,j}\big)\cdot\Big[-\big(u_0-c_{11}\big)^2 H_\varepsilon\big(\varphi^n_{2,i,j}\big)-\big(u_0-c_{10}\big)^2\big(1-H_\varepsilon\big(\varphi^n_{2,i,j}\big)\big) \\
&+\big(u_0-c_{01}\big)^2 H_\varepsilon\big(\varphi^n_{2,i,j}\big)+\big(u_0-c_{00}\big)^2\big(1-H_\varepsilon\big(\varphi^n_{2,i,j}\big)\big)\Big]\Big\}
\end{aligned}
\tag{4.70c}
$$

Similarly, the scheme for the second equation in (4.68c) is realized by first evaluating the four matrices

$$G_1 = \frac{1}{\sqrt{\left(\varphi^n_{2,i+1,j} - \varphi^n_{2,i,j}\right)^2 + \left[\left(\varphi^n_{2,i,j+1} - \varphi^n_{2,i,j-1}\right)/2\right]^2}}$$

$$G_2 = \frac{1}{\sqrt{\left(\varphi^n_{2,i,j} - \varphi^n_{2,i-1,j}\right)^2 + \left[\left(\varphi^n_{2,i-1,j+1} - \varphi^n_{2,i-1,j-1}\right)/2\right]^2}} \qquad (4.70d)$$

$$G_3 = \frac{1}{\sqrt{\left[\left(\varphi^n_{2,i+1,j} - \varphi^n_{2,i-1,j}\right)/2\right]^2 + \left(\varphi^n_{2,i,j+1} - \varphi^n_{2,i,j}\right)^2}}$$

$$G_4 = \frac{1}{\sqrt{\left[\left(\varphi^n_{2,i+1,j-1} - \varphi^n_{2,i-1,j-1}\right)/2\right]^2 + \left(\varphi^n_{2,i,j} - \varphi^n_{2,i,j-1}\right)^2}}$$

then letting

$$L = \tau\,\nu\,\delta_\varepsilon\left(\varphi^n_{2,i,j}\right), \quad G = 1 + L\,(G_1 + G_2 + G_3 + G_4) \qquad (4.70e)$$

and computing

$$\begin{aligned}\varphi^{n+1}_{2,i,j} = {} & \frac{1}{G}\Big\{\varphi^n_{2,i,j} + L\left(G_1\,\varphi^n_{2,i+1,j} + G_2\,\varphi^n_{2,i-1,j} + G_3\,\varphi^n_{2,i,j+1} + G_4\,\varphi^n_{2,i,j-1}\right) \\ & + \tau\,\delta_\varepsilon\left(\varphi^n_{2,i,j}\right)\cdot\Big[-\left(u_0 - c_{11}\right)^2 H_\varepsilon\left(\varphi^n_{1,i,j}\right) + \left(u_0 - c_{10}\right)^2 H_\varepsilon\left(\varphi^n_{1,i,j}\right) \\ & - \left(u_0 - c_{01}\right)^2\left(1 - H_\varepsilon\left(\varphi^n_{1,i,j}\right)\right) + \left(u_0 - c_{00}\right)^2\left(1 - H_\varepsilon\left(\varphi^n_{1,i,j}\right)\right)\Big]\Big\}\end{aligned} \qquad (4.70f)$$

4.4.5 *MATLAB Code and Examples*

The main MATLAB function that implements the above algorithm is cv4.m, which is listed below. The only other function required is ex_sym1.m which was listed earlier.

1. Function cv4.m

```
% To perform 4-phase image segmentation by using the active contour method
% proposed in L.A.Vese and T. F. Chan , "A multiphase level set framework
% for image segmentation using Mumford and Shah model," Int. J. Computer
% Vision, vol. 50, no. 3, pp. 271-293, 2002.
% The difference scheme employed here is a semi-implicit scheme also
% proposed in the above paper.
% Inputs:       u -- image to be segmented
%              p10 -- initial first level-set function
%              p20 -- initial second level-set function
%               dt -- step size in time
%               ep -- epsilon for regularized delta function
%               nu -- parameter nu
%                K -- number of internal iterations
% Outputs:          p1 -- final first level-set function
%                   p2 -- final second level-set function
%   c11,c10,c01,c00 -- optimal constants
```

```
%                   ua -- optimal 4-phase approximation of the image.
% Written by W.-S. Lu, University of Victoria.
% Last modified: March 15, 2008.
% Examples: [p1,p2,c11,c10,c01,c00,ua] = cv4(obj3_vc07,p01_vc,p02_vc,10,1,100,150);
%           [p1,p2,c11,c10,c01,c00,ua] = cv4(triplejn,p01j_vc,p02j_vc,0.1,1,600,150);
%           [p1,p2,c11,c10,c01,c00,ua] = cv4(ct_scan,p01_ct,p02_ct,10,1,25,200);

function [p1,p2,c11,c10,c01,c00,ua] = cv4(u,p10,p20,dt,ep,nu,K)

[M,N] = size(u);
yt = M:-1:1;
xt = 1:1:N;
epsi = 1e-6;
v = -1e-6:5e-7:1e-6;
pi1 = 1/pi;
pi2 = 2*pi1;
won = ones(M,N);
wep = ep*won;
map = gray(256);
figure(1)
subplot(121)
imshow(u,map)
title('Image to be segmented')
pause(1)
pw1 = p10;
pw2 = p20;
it = 0;
figure(2)
contour(xt,yt,pw1,v)
title('Zero-level of 1st initial level-set function')
axis square
figure(3)
contour(xt,yt,pw2,v)
title('Zero-level of 2nd initial level-set function')
axis square
pause(0.5)
while it < K,
   H1 = 0.5*(won+pi2*atan(pw1/ep));
   H1i = 1 - H1;
   H2 = 0.5*(won+pi2*atan(pw2/ep));
   H2i = 1 - H2;
   h11 = H1.*H2;
   h10 = H1.*H2i;
   h01 = H1i.*H2;
   h00 = H1i.*H2i;
   c11 = sum(sum(u.*h11))/sum(sum(h11));
   c10 = sum(sum(u.*h10))/sum(sum(h10));
   c01 = sum(sum(u.*h01))/sum(sum(h01));
   c00 = sum(sum(u.*h00))/sum(sum(h00));
   ua = c11*h11 + c10*h10 + c01*h01 + c00*h00;
   figure(1)
   subplot(122)
   imshow(ua,map)
   title(sprintf('4-value approximation after %.5g iterations',it))
```

```
    delt1 = wep./(pi*(wep.^2+pw1.^2));
    delt2 = wep./(pi*(wep.^2+pw2.^2));
    v11 = (u - c11).^2;
    v10 = (u - c10).^2;
    v01 = (u - c01).^2;
    v00 = (u - c00).^2;
    pww1 = ex_sym1(pw1);
    pww2 = ex_sym1(pw2);
    pa = pww1(2:(M+1),3:(N+2)); % pw1_i+1,j
    pb = pww1(3:(M+2),2:(N+1)); % pw1_i,j+1
    pc = pww1(2:(M+1),1:N);     % pw1_i-1,j
    pd = pww1(3:(M+2),1:N);     % pw1_i-1,j+1
    pe = pww1(1:M,2:(N+1));     % pw1_i,j-1
    pf = pww1(1:M,3:(N+2));     % pw1_i+1,j-1
    pg = pww1(1:M,1:N);         % pw1_i-1,j-1
    qa = pww2(2:(M+1),3:(N+2)); % pw2_i+1,j
    qb = pww2(3:(M+2),2:(N+1)); % pw2_i,j+1
    qc = pww2(2:(M+1),1:N);     % pw2_i-1,j
    qd = pww2(3:(M+2),1:N);     % pw2_i-1,j+1
    qe = pww2(1:M,2:(N+1));     % pw2_i,j-1
    qf = pww2(1:M,3:(N+2));     % pw2_i+1,j-1
    qg = pww2(1:M,1:N);         % pw2_i-1,j-1
    E1 = 1./sqrt((pa - pw1).^2 + ((pb - pe)/2).^2 + epsi);
    E2 = 1./sqrt((pw1 - pc).^2 + ((pd - pg)/2).^2 + epsi);
    E3 = 1./sqrt((pb - pw1).^2 + ((pa - pc)/2).^2 + epsi);
    E4 = 1./sqrt((pw1 - pe).^2 + ((pf - pg)/2).^2 + epsi);
    fw1 = dt*delt1;
    F = nu*fw1;
    E = 1 + F.*(E1 + E2 + E3 + E4);
    T1 = E1.*pa + E2.*pc + E3.*pb + E4.*pe;
    T2 = (v01-v11).*H2 + (v00- v10).*H2i;
    T3 = pw1 + F.*T1 + fw1.*T2;
    pw1 = T3./E;
    G1 = 1./sqrt((qa - pw2).^2 + ((qb - qe)/2).^2 + epsi);
    G2 = 1./sqrt((pw2 - qc).^2 + ((qd - qg)/2).^2 + epsi);
    G3 = 1./sqrt((qb - pw2).^2 + ((qa - qc)/2).^2 + epsi);
    G4 = 1./sqrt((pw2 - qe).^2 + ((qf - qg)/2).^2 + epsi);
    fw2 = dt*delt2;
    L = nu*fw2;
    G = 1 + L.*(G1 + G2 + G3 + G4);
    S1 = G1.*qa + G2.*qc + G3.*qb + G4.*qe;
    S2 = (v10 - v11).*H1 + (v00 - v01).*H1i;
    S3 = pw2 + L.*S1 + fw2.*S2;
    pw2 = S3./G;
    it = it + 1;
    figure(2)
    contour(xt,yt,pw1,v)
    title(sprintf('1st Zero-level after %.5g iterations',it'))
    axis square
    figure(3)
    contour(xt,yt,pw2,v)
    title(sprintf('2nd Zero-level after %.5g iterations',it'))
    axis square
end
```

```
p1 = pw1;
p2 = pw2;
H1 = 0.5*(won+pi2*atan(p1/ep));
H1i = 1 - H1;
H2 = 0.5*(won+pi2*atan(p2/ep));
H2i = 1 - H2;
h11 = H1.*H2;
h10 = H1.*H2i;
h01 = H1i.*H2;
h00 = H1i.*H2i;
c11 = sum(sum(u.*h11))/sum(sum(h11));
c10 = sum(sum(u.*h10))/sum(sum(h10));
c01 = sum(sum(u.*h01))/sum(sum(h01));
c00 = sum(sum(u.*h00))/sum(sum(h00));
ua = c11*h11 + c10*h10 + c01*h01 + c00*h00;
disp('Optimal values of c11, c10, c01, and c00:')
[c11 c10 c01 c00]
figure(1)
subplot(122)
imshow(ua,map)
title('4-value piecewise constant approximation of the image')
```

Example 4.12

- MATLAB function: [p1,p2,c11,c10,c01,c00,ua] = cv4(obj3_vc07,p01_vc,p02_vc,10,1,100,150);
  for function [p1,p2,c11,c10,c01,c00,ua] = cv4(u,p10,p20,dt,ep,nu,K)
- Image: obj3_vc07.mat (size: 120 × 120)
- Initial level-set functions: $\varphi_1(0,x,y) = 28 - \sqrt{(x-30)^2 + (y-30)^2}$ and $\varphi_2(0,x,y) = 28 - \sqrt{(x-90)^2 + (y-90)^2}$ (their discrete versions are given by p01_vc07.mat and p02_vc07.mat)
- Step size in time τ = 10
- No re-initialization was performed
- Parameter ε : 1
- Parameter ν : 100
- Number of iterations: 150

The light intensity of the objects was 30 for the triangle, 110 for the ring, 220 for the third object, and 160 for the background. The optimal values of the constants obtained from the algorithm (after 150 iterations) were c_{11} = 109.2690, c_{10} = 217.7539, c_{01} = 37.0268, and c_{00} = 159.8958.

Fig. 4.29 depicts the image for segmentation (on the left) and the 4-value piecewise-constant approximation of the image (on the right), and Figs. 4.30 and 4.31 show the zero-level of the two level-set functions after various number of iterations. It is noted that the 4-phase approximation is able to match the light intensity of the input image with good accuracy.

Example 4.13

- MATLAB function: [p1,p2,c11,c10,c01,c00,ua] = cv4(triplejn,p01j_vc,p02j_vc,0.1,1,600,150);
  for function [p1,p2,c11,c10,c01,c00,ua] = cv4(u,p10,p20,dt,ep,nu,K)

• Image: triplejn.mat (size: 64 × 64)
• Initial level-set functions: $\varphi_1(0, x, y) = 10 - \sqrt{(x-16)^2 + (y-16)^2}$ and
$\varphi_2(0, x, y) = 10 - \sqrt{(x-40)^2 + (y-40)^2}$ (their discrete versions are given by p01j_vc.mat and p02j_vc.mat)
• Step size in time τ = 0.1
• No re-initialization was performed
• Parameter ε : 1
• Parameter ν : 600
• Number of iterations: 150

The light intensity was 40 for the lower-left block, 120 for the upper-left block, and 220 for the block on the right. The optimal values of the constants obtained from the algorithm (after 150 iterations) were c_{11} = 184.2156, c_{10} = 40.0522, c_{01} = 219.7967, and c_{00} = 130.5431.

Fig. 4.32 depicts the image for segmentation (on the left) and the 4-value piecewise-constant approximation of the image (on the right), and Figs. 4.33 and 4.34 show the zero-level of the two level-set functions after various number of iterations. We see that the algorithm is able to segment images with triple junctions.

Example 4.14

• MATLAB function: [p1,p2,c11,c10,c01,c00,ua] = cv4(ct_scan,p01_ct,p02_ct,30,1,25,500);
for function [p1,p2,c11,c10,c01,c00,ua] = cv4(u,p10,p20,dt,ep,nu,K)
• Image: ct_scan.mat (size: 512 × 512)
• Initial level-set functions: $\varphi_1(0, x, y) = 60 - \sqrt{(x-168)^2 + (y-168)^2}$ and
$\varphi_2(0, x, y) = 60 - \sqrt{(x-344)^2 + (y-344)^2}$ (their discrete versions are given by p01_ct.mat and p02_ct.mat)
• Step size in time τ = 30
• No re-initialization was performed
• Parameter ε : 1
• Parameter ν : 25
• Number of iterations: 500

The optimal values of the constants obtained from the algorithm (after 300 iterations) were c_{11} = 234.9715, c_{10} = 153.0462, c_{01} = 167.9666, and c_{00} = 1.8149.

Fig. 4.35 depicts the image for segmentation (on the left) and the 4-value piecewise-constant approximation of the image (on the right).

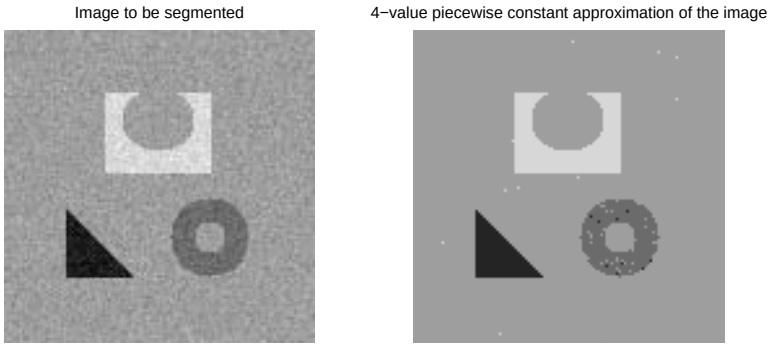


Figure 4.29 for Example 4.12

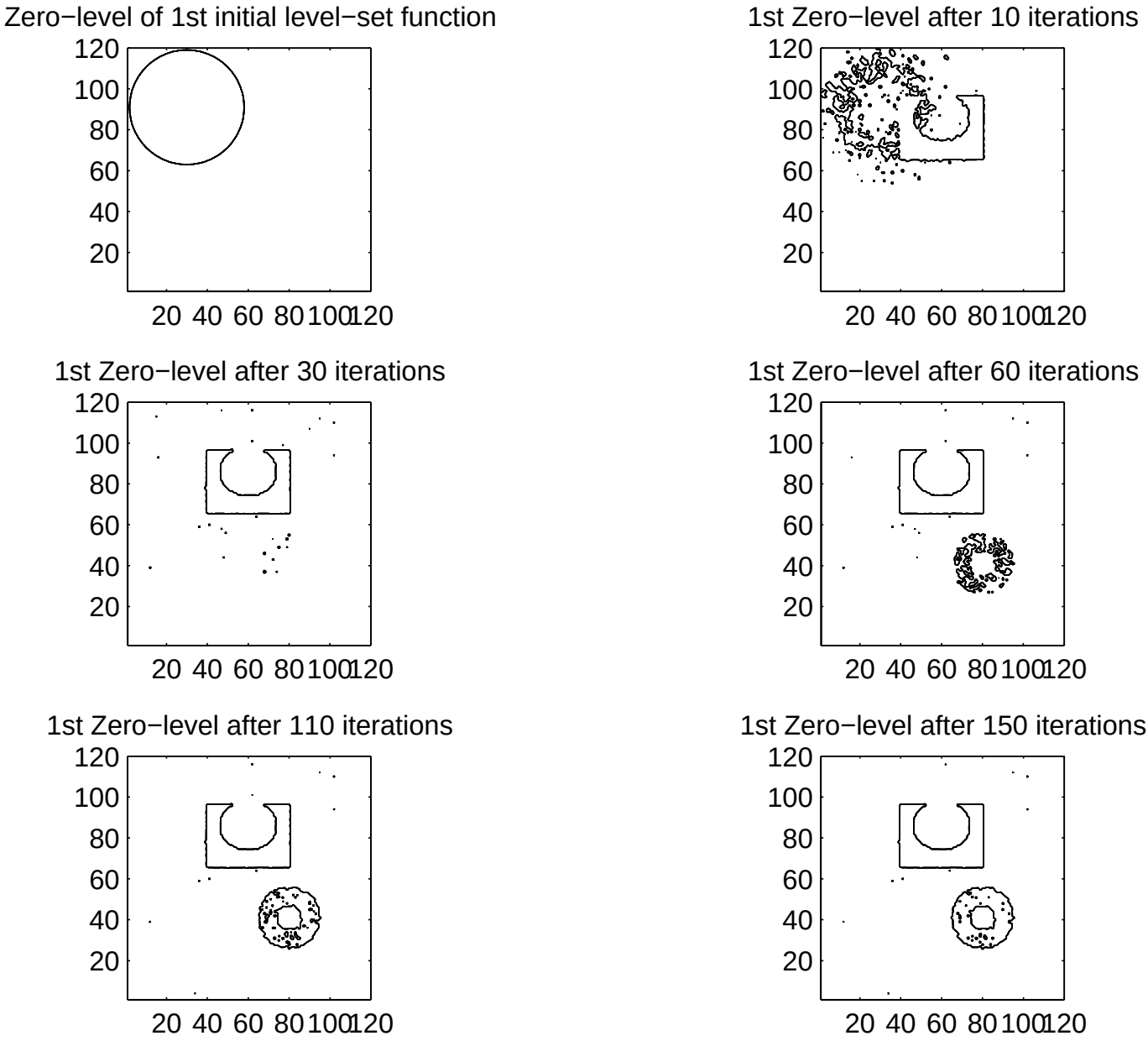


Figure 4.30 For Example 4.12

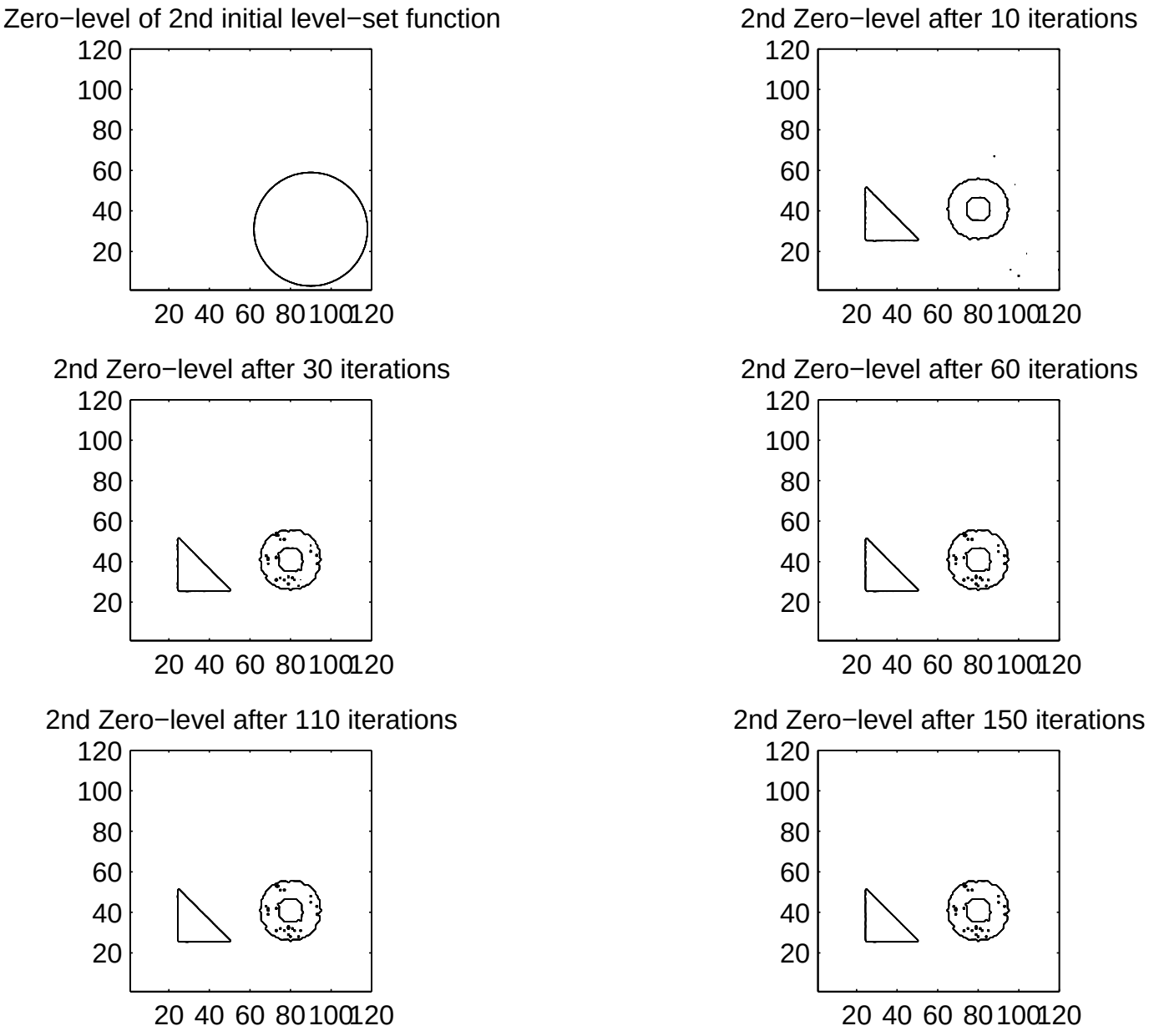


Figure 4.31 For Example 4.12.

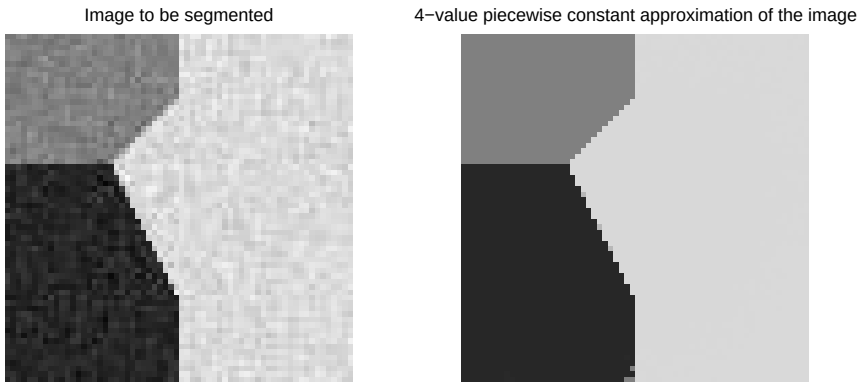


Figure 4.32 For Example 4.13.

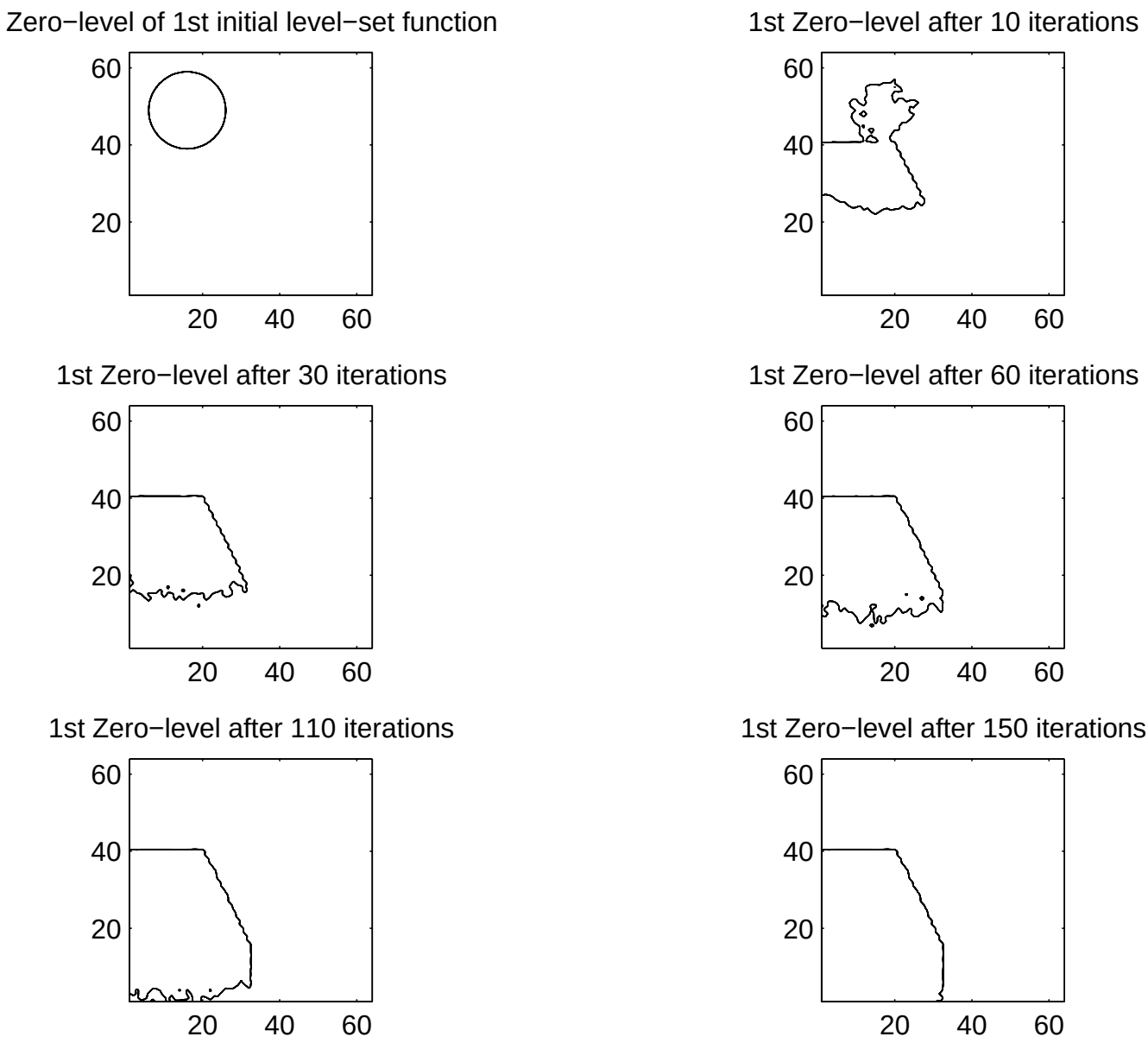


Figure 4.33 For Example 4.13.

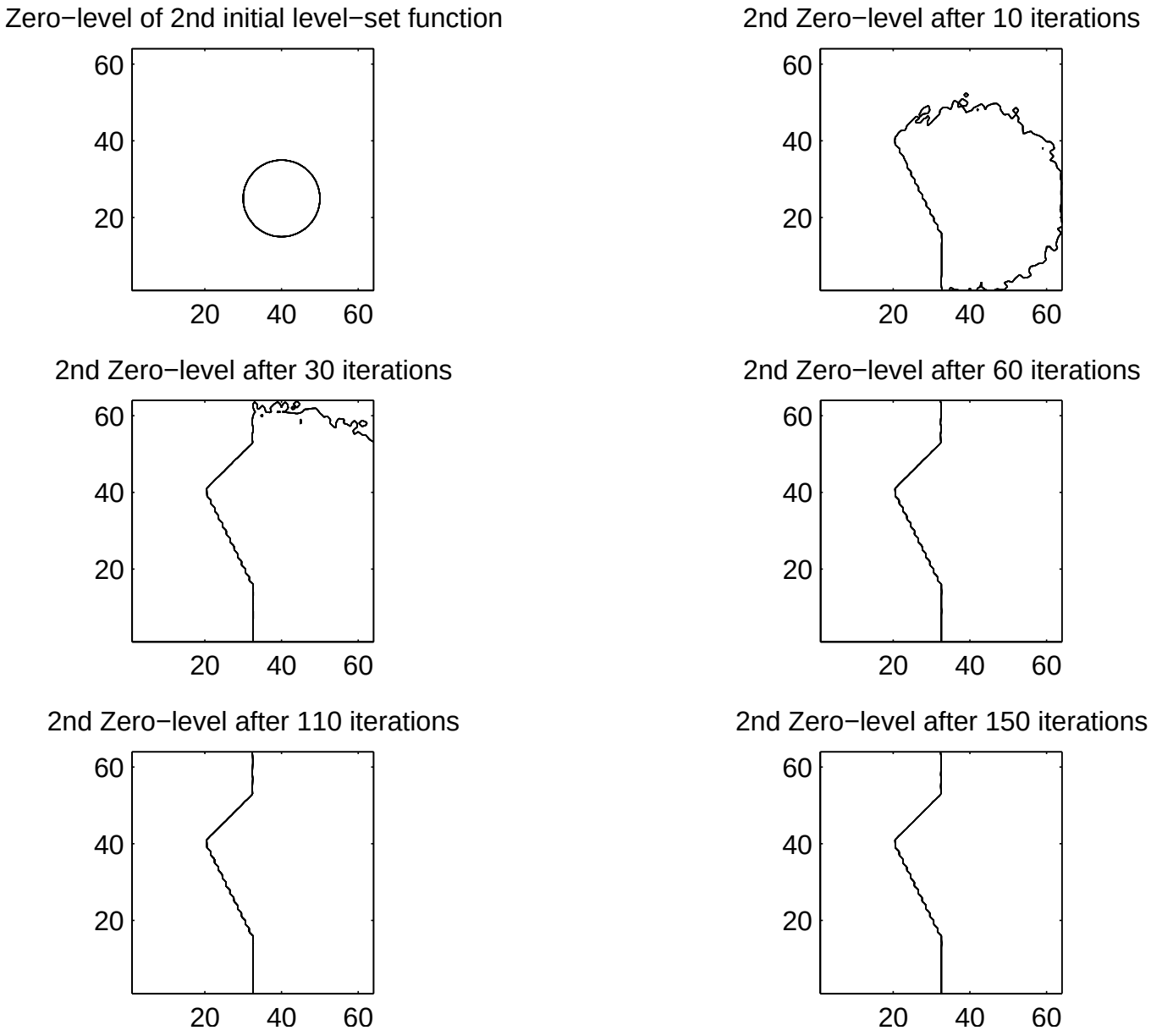


Figure 4.34 For Example 4.13.

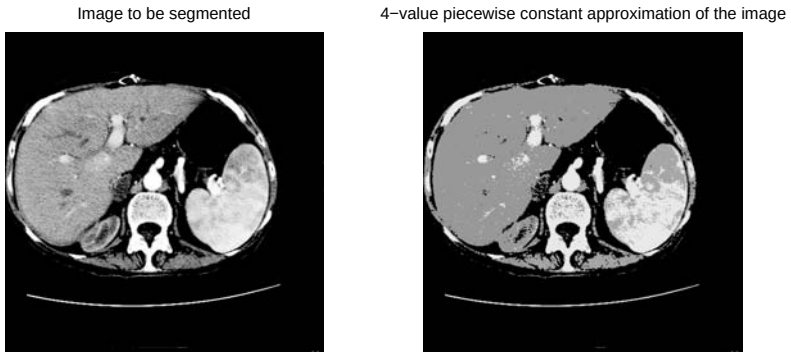


Figure 4.35 For Example 4.14.

Problems

4.1 Verify that the functional J_{CKS} in (4.8) is intrinsic, namely a different parameterization of the curve does not change the value of the functional (energy).

4.2 Apply the Caselles-Kimmel-Sapiro method studied in Sec. 4.2 to segment image obj5.mat. The MATLAB files you might want to use is cks_ak.m, re_initial.m and ex_sym2.m. You have to however prepare an appropriate initial level-set function. Report the parameters used in the MATLAB function and show the segmentation results.

4.3 Derive the Euler-Lagrange equation in (4.53) for the Chan-Vese two-phase model.

4.4 Derive the Euler-Lagrange equations in (4.68c) for the Vese-Chan four-phase model.

4.5 (a) Prepare MATLAB code that implements the difference scheme in (4.69) and (4.70) for 4-phase Vese-Chan model for image segmentation..

(b) Use the code from part (a) to repeat Example 4.12.

(c) Use the code from part (a) to repeat Example 4.13.

(d) Use the code from part (a) to repeat Example 4.14.