Saarland University
Faculty of Natural Sciences and Technology I
Department of Computer Science

Master's Thesis

Higher Order Anisotropic Smoothing of Images

submitted by
Jean Marc Roth

on January 19, 2009

Supervisor
Prof. Dr. Joachim Weickert

Advisor
Dr. Stephan Didas

Reviewers
Prof. Dr. Joachim Weickert
Prof. Dr.-Ing. Philipp Slusallek

Statement under Oath

I confirm under oath that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis.

Declaration of Consent

I agree to make my thesis (with a passing grade) accessible to the public by having it added to the library of the Computer Science Department.

Saarbrücken, January 19, 2009

J. M. Roth

Abstract

Diffusion filters are a well-established method for image smoothing, restoration and enhancement. Applications include contrast enhancement, denoising and interpolation. Current filters mainly employ derivatives of order two, and exist in linear and nonlinear, isotropic and anisotropic variations. Higher order diffusion filtering in the nonlinear *anisotropic* setting has not yet been a topic of much investigation — this is a gap that we would like to close.

First, we give a brief introduction into diffusion processes. We show what classes of diffusion filters exist and how some properties can be expressed in theory. We then introduce discrete theory, including fully discrete schemes and appropriate higher order boundary conditions. We investigate structure detectors and propose a special encoding of directional plots for our purpose.

In a second part, we introduce the fourth order evolution equation proposed by Tumblin and Turk, which is mainly used for contrast reduction of high dynamic range imagery. We extend it to the anisotropic setting. Furthermore, we propose a novel fourth order model, the Frobenius model, which can be considered a higher order analogon to existing lower order methods. We show how existing fourth order isotropic methods can be recast using this model. Then, we extend it to the anisotropic case and derive a variational formulation.

In the last part of this thesis, we start by putting all these models side by side and show their strengths and weaknesses using prominent numerical examples. We finally choose image inpainting as a suitable field of application for our novel method and compare it to lower order interpolants.

Acknowledgments

Thanks for the interesting topic go out to Joachim Weickert, who it is a pleasure to work with. Philipp Slusallek (Computer Graphics Group, Saarland University) spontaneously accepted to be the second reviewer on this thesis, I am very grateful for that.

I would also like to thank my advisor Stephan Didas for his professional and patient support and his many helpful suggestions throughout my work on this thesis, even though he already left Saarland University before its completion and was therefore in theory no longer responsible. He also provided several of his existing library functions for the implementations.

Last but not least, I would like to express my heartfelt gratitude to the few real friends that I have got to know up to this point in my life and who have been of invaluable support to me — you know who you are.

"Rien ne se perd, rien ne se crée, tout se transforme."
— Anaxagore de Clazomènes (500-428 B.C.)

Contents

Chapter 1

Introduction

1.1 Motivation and Problem Formulation

In image processing, image smoothing creates a function that approximates a certain data set, possibly leaving out noise or other undesired phenomena, while retaining important patterns. A subset of image smoothers are diffusion filters. Applications include contrast enhancement, denoising and interpolation.

Such filters can be categorized into isotropic and anisotropic[1] variants. The special case of anisotropic filters that are based on higher order derivatives than the current filters have not yet been a topic of much investigation — we are now going to tackle this issue.

The goal of this thesis is to develop a model of a higher order anisotropic image smoother. We will inspire ourselves using low order diffusion processes and also higher order isotropic ones that are already in existence, while we try to generalize and extend those formulations.

1.2 Existing Methods

Diffusion filters in image processing are inspired from physics. Their description dates back as far as the 19th century [Fic55]. A good overview can be found in [Phi05]. In the end, diffusion results in gradually smoothed and simplified versions of a signal. Some equations can no longer be called diffusion, in that case we employ the term *evolution equation*. This concept of scale spaces actually comes from Japan where it was first introduced in the late 1950s [Iij59, Iij63]. Those facts remained unknown to the Western world where they were discovered "again" in the 1980s. Weickert et al. summarize those findings [Wit83, WII99].

Regularization itself made its first appearance in [Whi23], used in the well-known [Tik63], and was later also carried over to image processing, e.g. in [Sch94], which also includes nonlinear diffusion. Well known work using a nonlinear total

[1] *anisotropic*: directionally dependent

variation based method can be found in [ROF92]. Relations between regularization and diffusion have been investigated in [SW00].

Another lasting discovery is by Perona and Malik [PM90], who are well-known for their diffusivity function allowing variable edge enhancement and smoothing. Additionally, it allows for some nice theoretical results, at least when it is regularized (preprocessed by Gaussian convolution, i.e. the signal is blurred) [CLMC92]. While we use their names mostly to denote the diffusivity function, sometimes the whole process of nonlinear isotropic diffusion is called the "Perona/Malik equation".

In [AGLM93], the authors give an axiomatic approach to linear and nonlinear scale spaces in image processing. [Eva98] is a whole book solely about modern theory of linear and nonlinear PDEs, while for computer scientists, [Far93] might be a nicer introduction into that subject. Also important for computer scientists are numerical issues that are treated e.g. in [MM95] (for mostly linear PDEs). For the sake of completeness we also refer to a book about variational calculus [GF00]. By the way, many things have been carried over to the matrix-valued case, e.g. in [FWBW06].

[Wei98] gives foundations for lower order anisotropic diffusion. In that context, structure detectors (extracting direction information from smoothed versions of a signal) were introduced in [FG87] and further investigated e.g. in [RS91]. [CZ98] explores structure detectors using higher order features and also space-frequency analysis.

As far as higher orders are concerned, [YK00] contains experiments on noise removal using higher order isotropic processes. The same topic is addressed by [LLT03]. What stands out is that they have used special (and very effective) discretizations for mixed derivatives. Linear energy functionals of arbitrary order, including solutions in the Fourier domain, have been investigated in [NFD97]. Total variation and higher order is treated in [CMM00], and the combination of second and fourth order terms in [LT06]. Higher order nonlinear diffusion filtering has also been investigated in-depth in [Did04, DWB05]. [Did08] contains many generalizations about higher order isotropic methods and also stability results.

[TT99] uses a non-symmetric divergence expression (different order of inner and outer derivatives) as a simple attempt to penalize everything down to linear gradients in their LCIS hierarchy, which is used for contrast reduction of high dynamic range (HDR) imagery. This is an evolution equation but it is not diffusion, since it cannot be cast in a variational formulation. [Wei99] uses the same evolution equation but with a different argument to the diffusivity function.

A nice overview over interpolation methods can be found in [Mei02], which includes over 350 references that will probably fulfill most of the reader's desires to know more about any of those methods. PDE-based interpolation is a recent development [BSCB00], and applications can e.g. be image super resolution [CK07], or image compression using anisotropic diffusion [GWW$^+$05, GWW$^+$08], optionally interleaved with mean curvature motion (MCM) [Zim07].

1.3 Overview

Chapter 2 gives a brief introduction into diffusion processes. We give an overview of what classes of diffusion filters exist and how some properties can be expressed in theory, at least for isotropic methods. We then show how these filters work in a discrete setting, including stencil notation, fully discrete schemes and – very important – boundary conditions that can be of use in higher order settings. We investigate structure detectors, a crucial part of anisotropic methods, and propose a special encoding of directional plots for our purposes.

Chapter 3 first introduces the fourth order evolution equation proposed by Tumblin and Turk. We give some facts about its stability in the isotropic setting, then we extend it to the anisotropic setting. In the same chapter, we propose a fourth order model, the Frobenius model, which can be considered a higher order analogon to existing lower order methods. We show how existing fourth order (isotropic) methods can be recast using this model. Then, we nontrivially extend it to the anisotropic case. The final section of this chapter derives a (linear) energy functional from the Euler-Lagrange equation.

Chapter 4 first puts all these models side by side and shows their strengths and weaknesses using prominent examples.

In Chapter 5, we choose image inpainting as a suitable field of application for our novel method, and compare it to lower order interpolants in a quantitative and qualitative manner.

Finally, Chapter 6 summarizes our work and gives an outlook of possible related work in the future.

1.4 Notation

This section gives a short presentation of the mathematical syntax used in this work.

Continuous and Discrete Images

An image is represented as a function

$$u : \Omega \to R\,, \tag{1.1}$$

where $\Omega_n \in \mathbb{R}^n$ is the n-dimensional image domain (here, $n = 2$ as we work in image planes, usually $\Omega := \Omega_2$) and R denotes the co-domain (usually, $R = \mathbb{R}$ for grayscale and $R = \mathbb{R}^3$ for color (RGB) images). We will mostly work on grayscale images because of notational convenience and also because the channel coupling involved in processing color images is mostly carried out in a straightforward way if the channels cannot be treated independently [GKKJ92].

In the continuous setting we usually denote the image coordinates as $(x, y) \in \Omega$. As we put theory into practice, we consider discrete images, in which we employ

the tuple (i, j) to denote (integer) coordinates and R then also represents a discrete range of values, e.g. the set of integers $\{0, \ldots, 255\}$, which, for convenience, we will also denote as Ω when needed. Our computations are obviously carried out using floating point arithmetic for the pixel values, the input and resulting images are however stored in integer precision for display/print.

Image Derivatives

In PDEs we have to deal a lot with partial derivatives, in our case of two-dimensional grayscale images $u(x, y) : \Omega \to \mathbb{R}$:

$$u_x := \partial_x u = \frac{\partial}{\partial x} u \quad \text{and} \quad u_y := \partial_y u = \frac{\partial}{\partial y} u \,. \tag{1.2}$$

Higher order (possibly mixed) derivatives will simply involve multiple indices.

Note that when addressing a certain pixel value of a discrete image, we write $u_{i,j}$, this shall not be mistaken for a derivative operator. On rare occasions, there might be constants (different ones for x- and y-direction) that are also written as h_x or h_y respectively — it will be obvious from the context that there is no derivative in play.

The gradient of $u(x, y)$ and its magnitude (using the Euclidean norm) are denoted as

$$\nabla u := \begin{pmatrix} u_x \\ u_y \end{pmatrix} \quad \text{and} \quad |\nabla u| = \sqrt{u_x^2 + u_y^2} \,, \tag{1.3}$$

respectively. The Hessian of u and its norm (using an entry-wise matrix norm, the Frobenius norm) are

$$\mathcal{H}(u) := \begin{pmatrix} u_{xx} & u_{xy} \\ u_{yx} & u_{yy} \end{pmatrix} \quad \text{and} \quad \|\mathcal{H}(u)\|_{\mathrm{Fr}} = \sqrt{u_{xx}^2 + u_{xy}^2 + u_{yx}^2 + u_{yy}^2} \,, \tag{1.4}$$

respectively.

Vector and Matrix Operations

We define a transpose operator on vectors, i.e. for a column vector

$$\mathbf{c} = \begin{pmatrix} c_1 \\ \vdots \\ c_n \end{pmatrix} \,, \quad \mathbf{c}^\top := \begin{pmatrix} c_1 & \cdots & c_n \end{pmatrix} = \mathbf{r} \tag{1.5}$$

represents the corresponding row vector with the same values. Obviously, also $\mathbf{r}^\top = \mathbf{c}$. The transpose of a $m \times n$ matrix $\mathbf{A}$ is the $n \times m$ matrix $\mathbf{A}^\top_{ij} = \mathbf{A}_{ji}$ for $1 \le i \le n$, $1 \le j \le m$.

For a two-dimensional vector $\mathbf{v} = (v_1 \; v_2)^\top$ there exist the perpendicular vectors

$$\mathbf{v}^\perp := \begin{pmatrix} \pm v_2 \\ \mp v_1 \end{pmatrix} \,. \tag{1.6}$$

The choice of sign depends on the direction in which one wishes to rotate the original vector by 90 degrees: clockwise or counterclockwise.

Multiplying a row vector $\mathbf{r}$ with a column vector $\mathbf{c}$ implicitly involves the scalar (inner) product $\langle \mathbf{r}, \mathbf{c} \rangle = \mathbf{r} \cdot \mathbf{c}$

$$\begin{aligned} \mathbf{r} \cdot \mathbf{c} :\ & \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R} \\ & \mathbf{r} \cdot \mathbf{c} \mapsto \mathbf{r}^\top \mathbf{c} = \sum_{i=1}^{n} r_i c_i \, . \end{aligned} \tag{1.7}$$

Conversely, multiplying a column vector with a row vector implicitly involves the tensor (outer) product

$$\begin{aligned} \mathbf{c} \otimes \mathbf{r} :\ & \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^{n \times n} \\ & \mathbf{c} \otimes \mathbf{r} \mapsto \mathbf{c}\mathbf{r}^\top = \begin{pmatrix} c_1 r_1 & c_1 r_2 & \cdots & c_1 r_n \\ c_2 r_1 & c_2 r_2 & \cdots & c_2 r_n \\ \vdots & \vdots & \ddots & \vdots \\ c_n r_1 & c_n r_2 & \cdots & c_n r_n \end{pmatrix} . \end{aligned} \tag{1.8}$$

The trace of a (square) $N \times N$ matrix $\mathbf{S}$ is the sum of its diagonal elements:

$$\operatorname{tr} \begin{pmatrix} s_{11} & s_{12} & \cdots & s_{1N} \\ s_{21} & s_{22} & \cdots & s_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ s_{N1} & s_{N2} & \cdots & s_{NN} \end{pmatrix} := \sum_{i=1}^{N} s_{ii} = s_{11} + s_{22} + \ldots + s_{NN} \, . \tag{1.9}$$

The trace of the Hessian is called Laplacian and written as

$$\Delta u := \operatorname{tr} \begin{pmatrix} u_{xx} & u_{xy} \\ u_{yx} & u_{yy} \end{pmatrix} = u_{xx} + u_{yy} \, . \tag{1.10}$$

In order to formulate a diffusion equation, we need the divergence of an n-dimensional vector $\mathbf{v} = (v_1 \; v_2 \; \cdots \; v_n)^\top$, which is defined as

$$\operatorname{div}(\mathbf{v}) := \sum_{i=1}^{n} \partial_i v_i = \nabla \cdot \mathbf{v} \, . \tag{1.11}$$

Convolution and Averaging

Averaging is often performed using Gaussian convolution. Convolution of two functions is defined as the integral transform

$$(f \star g)(\mathbf{x}) := \int_{-\infty}^{\infty} f(\chi) \cdot g(\mathbf{x} - \chi) \, d\chi \, , \tag{1.12}$$

whereas in Gaussian convolution one of those functions is the Gaussian function

$$G_{\mu,\sigma}(\mathbf{x}) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(\mathbf{x} - \mu)^2}{2\sigma^2} \right) \, , \tag{1.13}$$

where μ denotes the expected value, and $\sigma > 0$ the standard deviation. We usually have $\mu = 0$. In the sense of convolution, this is called a Gaussian convolution kernel. With regard to noise, this function is often used to model the behavior of noise, which is then called Gaussian noise.

Further details will be introduced as they become necessary.

Chapter 2

Diffusion Methods

This chapter starts with the origins and building blocks of the different classes of diffusion filters. Then, we will explore diffusivities (needed for nonlinear evolutions) and their corresponding flux functions. Finally we will treat discrete aspects, numerics and structure detection.

2.1 Diffusion Basics

Diffusion is part of transport phenomena. It is used to describe heat transfer and is modeled by the diffusion equation (also known as Fick's second law [Fic55]) which is a partial differential equation (PDE) that is derived from the following two laws:

Fick's first law describes the equilibration of concentration differences (like the diffusion of heat in the air or of particles in a solution). It postulates that the diffusive flux (flow of mass by diffusion) $\mathbf{j}$ goes from regions of high concentration (large u) to regions of low concentration with a magnitude D (the diffusion coefficient) that is proportional to the concentration gradient. This has been derived empirically and for now we write in 1-D:

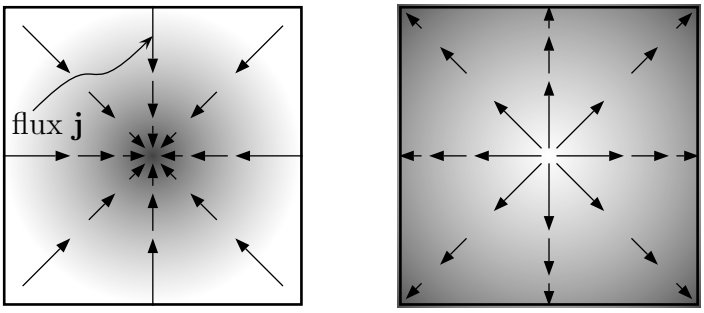
$$j = -D\frac{\partial u}{\partial x}\,. \tag{2.1a}$$

Note the minus sign making the tangent vector point outward (downhill).

Conservation of matter (continuity equation) describes that the change in concentration over time (t) is equal to the change in diffusive flux:

$$\partial_t u = -\frac{\partial j}{\partial x}\,, \tag{2.1b}$$

i.e. if there is convergent flow at a point ($\mathbf{j}_x < 0$, e.g. imagine water forced toward a single point via a "V"-shaped sink) the concentration at that point rises with time; if there is divergent flow ($\mathbf{j}_x > 0$), the concentration falls.



(a) Convergent flow (b) Divergent flow

FIGURE 2.1: Illustration of convergent and divergent flow.

See also Figure 2.1 for an illustration in 2-D, in which bright represents high concentration and dark represents low concentration, just like the gray values of images in our experiments later.

In multiple dimensions, equations (2.1a) and (2.1b) become

$$\mathbf{j} = -D \cdot \nabla u \,, \tag{2.2a}$$

and

$$\partial_t u = -\operatorname{div} \mathbf{j} \,, \tag{2.2b}$$

respectively.

Plugging (2.2a) into (2.2b) we obtain the general model for diffusion filters:

$$\boxed{\partial_t u = \operatorname{div} (D \cdot \nabla u)} \,. \tag{2.3}$$

This is also called the heat equation. Note that D remains inside the differential operator as it is a variable quantity (in this case, varies with location). We will see in the following what D, which we will later call a diffusion tensor, exactly does. Also note that in our notation, the flux (the expression inside the divergence) has opposite sign to the flux used in physics (see (2.2a)).

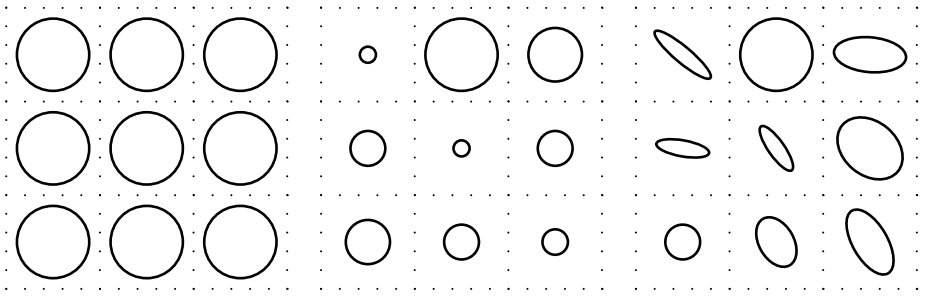
In the sense of scale spaces, one can write

$$u(x,t) = \begin{cases} f(x), & (t = 0) \,, \\ (\operatorname{op}(f))(x), & (t > 0) \,, \end{cases} \tag{2.4}$$

where op designates some operator, and t the stopping time. Diffusion in a computational sense is an iterative process, i.e. we start at time step $t = 0$ and evolve from there:

$$u(x, t+\tau) = \operatorname{op}((f|u)(x,t)) \;,\; \tau > 0 \,. \tag{2.5}$$

After a certain number of iterations n, at the stopping time $n \cdot \tau$, we have obtained the resulting image that has been created by repeatedly applying op() to f or u. Details on that difference and also on the iterative process in the discrete case can be found in the following sections.



(a) Homogeneous diffusion (b) Isotropic diffusion (c) Anisotropic diffusion

FIGURE 2.2: Illustration of convolution kernels corresponding to diffusion methods.

2.2 Classes of Diffusion Methods

In this section, we summarize the different classes of diffusion methods. Generally, to construct a diffusion filter, one has to choose amongst the following elements:

- ▷ linear / nonlinear diffusion,
- ▷ isotropic / anisotropic diffusion,
- ▷ lower order / higher order diffusion.

Let us start with the example of homogeneous diffusion (the simplest case of linear isotropic diffusion), which is equivalent to applying Gaussian convolution to the image. The circles as depicted in Figure 2.2 represent the view from above, of the standard deviation (i.e. points of inflection) of the 2-D Gaussians used to perform the convolution.

Nonlinear methods allow the size and shape of these convolution kernels to change over time, whereas for linear methods they always remain the same. Figuratively speaking, a linear filter only uses information of the initial image for feature detection and a nonlinear filter uses the evolving image for feature detection. This explains the choice of applying an operator to either f or u in (2.5) and can be closely related to image-driven and flow-driven optic flow methods [Bru06].

Truly isotropic diffusion can use kernels of different sizes depending on the location in the image (Figure 2.2(b)). Anisotropic diffusion additionally allows to treat directions independently, thus figuratively using ellipses instead of circles (Figure 2.2(c)).

Higher order methods cannot be illustrated that easily anymore, as circle/ellipse radii figuratively may become negative. This is one of the reasons that higher order methods sometimes produce oscillatory artifacts and do not adhere to simple forms of stability like maximum/minimum stability.

For now, let us stick to the lower order case and write down the previously mentioned methods formally.

(a) Popular noisy test image (b) Nonlinear isotropic diffusion (c) Nonlinear anisotropic diffusion

FIGURE 2.3: Lower order diffusion methods side by side.

2.2.1 Isotropic Diffusion

The "simplest" of the diffusion methods, linear isotropic diffusion, is written

$$\partial_t u = \operatorname{div}\left(g(|\nabla f|^2)\nabla u\right) \ . \tag{2.6}$$

$g(|\nabla f|^2)$ does not change during the evolution, since f does not change. A special case of linear isotropic diffusion is homogeneous diffusion ($g = 1$):

$$\partial_t u = \operatorname{div}\left(\nabla u\right) = \Delta u \ . \tag{2.7}$$

It is well-known that this is equal to convolution with a Gaussian kernel (1.13) of standard deviation $\sqrt{2t}$, where t denotes the stopping time of the evolution (see (2.4)).

If equation (2.6) is made nonlinear, we obtain the classic Perona/Malik equation [PM90]

$$\partial_t u = \operatorname{div}\left(g(|\nabla u|^2)\nabla u\right) \ . \tag{2.8}$$

In Figure 2.3(b), we see the result of applying this filter to the test image from Figure 2.3(a). Parameters have been optimized. One can see that this is a "lazy" filter: if it does not know what to do it prefers not to do anything (note the remaining noise at the discontinuities). We will explore the diffusivity function g in Section 2.3.

In order to be able to exploit properties like well-posedness, preservation of average gray value (conservation of mass) and maximum-minimum principle, furthermore making the process more robust under noise, one generally uses a blurred image to find the diffusivity at a certain point [CLMC92]. This is called regularization and written

$$\partial_t u = \operatorname{div}\left(g\left(|\nabla u_\sigma|^2\right)\nabla u\right) \ , \tag{2.9}$$

where $\nabla u_\sigma := \nabla\left(G_\sigma \star u\right)$ with σ being the noise scale, G_σ a Gaussian kernel of standard deviation σ, and $\star$ the convolution operator introduced in (1.12).

2.2.2 Anisotropic Diffusion

Anisotropic diffusion, i.e. diffusion with the property of being directionally dependent, involves the usage of a diffusion tensor that is not built using the identity matrix[1]. Extending the notation used in (2.9), we can write

$$\partial_t u = \operatorname{div}\left(g^*(J)\,\nabla u\right) \ . \tag{2.10}$$

g^* is derived from the original diffusivity function g, and now becomes a special function taking a matrix (the structure tensor J, depending on the blurred image u_σ, see below) as input and also becomes matrix-valued, the result of which we will call D (the diffusion tensor).

D is constructed as follows:

1. compute the structure tensor $J(u_\sigma) = \nabla u_\sigma \nabla u_\sigma^\top$,
2. perform principal axis transformation (PAT) of the structure tensor,
3. replace the dominant eigenvalue by diffusivity (the other one by 1), and
4. generate the diffusion tensor D.

We now discuss this step by step. First, we compute the traditional structure tensor (TST) [FG87]. It is used to overcome the cancellation problem of adjacent gradients having the same direction but opposite orientation, since $\nabla u_\sigma \nabla u_\sigma^\top = (-\nabla u_\sigma)(-\nabla u_\sigma^\top)$. It is typically averaged over a neighborhood in order to obtain the estimated orientation of the local structure; more on that in Section 2.2.3.

Then, we perform PAT (principal axis transformation), also called spectral decomposition, on the structure tensor:

$$J(u_\sigma) = V\,\mathring{\Lambda}\,V^\top = \begin{pmatrix} v_{1x} & v_{2x} \\ v_{1y} & v_{2y} \end{pmatrix} \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} \begin{pmatrix} v_{1x} & v_{1y} \\ v_{2x} & v_{2y} \end{pmatrix} \ , \tag{2.11}$$

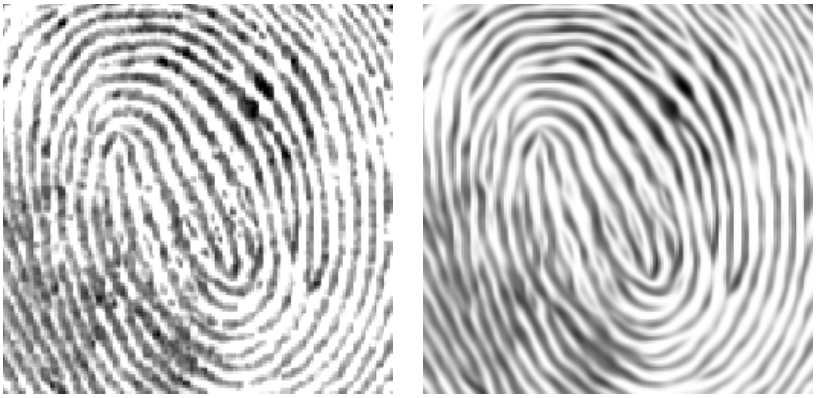
yielding its eigenvectors (principal directions) $(v_{1x}\ v_{1y})^\top = \mathbf{v_1} \parallel \nabla u_\sigma$ (across the feature) and $(v_{2x}\ v_{2y})^\top = \mathbf{v_2} \perp \nabla u_\sigma$ (along the feature) and the corresponding eigenvalues λ_1 and λ_2.

After that, the diffusivity is applied to the dominant eigenvector λ_1, yielding the diffusion tensor

$$D = g^*(J(u_\sigma)) = \begin{pmatrix} v_{1x} & v_{2x} \\ v_{1y} & v_{2y} \end{pmatrix} \begin{pmatrix} g(|\nabla u_\sigma|^2) & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} v_{1x} & v_{1y} \\ v_{2x} & v_{2y} \end{pmatrix} \ . \tag{2.12}$$

This is done in order to attenuate diffusion across structures and to allow diffusion along structures. Like that, important structures (regions and their borders) do not get destroyed or delocalized too much. In our case, $\lambda_2 = 1$, meaning that there can always be full flow along the structure.

[1] In (2.9), one could write gI instead of g alone.



(a) Original fingerprint image (b) Coherence enhancing diffusion

FIGURE 2.4: Result of lower order coherence enhancing diffusion filtering.

The whole procedure is called edge enhancing diffusion (EED) and is written

$$\partial_t u = \operatorname{div}\left(g^*(J(u_\sigma))\,\nabla u\right) \ . \tag{2.13}$$

The resulting image with optimized parameters ($\lambda = 3.5, \sigma = 3, t = 80, \tau = 0.2$) for the image from Figure 2.3(a) can be seen in Figure 2.3(c).

What we have seen in this section is in fact nonlinear anisotropic diffusion, a *linear* anisotropic filter can of course be created in complete analogy to what is explained at the beginning of this section.

2.2.3 Coherence Enhancing Diffusion

In the previous subsection, we have seen the direction of a possible structure only being inferred from the blurred image at the current pixel (inner averaging). One can extend this method to find the average direction within a certain *neighborhood* (outer averaging).

In the previous formulation of EED, this simply means to average the structure tensor J. This is called the integration scale ρ, and EED then becomes

$$\partial_t u = \operatorname{div}\left(g^*(J_\rho(u_\sigma))\,\nabla u\right) \ , \tag{2.14}$$

where $J_\rho(u_\sigma) = G_\rho \star \left(\nabla u_\sigma \nabla u_\sigma^\top\right)$ (Gaussian convolution). This is called coherence enhancing (anisotropic) diffusion (CED) and an example is shown in Figure 2.4.

We now see that EED can be written as $\partial_t u = \operatorname{div}(g^*(J_0(u_\sigma))\,\nabla u)$ by merely setting $\rho = 0$.

2.3 Diffusivity and Flux

The diffusivity function g penalizes smoothing of the image when a feature is present, as one would like to reduce the diffusion in areas of interest, i.e. containing important details which should not be lost.

A well-known diffusivity function is the one by Perona and Malik [PM90]:

$$g\left(s^2\right) = \frac{1}{1 + \frac{s^2}{\lambda^2}}\,, \tag{2.15}$$

a nonlinear function of the values of s. Note that sometimes this scalar-valued diffusivity varying by location is denoted as "anisotropic" in literature — what *we* mean with anisotropic however is a *tensor*-valued diffusivity, i.e. *directionally* dependent, which we will explore later. In this work, a diffusion equation using Perona/Malik diffusivity is simply a linear (non-homogeneous) isotropic filter.

A diffusion process can be characterized via its diffusivity function to perform forward diffusion (smoothing), backward diffusion (feature enhancement), or possibly both, depending on the location in the image, the nature of the diffusivity function and the magnitude of its argument. To analyze its behavior, one employs the flux function

$$\Phi(s) := g(s^2) \cdot s\,. \tag{2.16}$$

The factor λ (where present^2) is called contrast parameter (physicists would call it conductance threshold) and indicates the border between forward diffusion and backward diffusion, i.e. if the argument to diffusivity (e.g. the gradient) is lower than λ we are in presence of backward diffusion, otherwise of forward diffusion. The range of a diffusivity function's values commonly lies in the interval $[0; 1]$.

Let us now write down some well-known diffusivities. We include the flux functions alongside the diffusivity functions in Figure 2.5. We only show the part of the plot for a positive argument since the function is usually given as argument a squared value like the squared gradient norm $|\nabla u|^2 > 0$. Some common diffusivities are

Perona/Malik

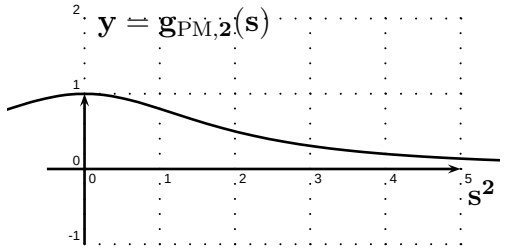
$$g_{\mathrm{PM},\lambda}(s^2) = \frac{1}{1 + \frac{s^2}{\lambda^2}}\,, \tag{2.17a}$$

Charbonnier

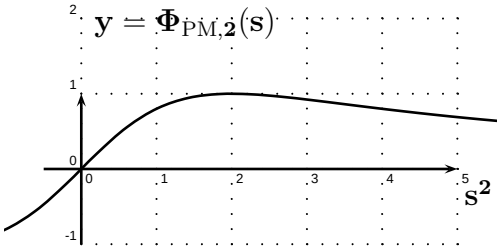
$$g_{\mathrm{CH},\lambda}(s^2) = \frac{1}{\sqrt{1 + \frac{s^2}{\lambda^2}}}\,, \quad \text{and} \tag{2.17b}$$

Weickert

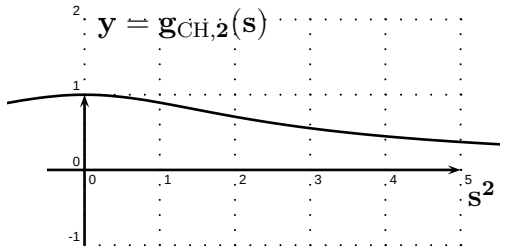
$$g_{\mathrm{WK},\lambda}(s^2) = \begin{cases} 1 & (s^2 = 0)\,, \\ 1 - \exp\left(\frac{-3.31488}{(\frac{s}{\lambda})^8}\right) & (s^2 > 0)\,. \end{cases} \tag{2.17c}$$



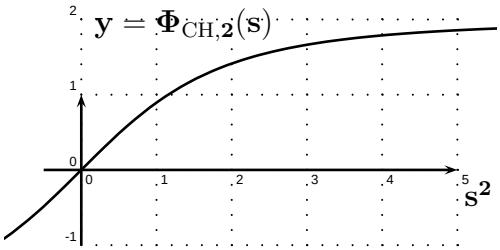
(a) Perona/Malik diffusivity



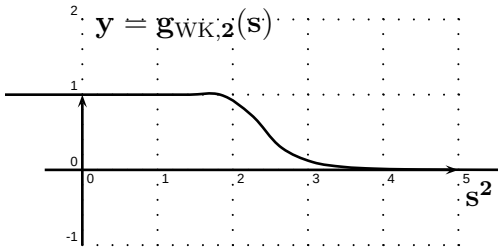
(b) Perona/Malik flux



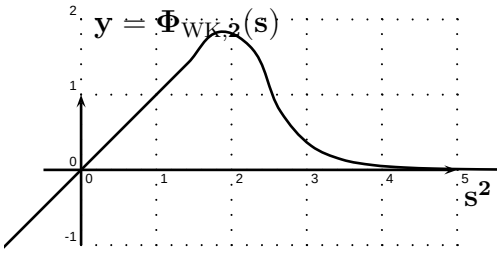
(c) Charbonnier diffusivity



(d) Charbonnier flux



(e) Weickert diffusivity



(f) Weickert flux

FIGURE 2.5: Function plots ($\lambda = 2$). ▶ **left**: diffusivity ▶ **right**: flux

As one can easily see, Charbonnier diffusivity [CBFAB97] provides no backward diffusion at all ($\Phi'_{\mathrm{CH}}(s) > 0$, for all s), Perona/Malik diffusivity [PM90] provides backward/forward diffusion (steered by the contrast parameter λ), and Weickert diffusivity [Wei98] takes it to the limits (very drastic falloff after the backward/-forward diffusion border). Therefore, in practice, Weickert diffusion allows to keep edges of discontinuities very sharp, and usually, for anything to happen, processes using this diffusivity might need to run for a long time.

[2]There exist diffusivities without such a parameter, such as total variation diffusivity $\frac{1}{|s|}$.

2.3.1 Flux Analysis — Isotropic Case

To analyze whether local feature enhancement (backward diffusion) is possible at all, independent of the diffusivity function in use[3], one tries to write the diffusion equation such that on the right hand side one obtains terms that are dependent on the diffusivity or flux function, or its derivatives.

Taking the example of the nonlinear isotropic filter from (2.8), one can rewrite it in several ways, and we show that one can be more meaningful than the other, although both evidently end up with the same result. Note that we perform these calculations in 1-D for simplicity's sake:

$$\begin{aligned}
\partial_t u &= \partial_x \left(g(u_x^2) u_x \right) \\
&= \frac{\mathrm{d}}{\mathrm{d}x} \left(\Phi(u_x(x)) \right) \\
&= \frac{\mathrm{d}}{\mathrm{d}u_x} \Phi(u_x) \frac{\mathrm{d}}{\mathrm{d}x} u_x(x) \\
&= \frac{\mathrm{d}}{\mathrm{d}u_x} \Phi(u_x) u_{xx}(x) \\
&= \Phi'(u_x) u_{xx}(x)
\end{aligned} \quad , \qquad
\begin{aligned}
\partial_t u &= \partial_x \left(g(u_x^2) u_x \right) \\
&= \frac{\mathrm{d}}{\mathrm{d}x} g(u_x^2) \cdot u_x(x) + g(u_x^2) \frac{\mathrm{d}}{\mathrm{d}x} u_x \\
&= \frac{\mathrm{d}}{\mathrm{d}u_x^2} g(u_x^2) \frac{\mathrm{d}}{\mathrm{d}x} u_x^2 \cdot u_x + g(u_x^2) u_{xx} \\
&= g'(u_x^2) \cdot 2 u_x u_{xx} \cdot u_x + g(u_x^2) u_{xx} \\
&= \underbrace{\left(2 g'(u_x^2) u_x^2 + g(u_x^2) \right)}_{\Phi'(u_x)} u_{xx}(x) \, .
\end{aligned} \tag{2.18}$$

It is important to always remember that almost everything is a function here, including the derivatives, which we sometimes write in lazy notation u_x which means $u_x(x)$. We have used the Leibniz notation for the derivatives as otherwise it would not always be perfectly clear (or visible) what variable we are differentiating against, especially during chain rule application in combination with the aforementioned lazy notation.

(2.18) shows that for $\Phi'(u_x) > 0$ we have (second order) forward diffusion, and for $\Phi'(u_x) < 0$ we have backward diffusion (again: only if the diffusivity function in use permits it at all).

Still in the isotropic case, one can write in 2-D [AGLM93]

$$\begin{aligned}
\partial_t u &= 2g' \cdot (u_x u_x u_{xx} + 2 u_x u_y u_{xy} + u_y u_y u_{yy}) + g \cdot (u_{xx} + u_{yy}) \\
&= 2g' \nabla u^\top \mathcal{H}(u) \nabla u + g \Delta u \\
&= 2g' |\nabla u|^2 \partial_{\eta\eta} u + g \partial_{\xi\xi} u + g \partial_{\eta\eta} u \\
&= \underbrace{\left(2g' |\nabla u|^2 + g \right)}_{\Phi'(\nabla u)} \partial_{\eta\eta} u + g \partial_{\xi\xi} u \, ,
\end{aligned} \tag{2.19}$$

where $\eta \parallel \nabla u$ and $\xi \perp \nabla u$, i.e. there is forward/backward diffusion along the flowlines (lines of maximal gray value variation) and forward diffusion along the isolines (contour line, line along constant gray value).

We see that this is analogous to the 1-D case, where the only exisiting direction was denoted as x-direction; here, this more generally is η-direction.

[3] Of course, if the diffusivity function does not permit it, there cannot be backward diffusion.

2.3.2 Flux Analysis — Anisotropic Case

Let us now investigate edge enhancement (or not) in the anisotropic case. Obviously, here we have to do these calculations in 2-D, as anisotropy in 1-D does not make much sense. Additionally, the diffusion tensor contains the convolution $u_\sigma = (G_\sigma \star u)$, which we however continue to use in this abstract notation.

We had previously introduced EED in (2.13). We rewrite it as

$$\partial_t u = \operatorname{div}(D\,\nabla u) \quad \text{with} \quad D = g^*\left(J(u_\sigma)\right) = \begin{pmatrix} a & b \\ b & c \end{pmatrix} . \tag{2.20}$$

From PAT (2.11), we know that

$$D = \begin{pmatrix} a & b \\ b & c \end{pmatrix} = \begin{pmatrix} \lambda_1 v_{1x}^2 + \lambda_2 v_{2x}^2 & \lambda_1 v_{1x} v_{1y} + \lambda_2 v_{2x} v_{2y} \\ \lambda_1 v_{1x} v_{1y} + \lambda_2 v_{2x} v_{2y} & \lambda_1 v_{1y}^2 + \lambda_2 v_{2y}^2 \end{pmatrix}, \tag{2.21}$$

where $v_1 = (v_{1x}, v_{1y})^\top$ and $v_2 = (v_{2x}, v_{2y})^\top$ are unit vectors across and along the feature, respectively.

After replacing λ_1 by diffusivity $g(|\nabla u_\sigma|^2)$ and λ_2 by 1, and with the reasoning that $v_1 \perp v_2$, that is $v_2 = (-v_{1y}, v_{1x})^\top$, we can rewrite (2.20) as

$$\begin{aligned} \partial_t u = &\frac{\partial}{\partial x}\left[(g v_1^2 + v_2^2) u_x\right] + \frac{\partial}{\partial x}\left[(g v_1 v_2 - v_1 v_2) u_y\right] \\ &+ \frac{\partial}{\partial y}\left[(g v_1 v_2 - v_1 v_2) u_x\right] + \frac{\partial}{\partial y}\left[(g v_2^2 + v_1^2) u_y\right] , \end{aligned} \tag{2.22}$$

where we simply wrote v_1 for the first component of the dominant eigenvector (previously v_{1x}) and v_2 for the second component (previously v_{1y}).

We are in the need of partial derivatives of the diffusivity function $g(|\nabla u_\sigma|^2)$. Let $h(\nabla u_\sigma) = h((u_\sigma)_x, (u_\sigma)_y) = |\nabla u_\sigma|^2 = (u_\sigma)_x^2(x,y) + (u_\sigma)_y^2(x,y)$, then by the chain rule we very formally obtain

$$\begin{aligned} &\frac{\partial}{\partial x} g(h((u_\sigma)_x(x,y), (u_\sigma)_y(x,y))) \\ &= \frac{\partial}{\partial h} g(h) \cdot \frac{\partial}{\partial x} h((u_\sigma)_x(x,y), (u_\sigma)_y(x,y)) \\ &= g'(|\nabla u_\sigma|^2)\Big(\frac{\partial}{\partial (u_\sigma)_x} \underbrace{h((u_\sigma)_x, (u_\sigma)_y)}_{(u_\sigma)_x^2 + (u_\sigma)_y^2} \frac{\partial}{\partial x}((u_\sigma)_x(x,y)) \\ &\qquad\qquad + \frac{\partial}{\partial (u_\sigma)_y} h((u_\sigma)_x, (u_\sigma)_y) \frac{\partial}{\partial x}((u_\sigma)_y(x,y))\Big) \\ &= g'(|\nabla u_\sigma|^2)\left(2(u_\sigma)_x \cdot (u_\sigma)_{xx} + 2(u_\sigma)_y \cdot (u_\sigma)_{yx}\right) . \end{aligned} \tag{2.23}$$

Therefore we have

$$g_x(|\nabla u_\sigma|^2) = 2g'(|\nabla u_\sigma|^2)\left((u_\sigma)_x (u_\sigma)_{xx} + (u_\sigma)_y (u_\sigma)_{yx}\right) , \tag{2.24a}$$

and obtain analogously

$$g_y(|\nabla u_\sigma|^2) = 2g'(|\nabla u_\sigma|^2)\left((u_\sigma)_x(u_\sigma)_{xy} + (u_\sigma)_y(u_\sigma)_{yy}\right) \ . \tag{2.24b}$$

Thus, (2.22) becomes

$$\begin{aligned}\partial_t u = &\left(g_x v_1^2 + g(v_1^2)_x + (v_2^2)_x\right) u_x + \left(g v_1^2 + v_2^2\right) u_{xx} \\ &+ \left(g_x v_1 v_2 + (g-1)(v_1 v_2)_x\right) u_y + \left((g-1)v_1 v_2\right) u_{yx} \\ &+ \left(g_y v_1 v_2 + (g-1)(v_1 v_2)_y\right) u_x + \left((g-1)v_1 v_2\right) u_{xy} \\ &+ \left(g_y v_2^2 + g(v_2^2)_y + (v_1^2)_y\right) u_y + \left(g v_2^2 + v_1^2\right) u_{yy} \ ,\end{aligned} \tag{2.25a}$$

which, after inserting the full expressions for g_x and g_y and then rearranging, gives

$$\begin{aligned}\partial_t u = &\left(2g'(u_\sigma)_x(u_\sigma)_{xx} + 2g'(u_\sigma)_y(u_\sigma)_{xy}\right)\left(v_1^2 u_x + v_1 v_2 u_y\right) \\ &+ \left(2g'(u_\sigma)_x(u_\sigma)_{xy} + 2g'(u_\sigma)_y(u_\sigma)_{yy}\right)\left(v_1 v_2 u_x + v_2^2 u_y\right) \\ &+ \left((g(v_1^2)_x + (v_2^2)_x) + ((g-1)(v_1 v_2)_y)\right) u_x \\ &+ \left(((g-1)(v_1 v_2)_x) + (g(v_2^2)_y + (v_1^2)_y)\right) u_y \\ &+ \left(g v_1^2 + v_2^2\right) u_{xx} + 2\left((g-1)v_1 v_2\right) u_{xy} + \left(g v_2^2 + v_1^2\right) u_{yy} \ .\end{aligned} \tag{2.25b}$$

More compactly, this could be written as

$$\begin{aligned}\partial_t u = &\ 2g'\left(\left(\mathcal{H}(u_\sigma)\nabla u_\sigma\right)^\top \cdot \left((vv^\top)\nabla u\right)\right) \\ &+ g \cdot \left((vv^\top)\nabla\right)\nabla u + g \cdot (\underbrace{v^\top \mathcal{H}(u) v}_{u_{vv}}) \\ &+ 1 \cdot \left((v^\perp v^{\perp\top})\nabla\right)\nabla u + 1 \cdot (\underbrace{v^{\perp\top}\mathcal{H}(u)v^\perp}_{u_{v^\perp v^\perp}}) \ .\end{aligned} \tag{2.26}$$

It is reassuring to see the terms with underbraces, which show diffusivity-regulated flow *across* the detected structure (v) and full flow *along* the detected structure ($v^\perp$), as it should be for an anisotropic process. We do not yet fully grasp the complete underlying meaning of this expression, and therefore cannot finally say if it now allows backward diffusion or not, although it seems so: v (and its derivatives) are affected by g and g', while $v^\perp$ is affected by the constant 1 alone.

2.4 Discrete Aspects

This section first introduces the basics of computing approximations of functions using its derivatives, which is necessary to understand in order to produce correct discretizations of those derivatives. As an example we compute a discretized version of the Perona/Malik equation. We show that all discrete diffusion equations can be rewritten in matrix notation, which is used to analyze certain properties and therefore stability issues. Stencil notation (or element-wise notation) and its relation to matrix notation is also introduced. We then proceed to fully discrete schemes and finally take a peek at boundary conditions.

2.4.1 Taylor's Theorem and Finite Differences

Taylor's theorem gives a sequence of approximations of a differentiable function around a given point by polynomials (the Taylor polynomials of that function) whose coefficients depend only on the derivatives of the function at that point [SB02].

For a sufficiently differentiable function f, an approximation of f at the point x near a is given by

$$f(x) \approx f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots + \frac{f^{(n)}(a)}{n!}(x-a)^n \,, \tag{2.27a}$$

or more compactly

$$\sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n \,, \tag{2.27b}$$

where $n!$ is the factorial of n, $f^{(0)} = f$, $(x-a)^0 = 1$ and $0! = 1$.

In full analogy, the multivariate version is

$$f(\mathbf{x}) = f(\mathbf{a}) + \mathrm{D}f(\mathbf{a})^T(\mathbf{x}-\mathbf{a}) + \frac{1}{2!}(\mathbf{x}-\mathbf{a})^T\mathrm{D}^2 f(\mathbf{a})(\mathbf{x}-\mathbf{a}) + \cdots \,, \tag{2.27c}$$

where $D^i f$ denotes the i-th order derivative of f (Euler's notation).

For a function that depends on two variables, x and y, the Taylor series to third order about the point (a, b) e.g. is

$$\begin{aligned} f(x,y) \approx\ & f(a,b) + f_x(a,b)\cdot(x-a) + f_y(a,b)\cdot(y-b) \\ & + \frac{1}{2!}\left(f_{xx}(a,b)\cdot(x-a)^2 + 2f_{xy}(a,b)\cdot(x-a)(y-b) + f_{yy}(a,b)\cdot(y-b)^2\right) \\ & + \frac{1}{3!}\big(f_{xxx}(a,b)\cdot(x-a)^3 + 3f_{xxy}(a,b)\cdot(x-a)^2(y-b) \\ & \quad + 3f_{xyy}(a,b)\cdot(x-a)(y-b)^2 + f_{yyy}(a,b)\cdot(y-b)^3\big) \,. \end{aligned} \tag{2.28}$$

The expression can be written including a final remainder term, which, when the series does indeed converge toward zero as n approaches infinity, helps show

that the function is equal to its Taylor series. It is then an analytic function and can be validly expressed as a Taylor series in a neighborhood of some point a.

Our goal is to exploit Taylor's theorem in order to obtain numerical approximations of derivatives, also called finite difference approximations.

As a very simple example, let us approximate the first order derivative $u_x(x)$ of $u(x)$ in pixel i using the two pixels i, and $i+1$. We expand those two points using Taylor's theorem and get

$$\begin{aligned} u_i &= u_i \text{ , as well as} \\ u_{i+1} &= u_i + hu_i' + \frac{h^2}{2}u_i'' + \frac{h^3}{6}u_i''' + O(h^4) \text{ ,} \end{aligned} \tag{2.29a}$$

where h is the pixel grid size. Comparing the coefficients yields

$$\begin{aligned} 0 \cdot u_i + 1 \cdot u_i' &\stackrel{!}{=} \alpha_0 u_i + \alpha_1 u_{i+1} \\ &= \alpha_0 u_i + \alpha_1 \left(u_i + hu_i' + \frac{h^2}{2}u_i'' + \cdots \right) \\ &= (\alpha_0 + \alpha_1) \cdot u_i + h\alpha_1 u_i' + O(h^2) \text{ .} \end{aligned} \tag{2.29b}$$

On the left hand side we first indicate the derivative that we would like to obtain the approximation for (by setting all other coefficients zero) and then note that we want this to be equal to some approximation using the two desired pixels, which we then insert from the list of initial Taylor approximations above, and then finally regroup the terms.

This example leads to the trivial linear system of equations

$$\begin{aligned} \alpha_0 + \alpha_1 &= 0 \cdot u_i \\ h\alpha_1 &= 1 \cdot u_i' \text{ ,} \end{aligned} \tag{2.29c}$$

which is written in matrix-vector notation as

$$\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \alpha_1 \end{pmatrix} = \begin{pmatrix} 0 \\ \frac{1}{h} \end{pmatrix} \text{ .} \tag{2.29d}$$

The solution is straightforward: $\alpha_1 = \frac{1}{h}$ and therefore $\alpha_0 = -\alpha_1 = -\frac{1}{h}$. This example has shown the creation of the popular forward finite difference

$$u_i' \approx \frac{u_{i+1} - u_i}{h} \text{ .} \tag{2.29e}$$

The backward difference, in which the center term has a positive sign, can be explained the same way. The system of equations would be

$$\begin{pmatrix} 0 & -1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} \alpha_{-1} \\ \alpha_0 \end{pmatrix} = \begin{pmatrix} \frac{1}{h} \\ 0 \end{pmatrix} \text{ ,} \tag{2.30a}$$

and therefore yield

$$u_i' \approx \frac{u_i - u_{i-1}}{h} \text{ .} \tag{2.30b}$$

Both are valid approximations and they are not the only ones. This example already shows that discretizations can be a delicate subject. When higher order derivatives (especially mixed derivatives) come into play, one has to be very careful as it is not always clear which is the best approximation, i.e. also works well in practice.

It is clear that, the more pixels contribute to the approximation, the better it probably gets, this is known as order of consistency. Furthermore, one should be careful about approximations that only use every other pixel (or not include the center pixel); they may produce checkerboard artifacts, oscillations, etc. and also they might not be able to detect certain structures (e.g. the very checkerboard pattern).

2.4.2 Discrete Theory of Diffusion Filters

So far we have seen but modeling and (space- and time-)continuous theory. In order to utilize these methods on a computing device, where images are sampled on a pixel grid, one obviously has to turn to the discrete setting for implementation and also to analyze its pitfalls using a suitable notation: matrix notation.

Let us start with the example of a nonlinear isotropic process of lower order, which we repeat from (2.9):

$$\begin{aligned} \partial_t u &= \operatorname{div}\left(g(|\nabla u_\sigma|^2)\nabla u\right) \\ &= \partial_x\left(g(|\nabla u_\sigma|^2)\partial_x u\right) + \partial_y\left(g(|\nabla u_\sigma|^2)\partial_y u\right) \ . \end{aligned} \tag{2.31}$$

In the semi-discrete setting, space is discretized, thus in pixel (i,j) we have

$$\begin{aligned} \frac{\mathrm{d}u_{i,j}}{\mathrm{d}t} &\approx \frac{\left(g(|\nabla u_\sigma|^2)\partial_x u\right)_{i+\frac{1}{2},j} - \left(g(|\nabla u_\sigma|^2)\partial_x u\right)_{i-\frac{1}{2},j}}{h_x} \\ &\quad + \frac{\left(g(|\nabla u_\sigma|^2)\partial_y u\right)_{i,j+\frac{1}{2}} - \left(g(|\nabla u_\sigma|^2)\partial_y u\right)_{i,j-\frac{1}{2}}}{h_y} \\ &= \frac{\left(g(|\nabla u_\sigma|^2)\right)_{i+\frac{1}{2},j}\left(\partial_x u\right)_{i+\frac{1}{2},j} - \left(g(|\nabla u_\sigma|^2)\right)_{i+\frac{1}{2},j}\left(\partial_x u\right)_{i-\frac{1}{2},j}}{h_x} \\ &\quad + \frac{\left(g(|\nabla u_\sigma|^2)\right)_{i,j+\frac{1}{2}}\left(\partial_y u\right)_{i,j+\frac{1}{2}} - \left(g(|\nabla u_\sigma|^2)\right)_{i,j-\frac{1}{2}}\left(\partial_y u\right)_{i,j-\frac{1}{2}}}{h_y} \\ &\approx \frac{\left(\frac{g_{i+1,j}+g_{i,j}}{2}\right)\left(\frac{u_{i+1,j}-u_{i,j}}{h_x}\right) - \left(\frac{g_{i,j}+g_{i-1,j}}{2}\right)\left(\frac{u_{i,j}-u_{i-1,j}}{h_x}\right)}{h_x} \\ &\quad + \frac{\left(\frac{g_{i,j+1}+g_{i,j}}{2}\right)\left(\frac{u_{i,j+1}-u_{i,j}}{h_y}\right) - \left(\frac{g_{i,j}+g_{i,j-1}}{2}\right)\left(\frac{u_{i,j}-u_{i,j-1}}{h_y}\right)}{h_y} \ , \end{aligned} \tag{2.32}$$

where h_x and h_y denote the grid sizes in x- and y-directions respectively, and have nothing to do with a derivative. The first approximation is produced by central derivatives; the second equality splits diffusivity (g) and image gradient (u_x, u_y)

into separate discretizations; finally, we obtain the diffusivities between the pixel grid by averaging the values on the grid points and the image data by using central derivatives again.

As we mentioned, this can be written in so-called matrix notation, which in fact is the notation of a system of ordinary differential equations using a matrix-vector product on the right hand side

$$\frac{\partial \mathbf{u}}{\partial t} = \mathbf{A}(\mathbf{u})\mathbf{u}\,, \tag{2.33}$$

where $\mathbf{A}$ is called the system matrix and contains the diffusivities, or more exactly: the map that tells us how much each pixel contributes to each other pixel. One simply has to rearrange the result of (2.32).

Note that the 2-D image $\mathbf{u}$ is encoded as a 1-D vector, its elements are the pixels of the image from left to right and from top to bottom. This means they are now ordered in a linear fashion, i.e. no longer indexed two-dimensionally with $u_{i,j}$ but as $u_{o(i,j)}$ via a function $o(i,j) = j \cdot N + i$, where j denotes the row and i the column of the respective element (zero-based).

The system matrix $\mathbf{A}$ is not necessarily symmetric; a symmetric matrix means that pixel i contributes to pixel j as much as pixel j contributes to pixel i. This does not need to be the case, but can be used to prove certain properties of the diffusion process, if it applies. It is a sparse matrix with as many diagonals as there are neighboring pixels to be considered to compute the new value at the current pixel. The number of intermediate diagonals containing only zeroes is therefore related to the image size and thus the "distance" to the next dimension.

The diagonals that are far away from the main diagonal in the system matrix contain the neighboring pixels from another row (but the same column) of the signal, while the diagonals that are adjacent to the center diagonal represent the neighbors from the current row (but the other columns). Clearly showing all interactions for a 4×2 image (4 columns, 2 rows, 8 pixels), this looks like

$$\underbrace{\begin{pmatrix} a_{0\to0} & a_{1\to0} & 0 & 0 & a_{4\to0} & 0 & 0 & 0 \\ a_{0\to1} & a_{1\to1} & a_{2\to1} & 0 & 0 & a_{5\to1} & 0 & 0 \\ 0 & a_{1\to2} & a_{2\to2} & a_{3\to2} & 0 & 0 & a_{6\to2} & 0 \\ 0 & 0 & a_{2\to3} & a_{3\to3} & 0 & 0 & 0 & a_{7\to3} \\ a_{0\to4} & 0 & 0 & 0 & a_{4\to4} & a_{5\to4} & 0 & 0 \\ 0 & a_{1\to5} & 0 & 0 & a_{4\to5} & a_{5\to5} & a_{6\to5} & 0 \\ 0 & 0 & a_{2\to6} & 0 & 0 & a_{5\to6} & a_{6\to6} & a_{7\to6} \\ 0 & 0 & 0 & a_{3\to7} & 0 & 0 & a_{6\to7} & a_{7\to7} \end{pmatrix}}_{\mathbf{A}} \begin{pmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \end{pmatrix}, \tag{2.34}$$

i.e. inside the system matrix we denote the contribution of pixel u_p^k (previous iteration's result) to pixel u_q^{k+1} by $a_{p\to q}$, which equals a_{qp} in traditional matrix element notation (note the switched indices). Coordinates (i,j) of p could be recovered by computing

$$i = p \mod N \qquad \text{and} \qquad j = \lfloor \frac{p}{N} \rfloor\,,$$

for a previously "linearized" pixel index $p = o(i, j)$, where $\lfloor \cdot \rfloor$ denotes rounding to the lower integer value[4]. Section 2.4.4 will take a look at the same issue from another perspective, making this issue more intuitive, especially the question why not all diagonals of matrix (2.34) are complete.

For now, we however consequently proceed from semi-discrete to fully discrete theory, in which time is discretized too:

$$\frac{\mathbf{u}^{k+1} - \mathbf{u}^k}{\tau} = \mathbf{A}(\mathbf{u})\mathbf{u}\,, \tag{2.35}$$

where k denotes the time step and τ the time step size. We currently do not have a time step on the right-hand side; this will be explored in the next section.

2.4.3 Fully Discrete Schemes

In fully discrete theory (remember (2.35)), there exist several schemes one can employ to solve for $\mathbf{u}^{k+1}$. Note that $\mathbf{u}^0 = \mathbf{f}$, where $\mathbf{f}$ is the original image.

Explicit Scheme (Forward Euler Method)

The explicit scheme is derived from (2.35) by setting time on the right-hand side to the previous time step and then rearranging terms:

$$\begin{aligned} &\frac{\mathbf{u}^{k+1} - \mathbf{u}^k}{\tau} = \mathbf{A}(\mathbf{u}^k)\mathbf{u}^k \\ \iff\ &\mathbf{u}^{k+1} = \mathbf{u}^k + \tau\mathbf{A}(\mathbf{u}^k)\mathbf{u}^k \\ \iff\ &\mathbf{u}^{k+1} = \underbrace{\big(\mathbf{I} + \tau\mathbf{A}(\mathbf{u}^k)\big)}_{\mathbf{Q}(\mathbf{u}^k)}\mathbf{u}^k\,, \end{aligned} \tag{2.36a}$$

where $\mathbf{I} = \mathrm{diag}(1)$ is the identity matrix of appropriate size.

This is the most trivial scheme, as it only involves matrix-vector multiplication to solve for the next iterand. It is restricted by the time step size τ which must lie below a certain maximum, above which the scheme destabilizes, which happens when certain entries of the matrix $\mathbf{Q}$ change sign, i.e. become negative (see the design criteria for diffusion filters, especially (D4) and (D5), by Weickert [Wei98]). Therefore it is the slowest of all schemes; nevertheless, it is used in our experiments because it is the numerically most exact scheme and easy to implement.

[4]In practice, this can be done in a computationally less intensive way using addition and subtraction of certain known array offsets. This is however only important when using a more advanced solver like SOR.

Semi-Implicit Scheme (Backward Euler Method)

The semi-implicit scheme replaces the current with the future time step on the right-hand side:

$$\begin{aligned} \frac{\mathbf{u}^{k+1}-\mathbf{u}^k}{\tau} &= \mathbf{A}(\mathbf{u}^k)\mathbf{u}^{k+1} \\ \iff \mathbf{u}^{k+1}-\tau\mathbf{A}(\mathbf{u}^k)\mathbf{u}^{k+1} &= \mathbf{u}^k \\ \iff \left(\mathbf{I}-\tau\mathbf{A}(\mathbf{u}^k)\right)\mathbf{u}^{k+1} &= \mathbf{u}^k \\ \iff \mathbf{u}^{k+1} &= \underbrace{\left(\mathbf{I}-\tau\mathbf{A}(\mathbf{u}^k)\right)^{-1}}_{\mathbf{Q}}\mathbf{u}^k\,, \end{aligned} \tag{2.36b}$$

and thus requires to solve a linear system of equations in each iteration (it is no great effort to obtain $\mathbf{A}(\mathbf{u}^k)$ from the known $\mathbf{u}^k$). This can be done efficiently using e.g. Jacobi, Gauss-Seidel or successive over-relaxation (SOR) solvers, which can however be less precise than the explicit method. SOR requires a symmetric and positive definite matrix $\mathbf{Q}$ to guarantee convergence; for Gauss-Seidel, a diagonally dominant matrix also works. The time step size can be chosen arbitrarily large, without having to worry about stability, although precision will suffer.

Fully Implicit Scheme

For the sake of completeness, we also mention the fully implicit scheme in which implicitness is introduced into the diffusivity term too, and is written

$$\begin{aligned} \frac{\mathbf{u}^{k+1}-\mathbf{u}^k}{\tau} &= \mathbf{A}(\mathbf{u}^{k+1})\mathbf{u}^{k+1} \\ \iff \left(\mathbf{I}-\tau\mathbf{A}(\mathbf{u}^{k+1})\right)\mathbf{u}^{k+1} &= \mathbf{u}^k \\ \iff \mathbf{u}^{k+1} &= \left(\mathbf{I}-\tau\mathbf{A}(\mathbf{u}^{k+1})\right)^{-1}\mathbf{u}^k\,. \end{aligned} \tag{2.36c}$$

It thus requires to solve a nonlinear system of equations in each iteration, as the matrix $\mathbf{A}$ depends *nonlinearly* on the *unknown* $\mathbf{u}^{k+1}$ because of the nonlinear diffusivity function g therein. $\mathbf{u}^{k+1}$ needs to be determined such that the resulting matrix $\mathbf{A}(\mathbf{u}^{k+1})$ is a solution of the linear system of equations. This scheme is computationally most expensive, as it needs to be processed e.g. by solving a series of linear systems of equations.

2.4.4 Stencil Notation

Diffusion in the discrete setting in the end is nothing more than consecutively applying a weighted sum of the pixels in a local neighborhood, optionally also including the center pixel, to the center pixel, for all pixels. Stencil notation clearly shows this for one single pixel and can therefore be considered as element-based notation. It is produced by sorting the terms from the discretized divergence expression by indices $(u_{i-1,j}, u_{i,j}, \ldots)$, extracting their coefficients (diffusivity). It

thus represents one row of the system matrix, therefore demystifying it a little bit; more on that correspondence below. Since all the rows of the system matrix are the same, except for an index shift, sometimes by simply looking at the sign pattern of the stencil, one can already make assumptions about the stability of a certain process.

A stencil indicates the new value of the center pixel (shown in gray) with regard to the sum of the values of the neighboring pixels[5]:

$(i-1, j-1)$	$(i, j-1)$	$(i+1, j-1)$
$(i-1, j)$	(i, j)	$(i+1, j)$
$(i-1, j+1)$	$(i, j+1)$	$(i+1, j+1)$

Applying a stencil to an image, means positioning its center (consecutively) onto each pixel of the image and performing the above-mentioned operation. We denote with "$\cdot$" the application of a stencil to some other grid of elements (e.g. other stencil, image, ...).

Formally, one can write for a stencil S and an image pixel $u_{i,j}$ the stencil application $u_{i,j}^{k+1} = S \cdot u_{i,j}^{k}$ as

$$u_{i,j}^{k+1} = \sum_{\substack{p \in \{0,\dots,s_x-1\} \\ q \in \{0,\dots,s_y-1\}}} S_{p,q} u_{i+p-2c_x, j+q-2c_y}^{k} \,, \tag{2.37}$$

where s_x and s_y denote the size of the stencil S in x- and y-direction, respectively, and c_x and c_y denote the center of the stencil, i.e. for a $s_x \times s_y$ stencil (usually, $s_x = s_y$) the center is at $(c_x, c_y) = (\lfloor \frac{s_x}{2} \rfloor, \lfloor \frac{s_y}{2} \rfloor)$, as the sizes must forcibly be odd values, otherwise the center was not clearly defined. Furthermore, if $i + p \notin \{0, \dots, M-1\}$ and $j + q \notin \{0, \dots, N-1\}$, the summands are supposed to adhere to some predefined boundary condition (e.g. simply be zero). For pseudocode, we refer the reader to Appendix A.3.

Now one can write the system matrix (2.34) as

$$\mathbf{A} = \left(\begin{array}{cccccccc} c_0 & x_0^+ & 0 & 0 & y_0^+ & 0 & 0 & 0 \\ x_1^- & c_1 & x_1^+ & 0 & 0 & y_1^+ & 0 & 0 \\ 0 & x_2^- & c_2 & x_2^+ & 0 & 0 & y_2^+ & 0 \\ \hline 0 & 0 & x_3^- & c_3 & 0 & 0 & 0 & y_3^+ \\ \hline y_4^- & 0 & 0 & 0 & c_4 & x_4^+ & 0 & 0 \\ 0 & y_5^- & 0 & 0 & x_5^- & c_5 & x_5^+ & 0 \\ 0 & 0 & y_6^- & 0 & 0 & x_6^- & c_6 & x_6^+ \\ 0 & 0 & 0 & y_7^- & 0 & 0 & x_7^- & c_7 \end{array}\right) . \tag{2.38}$$

We denote the center weights for the corresponding stencil application (centered at u_i) by c_i and their left, right, lower and upper neighbor weights by x_i^-, x_i^+,

[5] Again, note that this ordering is different from the organization of a matrix where the first index indicates the row and therefore goes downward, and the second index the column (rightward).

y_i^- and y_i^+, respectively. Here we can see, as previously mentioned, how the near diagonals contain the interactions with the elements in the same row (denoted as x^+ and x^-) and the far diagonals contain elements from other rows (denoted as y^+ and y^-). With the previously mentioned linear ordering, the image would be encoded in a structure like

$u_{o(0,0)} = u_0$	$u_{o(1,0)} = u_1$	$u_{o(2,0)} = u_2$	$u_{o(3,0)} = u_3$
$u_{o(0,1)} = u_4$	$u_{o(1,1)} = u_5$	$u_{o(2,1)} = u_6$	$u_{o(3,1)} = u_7$

.

Therefore, in *each row* of the system matrix one can see the stencil applied. The stencil with its corresponding weights at element u_3 (upper right corner of image) would thus look like

0	0	0
x_3^-	c_3	0
0	y_3^+	0

$\cdot\, u_3$.

The stencil application to pixel u_i can be seen in row i of the system matrix (zero-based). We have marked the row corresponding to the previous stencil in matrix (2.38). Note that since we use a very simple example (otherwise the system matrix would become too large to illustrate), we are always at a boundary of the image. These so-called boundary conditions can pose problems (e.g. note the upper row and rightmost column of the preceding stencil equal zero), and will therefore be treated next.

2.4.5 Boundary Conditions

Boundary treatment is important in the discrete handling of evolution equations, especially their derivatives. Imagine a 1-D signal u with domain $\{1, \ldots, N\}$ of which we try to find e.g. the first order derivatives. One of the simplest ways to do this would be to apply a forward finite difference scheme $u' \approx \frac{u_{i+1}-u_i}{h}$ (h being the pixel size) to each value in order to obtain the derivative at that point. Computing a value using this scheme at e.g. the border point N is not possible this way, since it would need the value at $N+1$ at which the function u is not defined. Using backward differences would generate the same problem at the lower end 1. Using central differences would create the problem at both ends of the domain.

To overcome this deficiency one usually defines certain boundary conditions which extend the domain of a function at both ends. The most popular methods are (still in 1-D for simplicity's sake):

periodic boundary conditions, which repeat the signal over and over again

$$u_{i+nN} = u_i \quad \text{for} \quad n \in \mathbb{Z} \text{ and } i \in \Omega\,, \tag{2.39a}$$

Dirichlet boundary conditions, which merely define everything outside of the domain to be zero (zeroth order boundary)

$$u_i = 0 \quad \text{for} \quad i \notin \Omega\,, \text{and} \tag{2.39b}$$

Neumann boundary conditions which make the first order derivative zero across boundaries, which happens when mirroring the signal at the boundaries, which is why these boundary conditions are also called reflecting or mirroring boundary conditions (first order boundary)

$$\frac{\partial}{\partial n} u_i = 0 \quad \text{for} \quad i \in \Gamma \,, \tag{2.39c}$$

where n is the (outer) normal vector on Γ, by which we informally denote the border region, and therefore $\frac{\partial u}{\partial n} := n^\top \nabla u$ the directional derivative of u in the direction of n. Typically, this normal vector is parallel to a coordinate axis.

When both Dirichlet and Neumann boundary conditions are imposed, we talk about Cauchy boundary conditions.

Generally, natural boundary conditions that arise out of a variational problem (see next section) could be any of those methods or other, more complex ones. For higher order methods one usually also needs more than a border size of one dummy pixel: the worst approximation of any derivative of order n using the finite difference method comprises $n+1$ function values, i.e. for a fourth order equation one needs two boundary pixels if the finite difference scheme that is employed is central (best case).

2.5 Variational Formulation and Numerics

It is important to know the two main strategies used to find a numerical solution (using a finite precision computing device) to the problems at hand. To that end we first have to introduce an energy functional [6], which is an integral expression of the form

$$E(u) = \int_\Omega D(u(x,y)) + \sum_i \alpha_i S_i(u(x,y))\, dx\, dy \,, \tag{2.40}$$

where D denotes a data term (or similarity term) like the squared difference $(u-f)^2$, with u being the evolved and f the original image, and S designates a smoothness assumption, of which there can be many although it is nice to have a well-working model with as few smoothness assumptions as possible. α_i is the regularization parameter with regard to the smoothness term S_i; in the case of multiple smoothness terms, it allows to assign a certain weight/importance to each one. This energy functional is also called variational problem or formulation.

As mentioned before, the two numerical solution strategies to find a minimizer are the

Elliptic method, using which one obtains the divergence expression by computing the Euler-Lagrange equation(s) from the functional, which constitute a

[6] *functional*: a function taking function(s) in its argument

necessary condition for a minimizer [GF00]. The general Euler-Lagrange equation up to order M can be written

$$0 \stackrel{!}{=} \sum_{k=0}^{M} \left[(-1)^k \cdot \sum_{i=1}^{2^k} D_i^k E_{D_i^k u} \right] , \tag{2.41}$$

where $D_i^k u$ here denotes, similar to Euler's notation, the i-th k-th order derivative (also mixed) of u [Har06].

The Euler-Lagrange equation for a 2-D signal up to an order of two thus is

$$0 \stackrel{!}{=} E_u - \frac{\partial}{\partial x} E_{u_x} - \frac{\partial}{\partial y} E_{u_y} + \frac{\partial}{\partial xx} E_{u_{xx}} + \frac{\partial}{\partial xy} E_{u_{xy}} + \frac{\partial}{\partial yx} E_{u_{yx}} + \frac{\partial}{\partial yy} E_{u_{yy}} . \tag{2.42}$$

These finally lead to large linear or nonlinear systems of equations, that need to be solved.

Parabolic method, in which one applies gradient descent to the energy functional, resulting in a diffusion equation, or in a diffusion-reaction system (if a data term was specified). One then has to compute the steady state of this evolution, i.e. the solution for $t \to \infty$.

A minimizer of a (discrete) energy functional necessarily satisfies $\nabla_{\mathbf{u}} E = 0$, with $\mathbf{u} \in \mathbb{R}^n$. Gradient descent then is

$$\partial_t \mathbf{u} = -\gamma \nabla_{\mathbf{u}} E , \tag{2.43}$$

with an arbitrary speed factor $\gamma > 0$.

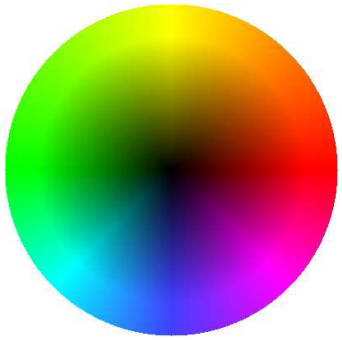
Obviously, this is in no contradiction to the Euler-Lagrange equations above, since for $t = \infty$ we have $\partial_t \mathbf{u} = 0$.

In the elliptic world, one talks about smoothness and data terms, and their counterparts in the parabolic world are diffusion and reaction terms, respectively. In this work, we employ the parabolic approach, i.e. we compute evolutions using methods that we describe in one of the upcoming sections. These evolutions create so-called scale spaces [RSW99, SW00].

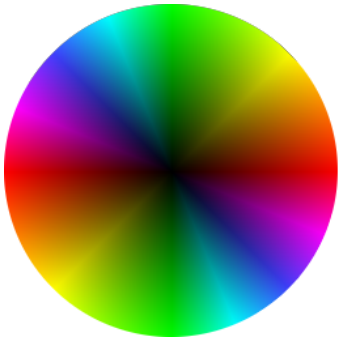
We do not base our novel methods on any energy functional – finding one is no obvious task – we are working on the Euler-Lagrange equations directly, so to speak. Afterward, we will see if there is a possibility to nicely recast this as energy functional.

Note the strange writing of the diffusivity functions with a squared argument on the left hand side (remember Section 2.3). This is justified by the fact that like this, we can interpret g directly as diffusivity. Let us use an energy functional with a nonlinear isotropic smoothness term (i.e. as the evolution evolves, the edges are allowed to "move")

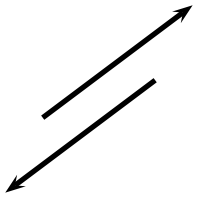
$$E(u(x,y)) = \int_\Omega D(u) + \alpha \Psi(|\nabla u|^2) \, dx \, dy . \tag{2.44a}$$



(a) Classic optic flow color coding scheme (includes orientation)



(b) Structure detection color coding scheme (no orientation)



(c) Vectors that have same direction but opposite orientation

FIGURE 2.6: Color coding schemes with and without orientation information.

Next, we do not use a squared argument in the smoothness term.

$$E(u(x,y)) = \int_\Omega D(u) + \alpha\Psi(|\nabla u|)\,dx\,dy\,. \tag{2.44b}$$

The corresponding Euler-Lagrange equations would be

$$0 = E_u - \alpha\,\mathrm{div}\left(\Psi'\left(|\nabla u|^2\right)\nabla u\right) \tag{2.45a}$$

and

$$0 = E_u - \alpha\,\mathrm{div}\left(\Psi'\left(|\nabla u|\right)\frac{\nabla u}{|\nabla u|}\right)\,, \tag{2.45b}$$

respectively. Note that only in the former case, we can correctly call Ψ' diffusivity, which we usually denote as g.

2.6 Structure Detection

In this section, we give an overview over some of the modeling choices when designing an image smoother. More specifically, we are interested in the anisotropic case, where direction, orientation, structure tensor and the argument to the diffusivity function come into play.

2.6.1 Direction and Orientation

A straight line only has a direction and contains no orientation information. A vector $\mathbf{v}$ that is parallel to that line can have two possible orientations. If we scale the vector $\mathbf{v}$ by a scalar λ, we have that $\lambda\mathbf{v}, \lambda > 0$ has the same orientation than $\mathbf{v}$ whereas $\lambda\mathbf{v}, \lambda < 0$ and $\mathbf{v}$ have opposite orientation, while all have the same direction. See also Figure 2.6(c).

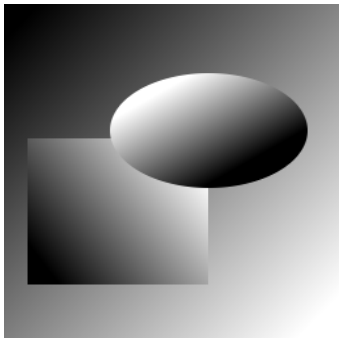


FIGURE 2.7: Test image featuring discontinuities and linear gradients (Source:[Did08]).

In contrast to optic flow problems, where it is necessary to know the actual direction *and* orientation of how things change, we need only consider the direction. Therefore we introduce a novel color coding scheme, which can be seen in Figure 2.6(b) alongside the classic coding (as used e.g. by [Bru06]) in Figure 2.6(a).

2.6.2 Structure Tensors

We have already seen the traditional structure tensor (TST) [FG87]:

$$G_\rho \star \left(\nabla u_\sigma \nabla u_\sigma^\top\right) = G_\rho \star \begin{pmatrix} (u_\sigma)_x (u_\sigma)_x & (u_\sigma)_x (u_\sigma)_y \\ (u_\sigma)_y (u_\sigma)_x & (u_\sigma)_y (u_\sigma)_y \end{pmatrix} . \tag{2.46}$$

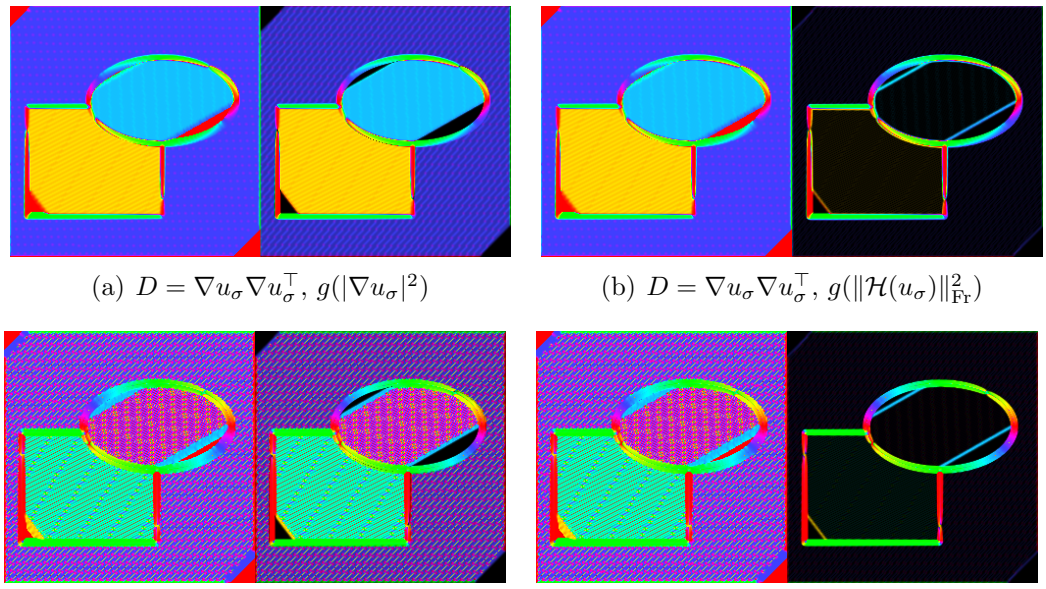
When we are interested in recognizing higher order features we might simply use the Hessian matrix of the regularized image

$$\mathcal{H}(u_\sigma) = \begin{pmatrix} (u_\sigma)_{xx} & (u_\sigma)_{xy} \\ (u_\sigma)_{yx} & (u_\sigma)_{yy} \end{pmatrix} \tag{2.47}$$

instead of the aforementioned structure tensor, optionally including the outer averaging by Gaussian convolution.

This cannot be called "higher order structure tensor" (as in [SWS09]) but rather is a TST featuring higher order components (TST^{HOC}) in the spirit of [CZ98]. Figure 2.8 illustrates the use of the TST and TST^{HOC} with different diffusivity arguments on the test image 2.7 featuring linear gradients. Each subfigure contains a plot of the dominant eigenvector of the corresponding structure tensor (left), where the recognized directions are colored according to the novel color coding from Figure 2.6(b). The right element of each subfigure contains the same representation, only this time color values are attenuated with regard to the argument to the diffusivity function g. Dark areas indicate regions of no features (high diffusivity) and the colored areas remain where diffusivity is low, i.e. a structure was detected, in order to see what direction was detected for that structure.

The red color in the directional plots can indicate a horizontal direction being detected, but it can also be a sign that no direction was detected, i.e. the default dominant eigenvector is $(1, 0)^\top$ (treated as such in our code), which is exactly red.



(a) $D = \nabla u_\sigma \nabla u_\sigma^\top$, $g(|\nabla u_\sigma|^2)$ (b) $D = \nabla u_\sigma \nabla u_\sigma^\top$, $g(\|\mathcal{H}(u_\sigma)\|_{\mathrm{Fr}}^2)$

(c) $D = \mathcal{H}(u_\sigma)$, $g(|\nabla u_\sigma|^2)$ (d) $D = \mathcal{H}(u_\sigma)$, $g(\|\mathcal{H}(u_\sigma)\|_{\mathrm{Fr}}^2)$

FIGURE 2.8: Principal directions and diffusivity of test image with linear gradients.

This is in fact the case here: after applying the diffusivity function we see that most of those regions turn dark.

In the upper row of Figure 2.8 we see the TST applied to the image with both first (left subfigure) and second order (right subfigure) arguments given to the diffusivity function, i.e. squared gradient magnitude and squared Frobenius norm of the Hessian, respectively. We can see that the directions of the linear gradients are detected very concisely using the TST. Since the Hessian norm of a linear change in values is zero, the upper right picture gives the desired effect after applying diffusivity: low diffusivity is attributed to edges and high diffusivity happens in the linear regions.

The lower row of Figure 2.8 contains the same experiment, only this time using $\mathrm{TST}^{\mathrm{HOC}}$. We can see that no directions are (correctly) detected. This is easy to explain, as a second order derivative is zero for constant parts of the image and also for linear gradients. Also note that the second order structure detector does *not* jump at the edge itself but only at its boundaries (as the second derivative only is high at high variations of the gradient, i.e. at high curvature).

Derivatives of odd order have an extremum at the edge, while derivatives of even order are only zero there, as Figure 2.9 illustrates for an edge-like signal and a signal with a peak. In a practical example, one can see this in Figure 2.10, showing the detected directions weighted by diffusivity after an evolution process applied to the image from Figure 2.3(a).

It follows in a straightforward manner that in the next step we have to use some test image with higher order features. To that end we use the two-dimensional sine signal from Figure 2.11. Note that the colors in Figure 2.11(b) have nothing to do

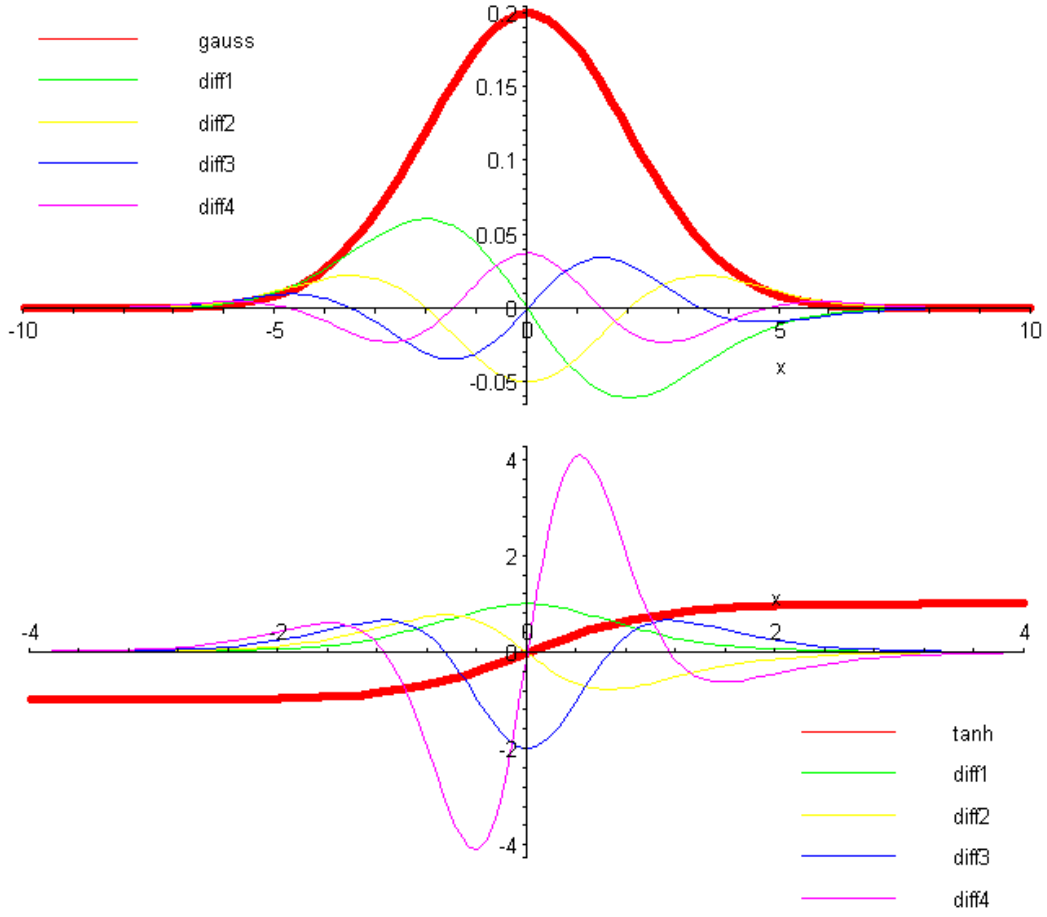
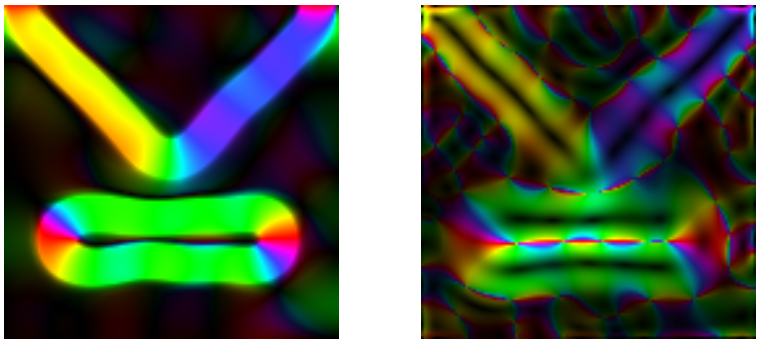


FIGURE 2.9: Derivatives of up to order four of a ▶ **top**: noise-like function (Gaussian) , and ▶ **bottom**: an edge-like function ($\tanh(x)$) .

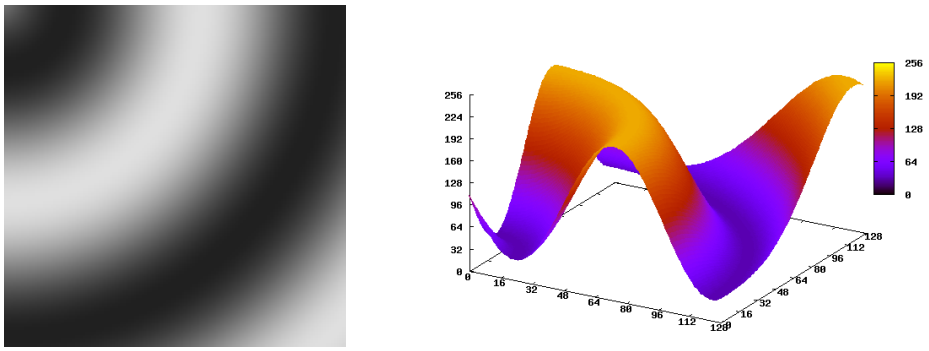
with directions, they merely indicate the magnitude of the signal (gray value from subfigure (a) treated as third dimension). The results can be inspected in Figure 2.12.

This time we see that both TST and TST^{HOC} correctly recognize the feature directions. The TST still seems to be the cleaner solution as the TST^{HOC} appears noisier. This could however be attributed to the discretizations of higher order derivatives, where especially the implementation of high-quality mixed derivatives can be tricky. Additionally, the TST^{HOC} (correctly) detects a structure at the (non-stationary) points of inflection ($u'' = 0, u' \neq 0$); it is not clear if we want this or not. In any case, this feature is of no particular interest and when the higher order diffusivity argument comes into play (rightmost items in Figure 2.12), those areas are attenuated anyway, so the results are more or less the same.



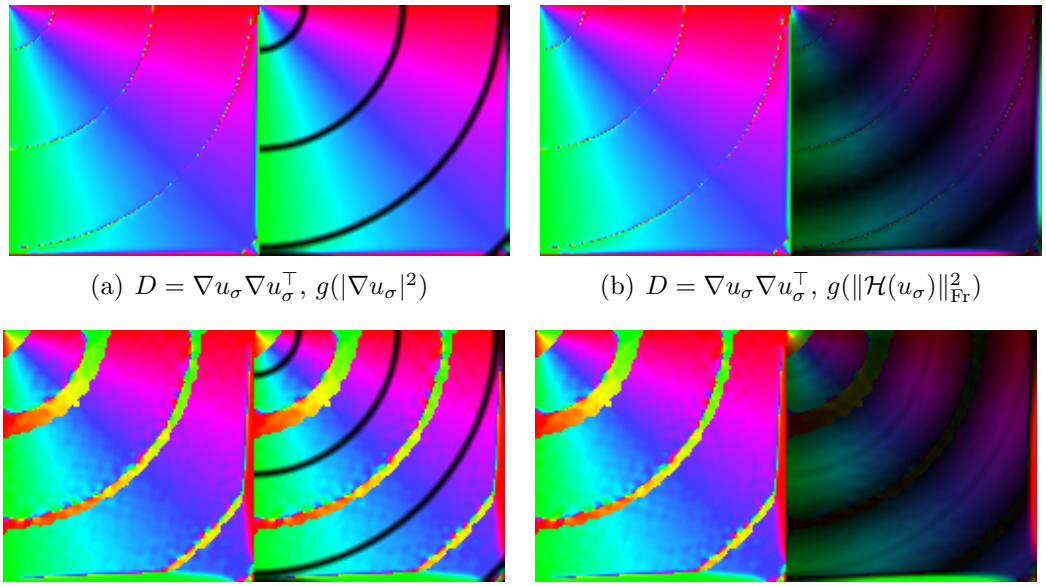
(a) $g(|\nabla u_\sigma|)$ (b) $g(\|\mathcal{H}(u)\|^2_{\mathrm{Fr}})$

FIGURE 2.10: Illustration of the impact of different arguments to the diffusivity function.



(a) 2-D image (b) 3-D plot of 2-D image

FIGURE 2.11: Sinusoidal test image (Source:[Did08]).



(a) $D = \nabla u_\sigma \nabla u_\sigma^\top$, $g(|\nabla u_\sigma|^2)$ (b) $D = \nabla u_\sigma \nabla u_\sigma^\top$, $g(\|\mathcal{H}(u_\sigma)\|^2_{\mathrm{Fr}})$

(c) $D = \mathcal{H}(u_\sigma)$, $g(|\nabla u_\sigma|^2)$ (d) $D = \mathcal{H}(u_\sigma)$, $g(\|\mathcal{H}(u_\sigma)\|^2_{\mathrm{Fr}})$

FIGURE 2.12: Principal directions and diffusivity for a sinusoidal signal.

Chapter 3

Higher Order Methods

In the previous chapter, we have revisited the common diffusion methods. Which one to choose for a specific task depends on experience and on the field of application — there is no general recipe. One could for example argue that a nonlinear (flow-driven) method prevails over a linear (image-driven) method to enhance a noisy image, as the noise in the initial image can be misinterpreted as features. On the other hand, one can argue for linear diffusion in the case where prominent structure is featured in the initial image and which probably needs to be kept. It would be nonsense to use a nonlinear process in this case, which, at some point, might blur the important structure away, or at least create blurred edges, which (at least visually) will not be very appealing.

What we left out in the previous chapter was the fundamental change of not only using derivatives of order two in the evolution equation, but derivatives of higher order (than order two). It is important to understand that when using derivatives of order four in the evolution equation, we penalize features of order two or more. If we wrote a fourth order evolution equation as an energy functional (when possible), we would encounter smoothness terms including derivatives of order two.

This means that structures are altered, such that their final order is less than order two, i.e. in the extreme case they become linear variations (order one). Features that are of less than order two are left unaltered because they are not recognized by the higher order terms (e.g. a second order derivative of a linear function is zero). In general, one can consider higher order methods to contain less staircasing and rather be curvature-driven.

As a sidenote: as far as e.g. linear isotropic diffusion is concerned, it can be generalized easily. Let $u(x, y)$ be a signal in two-dimensional Cartesian coordinates, then linear isotropic diffusion of order $2n$ can be written

$$\partial_t u = (-1)^{n+1} \Delta^n(u) \quad \text{with} \quad \Delta^n(u) := \sum_{i=0}^{n} \binom{n}{i} \frac{\partial^{2n} u}{\partial_x^{2n-2i} \partial_y^{2i}} \,. \tag{3.1}$$

where Δ denotes the Laplacian as introduced in (1.10).

3.1 Theoretical Aspects

It is important to know if a numerical scheme is stable, and under what conditions. Stability depends on many things, most notably the discretization and the fully discrete scheme used (see Section 2.4.3).

Many lower order methods adhere to a maximum-minimum principle

$$\min_j f_j \leq u_i(t) \leq \max_j f_j \quad \forall i, \forall t > 0\,, \tag{3.2}$$

i.e. the maximum and minimum value of the original image f will not be exceeded by the evolved image u during any time of the evolution. Other properties are e.g. convergence toward a constant steady state and preservation of average gray value [Wei98].

Using higher order methods however, a stability like this is more rarely encountered — over- and undershoots happen frequently. Experience shows that higher order methods can nevertheless adhere to weaker stability conditions like stability in the $\mathcal{L}_2$-norm

$$\|\mathbf{u}^{k+1}\|_2 \leq \|\mathbf{u}^k\|_2\,, \tag{3.3}$$

where $\|\mathbf{u}\|_2 := \sqrt{\sum_{i=1}^n |u_i|^2}$, which is synonymous with monotonically decreasing variance between the original and evolved image during the evolution, thus limiting the magnitude of over- and undershoots. Let $(\sigma^2(\mathbf{u}))^k$ be the variance of the image $\mathbf{u}$ at iteration k, then we have

$$\begin{aligned}
\|\mathbf{u}^{k+1}\|_2 &\leq \|\mathbf{u}^k\|_2 \\
\sqrt{\sum_{i=1}^{|\Omega|} |u_i^{k+1}|^2} &\leq \sqrt{\sum_{i=1}^{|\Omega|} |u_i^k|^2} \\
\sum_{i=1}^{|\Omega|} \left(u_i^{k+1}\right)^2 &\leq \sum_{i=1}^{|\Omega|} \left(u_i^k\right)^2
\end{aligned}
\quad \text{and} \quad
\begin{aligned}
\left(\sigma^2(\mathbf{u})\right)^{k+1} &\leq \left(\sigma^2(\mathbf{u})\right)^k \\
\frac{\sum_{i=1}^{|\Omega|} \left(u_i^{k+1} - M(\mathbf{u}^{k+1})\right)^2}{|\Omega|} &\leq \frac{\sum_{i=1}^{|\Omega|} \left(u_i^{k} - M(\mathbf{u}^{k})\right)^2}{|\Omega|} \\
\sum_{i=1}^{|\Omega|} \left(u_i^{k+1} - M(\mathbf{u}^{k+1})\right)^2 &\leq \sum_{i=1}^{|\Omega|} \left(u_i^{k} - M(\mathbf{u}^{k})\right)^2\,,
\end{aligned} \tag{3.4}$$

since the number of pixels stays the same across iterations.

We thus see that if the average gray value (mean M) is preserved across the evolution (which it usually is, also for higher order processes), $\mathcal{L}_2$ stability is the same as monotonously decreasing variance.

We will soon need Gershgorin's circle theorem, therefore we introduce it here. It gives upper and lower bounds

$$|\lambda - a_{ii}| \leq \sum_{j \neq i} |a_{ij}|\,, \forall j \tag{3.5}$$

for the eigenvalues of a matrix [SB02]. This is called circle theorem because we can imagine these bounds to form a circle in the real/imaginary plane at center a_{ii} and with radius $\sum_{j \neq i} |a_{ij}|$.

3.2 Low Curvature Image Simplifier

In [TT99] Tumblin and Turk have introduced the so-called low curvature image simplifier (LCIS), an evolution equation which they use inside their LCIS hierarchy to perform detail-preserving contrast reduction of images.

Using our notations, their evolution equation is written

$$\boxed{\partial_t u = -\operatorname{div}\left(g\left(\|\mathcal{H}(u)\|_{\mathrm{Fr}}^2\right)\nabla\left(\Delta u\right)\right)}\,. \tag{3.6}$$

While their hierarchy does more than just apply the evolution equation [Tum99], their goal with the said equation is to be able to sort of segment an image into regions of large features and small features, where "large features" means e.g. a wall that is lit by a light and therefore exhibits many small gradients (radial gradient), and "small features" means small areas of high contrast details. Therefore they developed this higher order method to be able to create piecewise linear approximations and thus minimize the overall curvature (change in gradient) instead of the usual piecewise constant approximations which try to minimize the gradient itself.

The LCIS evolution equation (3.6) is not practical in the sense that it certainly cannot be written in variational formulation because inner and outer derivatives are not of the same order and therefore the inner and outer differentiation operators cannot be adjoint. One must also be careful with the nomenclature: this is an evolution equation but it is *not* diffusion.

In [Met08], Metzner analyzes the flux of this evolution method and concludes that it performs forward diffusion only, i.e. only smoothes the image and does not perform contrast enhancement. This probably means that without the entire LCIS hierarchy it might be of limited use. We have nevertheless chosen this method as a "warm-up" and a candidate for comparison in our exploration of higher order methods.

By the way, the homogeneous version of the LCIS evolution equation reduces to smoothing using the biharmonic operator

$$\partial_t u = -\operatorname{div}(\nabla\Delta u) = -\Delta^2 u\,. \tag{3.7}$$

The equation of Wei [Wei99]

$$\partial_t u = -\operatorname{div}\left(g\left(|\nabla u|^2\right)\nabla\left(\Delta u\right)\right) \tag{3.8}$$

employs another argument to diffusivity than the original LCIS evolution equation (3.6). We use it as an interesting variation of parameters for our experiments, since we have already seen in Section 2.6 that this is an important parameter. Results will be shown in the next chapter — for now we turn to the discretization and theoretical aspects.

3.2.1 Discretization

In this section, we show our discretization for the evolution equation presented in (3.6). We proceed analogously to the example in Section 2.4.2, while we simply write g as shortcut for $g(\|\mathcal{H}(u)\|^2_{\mathrm{Fr}})$:

$$\begin{aligned} \partial_t u &= -\operatorname{div}\left(g\,\nabla\left(\Delta u\right)\right) \\ &= -\partial_x\left(g\,\partial_x(\Delta u)\right) - \partial_y\left(g\,\partial_y(\Delta u)\right) \end{aligned}. \tag{3.9}$$

Using backward and forward finite differences for the outer and inner differentiation operators respectively, gives

$$\begin{aligned} \frac{\mathrm{d}u_{i,j}}{\mathrm{d}t} &\approx -\frac{(g\,\partial_x\Delta u)_{i,j} - (g\,\partial_x\Delta u)_{i-1,j}}{h_x} - \frac{(g\,\partial_y\Delta u)_{i,j} - (g\,\partial_y\Delta u)_{i,j-1}}{h_y} \\ &\approx -\frac{g_{i,j}(\partial_x\Delta u)_{i,j} - g_{i-1,j}(\partial_x\Delta u)_{i-1,j}}{h_x} - \frac{g_{i,j}(\partial_y\Delta u)_{i,j} - g_{i,j-1}(\partial_y\Delta u)_{i,j-1}}{h_y} \\ &\approx -\frac{g_{i,j}}{h_x^4}\left((u_{i,j} - 2u_{i+1,j} + u_{i+2,j}) - (u_{i-1,j} - 2u_{i,j} + u_{i+1,j})\right) \\ &\quad - \frac{g_{i,j}}{h_x^2 h_y^2}\left((u_{i+1,j-1} - 2u_{i+1,j} + u_{i+1,j+1}) - (u_{i,j-1} - 2u_{i,j} + u_{i,j+1})\right) \\ &\quad + \frac{g_{i-1,j}}{h_x^4}\left((u_{i-1,j} - 2u_{i,j} + u_{i+1,j}) - (u_{i-2,j} - 2u_{i-1,j} + u_{i,j})\right) \\ &\quad + \frac{g_{i-1,j}}{h_x^2 h_y^2}\left((u_{i,j-1} - 2u_{i,j} + u_{i,j+1}) - (u_{i-1,j-1} - 2u_{i-1,j} + u_{i-1,j+1})\right) \\ &\quad - \frac{g_{i,j}}{h_x^3 h_y}\left((u_{i-1,j+1} - 2u_{i,j+1} + u_{i+1,j+1}) - (u_{i-1,j} - 2u_{i,j} + u_{i+1,j})\right) \\ &\quad - \frac{g_{i,j}}{h_x h_y^2}\left((u_{i,j} - 2u_{i,j+1} + u_{i,j+2}) - (u_{i,j-1} - 2u_{i,j} + u_{i,j+1})\right) \\ &\quad + \frac{g_{i,j-1}}{h_x^3 h_y}\left((u_{i-1,j} - 2u_{i,j} + u_{i+1,j}) - (u_{i-1,j-1} - 2u_{i,j-1} + u_{i+1,j-1})\right) \\ &\quad + \frac{g_{i,j-1}}{h_x h_y^3}\left((u_{i,j-1} - 2u_{i,j} + u_{i,j+1}) - (u_{i,j-2} - 2u_{i,j-1} + u_{i,j})\right) . \end{aligned}$$

While one would not need to present a stencil here, as from an implementational point of view, one can simply insert an approximation for Δu instead of u into an existing implementation of EED, we nevertheless show the stencil in Figure 3.1 as it can help obtain stability results.

Alternative Discretization

Another suitable discretization for (3.6) could e.g. be

$$\frac{\mathrm{d}u_{i,j}}{\mathrm{d}t} \approx -\frac{(g\,\partial_x\Delta u)_{i+\frac{1}{2},j} - (g\,\partial_x\Delta u)_{i-\frac{1}{2},j}}{h_x} - \frac{(g\,\partial_y\Delta u)_{i,j+\frac{1}{2}} - (g\,\partial_y\Delta u)_{i,j-\frac{1}{2}}}{h_y}$$

	$i-2$	$i-1$	i	$i+1$	$i+2$
$j-2$	0	0	$\frac{-1}{2h_y^4}\Big(g_{i,j-1}+g_{i,j}\Big)$	0	0
$j-1$	0	$\frac{-1}{2h_x^2h_y^2}\Big(g_{i,j-1}+g_{i-1,j}+2g_{i,j}\Big)$	$\frac{1}{2h_x^2h_y^2}\Big(2g_{i,j-1}+g_{i-1,j}+4g_{i,j}+g_{i+1,j}\Big)+\frac{1}{2h_y^4}\Big(3g_{i,j-1}+4g_{i,j}+g_{i,j+1}\Big)$	$\frac{-1}{2h_x^2h_y^2}\Big(g_{i,j-1}+2g_{i,j}+g_{i+1,j}\Big)$	0
j	$\frac{-1}{2h_x^4}\Big(g_{i-1,j}+g_{i,j}\Big)$	$\frac{1}{2h_x^4}\Big(3g_{i-1,j}+4g_{i,j}+g_{i+1,j}\Big)+\frac{1}{2h_x^2h_y^2}\Big(g_{i,j-1}+2g_{i-1,j}+4g_{i,j}+g_{i,j+1}\Big)$	$\frac{-1}{2h_x^4}\Big(3g_{i-1,j}+6g_{i,j}+3g_{i+1,j}\Big)+\frac{-1}{2h_x^2h_y^2}\Big(2g_{i,j-1}+2g_{i-1,j}+8g_{i,j}+2g_{i+1,j}+2g_{i,j+1}\Big)+\frac{-1}{2h_y^4}\Big(3g_{i,j-1}+6g_{i,j}+3g_{i,j+1}\Big)$	$\frac{1}{2h_x^4}\Big(g_{i-1,j}+4g_{i,j}+3g_{i+1,j}\Big)\frac{1}{2h_x^2h_y^2}\Big(g_{i,j-1}+4g_{i,j}+2g_{i+1,j}+g_{i,j+1}\Big)$	$\frac{-1}{2h_x^4}\Big(g_{i,j}+g_{i+1,j}\Big)$
$j+1$	0	$\frac{-1}{2h_x^2h_y^2}\Big(g_{i-1,j}+2g_{i,j}+g_{i,j+1}\Big)$	$\frac{1}{2h_x^2h_y^2}\Big(g_{i-1,j}+4g_{i,j}+g_{i+1,j}+2g_{i,j+1}\Big)+\frac{1}{2h_y^4}\Big(g_{i,j-1}+4g_{i,j}+3g_{i,j+1}\Big)$	$\frac{-1}{2h_x^2h_y^2}\Big(2g_{i,j}+g_{i+1,j}+g_{i,j+1}\Big)$	0
$j+2$	0	0	$\frac{-1}{2h_y^4}\Big(g_{i,j}+g_{i,j+1}\Big)$	0	0

$\cdot u$

FIGURE 3.1: Stencil for the discretization of the isotropic LCIS evolution equation.

$$
\begin{aligned}
&\approx -\frac{g_{i+\frac{1}{2},j}(\partial_x \Delta u)_{i+\frac{1}{2},j} - g_{i-\frac{1}{2},j}(\partial_x \Delta u)_{i-\frac{1}{2},j}}{h_x} \\
&\quad - \frac{g_{i,j+\frac{1}{2}}(\partial_y \Delta u)_{i,j+\frac{1}{2}} - g_{i,j-\frac{1}{2}}(\partial_y \Delta u)_{i,j-\frac{1}{2}}}{h_y} \\
&\approx -\frac{\frac{g_{i+1,j}+g_{i,j}}{2}(u_{xxx}+u_{yyx})_{i+\frac{1}{2},j} - \frac{g_{i,j}+g_{i-1,j}}{2}(u_{xxx}+u_{yyx})_{i-\frac{1}{2},j}}{h_x} \\
&\quad - \frac{\frac{g_{i,j+1}+g_{i,j}}{2}(u_{xxy}+u_{yyy})_{i,j+\frac{1}{2}} - \frac{g_{i,j}+g_{i,j-1}}{2}(u_{xxy}+u_{yyy})_{i,j-\frac{1}{2}}}{h_y} \\
&\approx -\frac{\frac{g_{i+1,j}+g_{i,j}}{2}\left(\frac{(u_{xx})_{i+1,j}-(u_{xx})_{i,j}}{h_x} + \frac{(u_{yy})_{i+1,j}-(u_{yy})_{i,j}}{h_x}\right)}{h_x} \\
&\quad + \frac{\frac{g_{i,j}+g_{i-1,j}}{2}\left(\frac{(u_{xx})_{i,j}-(u_{xx})_{i-1,j}}{h_x} + \frac{(u_{yy})_{i,j}-(u_{yy})_{i-1,j}}{h_x}\right)}{h_x} \\
&\quad - \frac{\frac{g_{i,j+1}+g_{i,j}}{2}\left(\frac{(u_{xx})_{i,j+1}-(u_{xx})_{i,j}}{h_y} + \frac{(u_{yy})_{i,j+1}-(u_{yy})_{i,j}}{h_y}\right)}{h_y} \\
&\quad + \frac{\frac{g_{i,j}+g_{i,j-1}}{2}\left(\frac{(u_{xx})_{i,j}-(u_{xx})_{i,j-1}}{h_y} + \frac{(u_{yy})_{i,j}-(u_{yy})_{i,j-1}}{h_y}\right)}{h_y} \\
&\approx -\frac{g_{i+1,j}+g_{i,j}}{2h_x^4}\left(u_{i+2,j} - 2u_{i+1,j} + u_{i,j} - (u_{i+1,j} - 2u_{i,j} + u_{i-1,j})\right) \\
&\quad - \frac{g_{i+1,j}+g_{i,j}}{2h_x^4}\left(u_{i+1,j+1} - 2u_{i+1,j} + u_{i+1,j-1} - (u_{i,j+1} - 2u_{i,j} + u_{i,j-1})\right) \\
&\quad + \frac{g_{i,j}+g_{i-1,j}}{2h_x^4}\left(u_{i+1,j} - 2u_{i,j} + u_{i-1,j} - (u_{i,j} - 2u_{i-1,j} + u_{i-2,j})\right) \\
&\quad + \frac{g_{i,j}+g_{i-1,j}}{2h_x^2h_y^2}\left(u_{i,j+1} - 2u_{i,j} + u_{i,j-1} - (u_{i-1,j+1} - 2u_{i-1,j} + u_{i-1,j-1})\right) \\
&\quad - \frac{g_{i,j+1}+g_{i,j}}{2h_x^2h_y^2}\left(u_{i+1,j+1} - 2u_{i,j+1} + u_{i-1,j+1} - (u_{i+1,j} - 2u_{i,j} + u_{i-1,j})\right) \\
&\quad - \frac{g_{i,j+1}+g_{i,j}}{2h_y^4}\left(u_{i,j+2} - 2u_{i,j+1} + u_{i,j} - (u_{i,j+1} - 2u_{i,j} + u_{i,j-1})\right) \\
&\quad + \frac{g_{i,j}+g_{i,j-1}}{2h_x^2h_y^2}\left(u_{i+1,j} - 2u_{i,j} + u_{i-1,j} - (u_{i+1,j-1} - 2u_{i,j-1} + u_{i-1,j-1})\right) \\
&\quad + \frac{g_{i,j}+g_{i,j-1}}{2h_y^4}\left(u_{i,j+1} - 2u_{i,j} + u_{i,j-1} - (u_{i,j} - 2u_{i,j-1} + u_{i,j-2})\right) .
\end{aligned}
$$

The only difference here are the averaged diffusivities. This has no effect on the stability conditions that we just derived, as the averaged diffusivities are still bounded by $[0; 1]$.

3.2.2 Properties

Remember that implementing the easiest discrete scheme, the explicit scheme (Section 2.4.3), puts a limit on the time step size that one can use. This is due to the fact that stability criterions usually impose limits on the values of the entries of the system matrix, e.g. off-diagonals must not be negative [Wei98].

We can now use the stencil in Figure 3.1 (page 37) to compute an approximation of the eigenproperties of the system matrix. Remember, the stencil (left to right, top to bottom) represents the coefficients of one row of the system matrix, the empty cells explaining why those matrices usually are sparse: a stencil represents a very local transformation.

Symmetry

The system matrix for the LCIS evolution is not symmetric, although the row sums vanish. Let us show the non-symmetry which is quickly proven with a counter-example. Imagine the following to be cut from the midst of a system matrix, where $a_{(i,j)\to(k,l)}$ indicates, as before, the contribution of pixel (i,j) to pixel (k,l). From

$$\left(\begin{array}{ccccc} \ddots & & \vdots & & \\ & a_{(9,10)\to(9,10)} & \vdots & \frac{-1}{2h_x^4}\left(g_{(9,10)}+g_{(10,10)}\right) & \\ \cdots & \cdots & a_{(10,10)\to(10,10)} & \cdots & \cdots \\ & \frac{-1}{2h_x^4}\left(g_{(10,10)}+g_{(11,10)}\right) & \vdots & a_{(11,10)\to(11,10)} & \\ & & \vdots & & \ddots \end{array}\right), \tag{3.10}$$

where we have inserted only the necessary entries (i.e. the right- and leftmost entries from the stencil after an appropriate index shift), we see that the matrix is not symmetric (the entries with gray background are not the same).

$\mathcal{L}_2$ Stability

The LCIS evolution equation is not subject to maximum-minimum stability since off-diagonals are negative. We now try to find out what would be the restrictions (e.g. on the time step size) for it to adhere to $\mathcal{L}_2$ stability.

Note that, in theory, for a non-symmetric matrix, eigenvalues can be complex numbers. In this case however, the influence of the imaginary part seems to be very small as the values a_{ij} and a_{ji} are mostly very close to each other (see e.g. (3.10)); there certainly is no change in sign, as it would be for a skew-symmetric matrix ($a_{ij} = -a_{ji}$ for all i and j), which would make the eigenvalues truly complex (large imaginary part).

Considering the stencil in Figure 3.1, if we assume unit pixel grid size, i.e. $h_x = h_y = 1$, then by Gershgorin's circle theorem (3.5) we have $|\lambda + 20| \leq 44$, which implies $\lambda \in [-64; 24]$, since our usual diffusivity functions g are bounded by

the interval $[0;1]$. We can restrict ourselves to real values for previously mentioned reasons.

In [Did08], Didas has shown the condition for stability in the $\mathcal{L}_2$-norm, which we repeat in a syntactically slightly altered way as

$$\tau \leq \frac{2}{\sup_{s\in\mathbb{R}} g(s)\cdot\lambda_{\max}}\,, \tag{3.11}$$

where $\lambda_{\max}$ denotes the spectral norm of the (system) matrix A

$$\|A\| = \left\{\sqrt{|\lambda|} : \lambda \text{ eigenvalue of } A^\top A\right\}\,, \tag{3.12}$$

which is the largest singular value of the matrix and which more intuitively can also be read as "compute the eigenvalue range of A e.g. by means of the Gershgorin circle theorem, and then take the largest value by magnitude from that interval omitting the sign", e.g. if $\lambda_A \in \{-4,\dots,2\}$, then $\|A\| = 4$. This is because taking $A^\top A$ from A merely creates a symmetric matrix with squared eigenvalues, and we all know that $\sqrt{x^2} = |x|$, which is exactly the definition of singular value.

For the isotropic LCIS process, this gives a maximum time step for the explicit scheme of $\tau = \frac{2}{1\cdot 64} = \frac{1}{32} = 0.03125$. More formally, this can sometimes be seen by writing down the evolution equation (3.9)

$$\partial_t u = -\partial_x g u_{xxx} - \partial_x g u_{xyy} - \partial_y g u_{yxx} - \partial_y g u_{yyy}$$

in (discretized) matrix notation

$$\frac{u^{k+1}-u^k}{\tau} = -\underbrace{D_x\Phi D_{xxx}}_{A_1}u^k - \underbrace{D_x\Phi D_{xyy}}_{A_2}u^k - \underbrace{D_y\Phi D_{yxx}}_{A_3}u^k - \underbrace{D_y\Phi D_{yyy}}_{A_4}u^k\,, \tag{3.13}$$

where Φ is the diagonal matrix containing diffusivities g, and D stands for derivative, and one could for example use the approximations (written in stencil notation)

$$D_{xxx} = D_x\cdot D_{xx} \approx \frac{1}{h_x^3}\begin{array}{|c|c|c|c|}\hline 0&0&0&0\\\hline -1&3&3&1\\\hline 0&0&0&0\\\hline\end{array}\,,\quad D_{yyy} = D_y\cdot D_{yy} \approx \frac{1}{h_y^3}\begin{array}{|c|c|c|c|}\hline 0&-1&0&0\\\hline 0&3&0&0\\\hline 0&-3&0&0\\\hline 0&1&0&0\\\hline\end{array}\,,$$

$$D_{xyy} = D_x\cdot D_{yy} \approx \frac{1}{h_x h_y^2}\begin{array}{|c|c|c|}\hline 1&-1&0\\\hline -2&2&0\\\hline 1&-1&0\\\hline\end{array}\,,\quad D_{yxx} = D_y\cdot D_{xx} \approx \frac{1}{h_x^2 h_y}\begin{array}{|c|c|c|}\hline 1&-2&1\\\hline -1&2&-1\\\hline 0&0&0\\\hline\end{array}\,,$$

in which e.g. $D_x\cdot D_{xx}$ denotes stencil application as explained in Section 2.4.4, and standard discretizations for the second derivatives are used.

We know that, if, for two Hermitian matrices (in our case: real and symmetric) M_1 and M_2, the eigenvalues lie in the interval $[0;m_1]$ and $[0;m_2]$, respectively, then the matrix $M_1\cdot M_2$ has all its eigenvalues in $[0;m_1m_2]$.

Here, however, the matrices D_* and D_{***} are not symmetric (the asterisk denotes a wildcard). We could still apply Gershgorin's theorem but not the fact from the previous paragraph. In general however, $\|M_1 M_2\| \leq \|M_1\|\|M_2\|$. We can compute $\|D_*\|\|D_{***}\| = 16$, while $\|D_* D_{***}\|$ is thus less or equal, and we therefore have a minimal largest time step size (since it is inversely proportional). If we decide to look at the resulting matrices A_1 through A_4, we have $\|A_i\| = 16$ for all $i \in \{1, \ldots, 4\}$ (as $\|\Phi\| = 1$ and $g = h_x = h_y = 1$), so in this case even $\|D_* D_{***}\| = \|D_*\|\|D_{***}\|$ holds.

This verifies that the necessary and sufficient condition for stability, as postulated above, is

$$\tau \leq \frac{2}{1 \cdot (4 \cdot 16)} = \frac{1}{32} = 0.03125 \,.$$

Indeed, experiments show that this is correct, see Figure 3.2, where we have only modified the parameter τ. The artifacts that appear in the lower left figure are a paradigm for too large time steps or other major errors in the implementation. The center column contains several values of the image throughout the evolution: minimum, maximum, mean and variance in red, green, blue, and pink respectively. The variance is plotted against a second y-axis, which has a logarithmic scale, otherwise the other values would no longer be visible. One can see that already after few iterations, the process visibly destabilizes.

Further theory about this class of evolution equations can be found in [GB04].

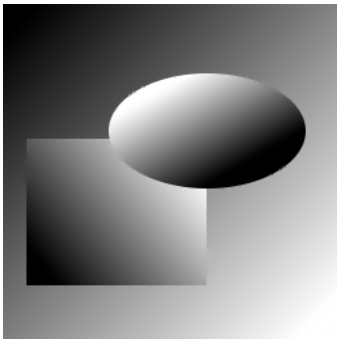
Preservation of Mean

Usually, one uses the combination of system matrix symmetry and unit row sums to show preservation of average gray value. In fact however, those two requirements are only there to guarantee unit *column* sums, which in turn guarantees the preservation of the mean all on its own. So, we do not need even symmetry, which is not given anyway, as we have seen above.

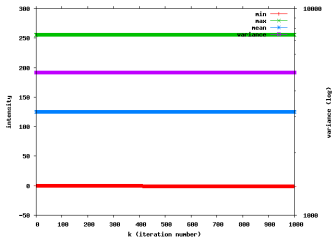
If we have $\sum_{i=1}^{N} a_{ij}^k = 1$, for all j, i.e. unit column sums (the contributions of one pixel to all other pixels sum up to one), then

$$\begin{aligned} \sum_{i=1}^{N} u_i^{k+1} &= \sum_{i=1}^{N} \sum_{j=1}^{N} a_{ij}^k u_j^k \\ &= \sum_{j=1}^{N} \sum_{i=1}^{N} a_{ij}^k u_j^k \\ &= \sum_{j=1}^{N} \underbrace{\left(\sum_{i=1}^{N} a_{ij}^k \right)}_{1} u_j^k = \sum_{j=1}^{N} u_j^k, \ \forall j \end{aligned} \tag{3.14}$$

proves the preservation of the average gray value. Unit column sums can in fact not be easily seen e.g. in the stencil, as the stencil represents one row and not one column of the system matrix.

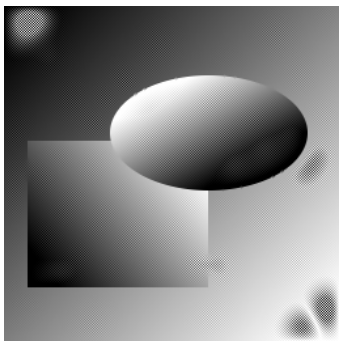


(a) Result at $t = 31$ with $\tau = 0.031$

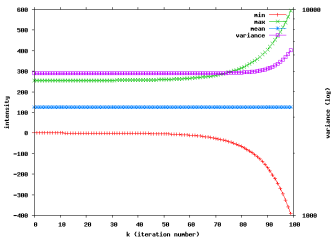


(b) Evolution

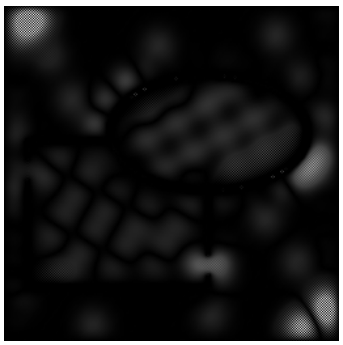
(c) Difference image after 1000 iterations



(d) Result at $t = 3.3$ with $\tau = 0.033$



(e) Evolution



(f) Difference image after 100 iterations

FIGURE 3.2: LCIS process at the verge of instability. ▶ **top**: stable ▶ **bottom**: unstable

Even if one somehow could look at the system matrix as a whole, one would still have to be careful as e.g. writing $g_{i,j}$ in one row (i.e. copying the elements of the stencil into each row of the matrix) does *not* equal the term $g_{i,j}$ from another row, as the tuples (i, j) are dependent on the center pixel to be considered in that matrix row p, i.e. $g_{p \mod N, \lfloor \frac{p}{N} \rfloor}$, for the reasons explained in Section 2.4.2.

Let us use the pixel at location $(10, 10)$ of an $N \times N$ image with $N = 256$ as an example. In Figure 3.3 (page 43), we show the corresponding system matrix, especially column $10 \cdot 256 + 10 = 2570$, where the center pixel (gray) is located in row 2570, consequently. Adding the different elements of diffusivity $(g_{*,*})$ from that column together, one can see that they cancel out.

3.2.3 Anisotropic LCIS

Our extension toward anisotropic LCIS is straightforward:

$$\partial_t u = -\operatorname{div}\left(D \cdot \nabla(\Delta u)\right) , \tag{3.15}$$

where D denotes the standard diffusion tensor.

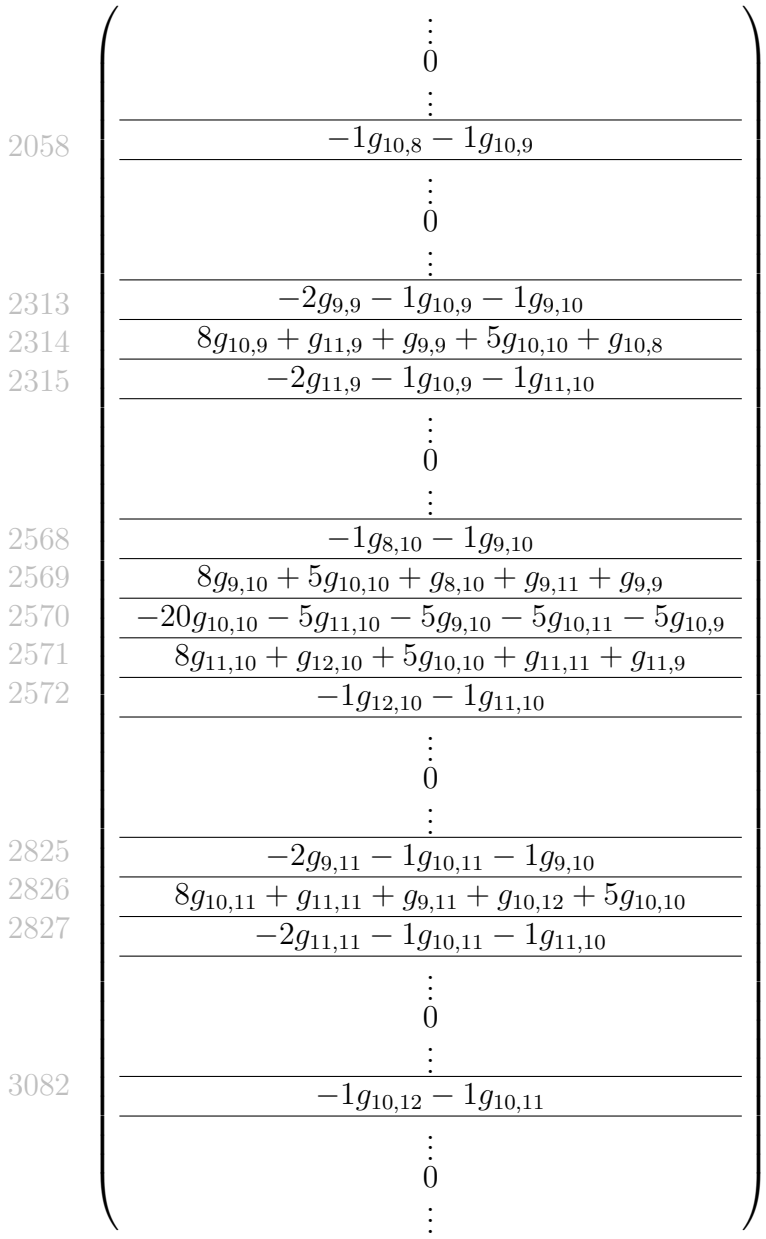


FIGURE 3.3: Column 2570 of LCIS system matrix for 256×256 image. ▶ **left**: row number ▶ **right**: corresponding destination pixel of contributions from source pixel at location (10,10) (matrix element (2570,2570))

3.3 The Frobenius Model

Nonlinear *isotropic* diffusion of order four was investigated e.g. in [Did08]:

$$\begin{aligned}\partial_t u = &- \partial_{xx}\left(g(\|\mathcal{H}(u)\|^2_{\mathrm{Fr}})u_{xx}\right) - \partial_{yx}\left(g(\|\mathcal{H}(u)\|^2_{\mathrm{Fr}})u_{xy}\right)\\ &- \partial_{xy}\left(g(\|\mathcal{H}(u)\|^2_{\mathrm{Fr}})u_{yx}\right) - \partial_{yy}\left(g(\|\mathcal{H}(u)\|^2_{\mathrm{Fr}})u_{yy}\right)\ .\end{aligned} \tag{3.16}$$

Note the inner and outer differentiations of the mixed terms being switched, this is done because ∂_{xy} and ∂_{yx} are adjoint operators and in matrix notation the adjoint of the matrix A_{xy}

$$A^*_{xy} = A^\top_{xy} \tag{3.17}$$

is just its transpose (because it is real-valued). In matrix notation one therefore can elegantly write e.g. $D^\top_{**}\Phi D_{**}$ for the contributions to the system matrix.

In 1-D, (3.16) can again be generalized to nonlinear isotropic diffusion of order $2n$ [Did08]:

$$\partial_t u = (-1)^{n+1}\partial^n_x\left(g((\partial^n_x u)^2)\partial^n_x u\right)\ . \tag{3.18}$$

3.3.1 Higher Order Generalization

The extension from the general lower order equation

$$\partial_t u = \mathrm{div}(D\cdot\nabla u)$$

seems pretty straightforward: we replace gradients and their norm by the Hessian matrix and its associated norm, the Frobenius norm, respectively. Of course, we also have to adapt the outer differentiation operator. Inspired by the lower order methods like EED we use a higher order analogon to the divergence: entry-wise second order derivatives.

The general equation for the Frobenius[1] model then is

$$\partial_t u = -\begin{pmatrix}\partial_{xx} & \partial_{xy}\\ \partial_{yx} & \partial_{yy}\end{pmatrix} : (D \bullet \mathcal{H}(u))\ , \tag{3.19}$$

where : denotes the Frobenius inner product [HJ90]

$$\begin{aligned}\mathbf{A} : \mathbf{B} &= \begin{pmatrix}a_{11} & a_{12}\\ a_{21} & a_{22}\end{pmatrix} : \begin{pmatrix}b_{11} & b_{12}\\ b_{21} & b_{22}\end{pmatrix}\\ &= a_{11}b_{11} + a_{12}b_{12} + a_{21}b_{21} + a_{22}b_{22}\\ &= \mathrm{tr}(\mathbf{A}^\top\mathbf{B})\ ,\end{aligned} \tag{3.20}$$

i.e. the component-wise (inner) product of two matrices, which, as we see, can also be written using the trace.

[1] Ferdinand Georg Frobenius (October 26, 1849 — August 3, 1917) was a German mathematician, best-known for his contributions to the theory of differential equations and to group theory.

We have found that we also need to replace the standard matrix multiplication found inside the divergence term in EED. It could theoretically still be used, as it does not only allow matrices and vectors to be multiplied, as it is the case for EED, but also two matrices. We however found that this does no longer work as expected. Therefore, we also have to generalize it. We choose to employ the preconditioning product (as in [BDFW07]), using which the general equation then finally becomes

$$\boxed{\partial_t u = -\begin{pmatrix} \partial_{xx} & \partial_{xy} \\ \partial_{yx} & \partial_{yy} \end{pmatrix} : (D \bullet_P \mathcal{H}(u))}, \tag{3.21}$$

where $A \bullet_P B = A^{\frac{1}{2}} B A^{\frac{1}{2}} = \sqrt{A} B \sqrt{A}$ and applying the square root to a matrix in our case means applying the square root to the eigenvalues during PAT (construction of the diffusion tensor)

$$\sqrt{D} := \begin{pmatrix} v_{1_x} & v_{2_x} \\ v_{1_y} & v_{2_y} \end{pmatrix} \begin{pmatrix} \sqrt{\lambda_1} & 0 \\ 0 & \sqrt{\lambda_2} \end{pmatrix} \begin{pmatrix} v_{1_x} & v_{1_y} \\ v_{2_x} & v_{2_y} \end{pmatrix} \text{, usually with } \lambda_2 = 1.$$

This model is motivated by the fact that computing the second directional derivative of u in direction of ζ evaluates the quadratic form

$$u_{\zeta\zeta} = \zeta^\top \mathcal{H}(u) \zeta \,, \tag{3.22}$$

and using the preconditioning product, one exactly modifies the vector ζ, as

$$\zeta^\top \sqrt{D}^\top \mathcal{H}(u) \sqrt{D} \zeta = (\sqrt{D}\zeta)^\top \mathcal{H}(u) \sqrt{D} \zeta \,. \tag{3.23}$$

This generalization is of no detriment to the isotropic case (scalar-valued diffusivity), as the preconditioning product then degenerates to standard scalar multiplication for which $\sqrt{g} B \sqrt{g} = g \cdot B$ holds. In that case, we have exactly the isotropic case (3.16).

3.3.2 Flux Analysis

The argument to diffusivity is quite important for the analysis of the flux, i.e. whether the method performs forward or backward diffusion (and in which cases). For the sake of simplicity we perform this analysis in 1-D and for the isotropic case, in which the Frobenius model is

$$\partial_t u = -\partial_{xx} \left(g(s) \cdot u_{xx} \right) \tag{3.24}$$

with

$$\nabla u \to u_x \qquad \text{and} \qquad \mathcal{H}(u) \to u_{xx} \,.$$

We differentiate using product and chain rules (remember, u is a function too) for two different arguments to diffusivity. Part of this has also been treated in [Did08, Met08].

$s = u_x^2$:

$$\begin{aligned}
\partial_t u &= -\partial_{xx}\left(g(u_x^2)u_{xx}\right) \\
&= -\partial_x\left(g'(u_x^2)2u_x u_{xx}^2 + g(u_x^2)u_{xxx}\right) \\
&= -(g''(u_x^2)2u_x u_{xx}\cdot 2u_x u_{xx}^2 + g'(u_x^2)\cdot 2(u_{xx}u_{xx}^2 + u_x 2u_{xx}u_{xxx}) \\
&\quad + g'(u_x^2)2u_x u_{xx}\cdot u_{xxx} + g(u_x^2)\cdot u_{xxxx}) \\
&= -(g''(u_x^2)4u_x^2 u_{xx}^3 + g'(u_x^2)(2u_{xx}^3 + 4u_x u_{xx}u_{xxx}) \\
&\quad + g'(u_x^2)2u_x u_{xx}u_{xxx} + g(u_x^2)u_{xxxx}) \\
&= -u_{xx}\left(g''(u_x^2)4u_x^2 u_{xx}^2 + g'(u_x^2)(2u_{xx}^2 + 6u_x u_{xxx})\right) \\
&\quad - u_{xxxx}g(u_x^2)
\end{aligned} \tag{3.25a}$$

$s = u_{xx}^2$:

$$\begin{aligned}
\partial_t u &= -\partial_{xx}\left(g(u_{xx}^2)u_{xx}\right) \\
&= -\partial_x\left(g'(u_{xx}^2)2u_{xx}^2 u_{xxx} + g(u_{xx}^2)u_{xxx}\right) \\
&= -(g''(u_{xx}^2)4u_{xx}^3 u_{xxx}^2 + g'(u_{xx}^2)\cdot 2(2u_{xx}u_{xxx}^2 + u_{xx}^2 u_{xxxx}) \\
&\quad + g'(u_{xx}^2)2u_{xx}u_{xxx}^2 + g(u_{xx}^2)\cdot u_{xxxx}) \\
&= -(g''(u_{xx}^2)4u_{xx}^3 u_{xxx}^2 + g'(u_{xx}^2)(6u_{xx}u_{xxx}^2 + 2u_{xx}^2 u_{xxxx}) + g(u_{xx}^2)u_{xxxx}) \\
&= -u_{xx}\left(g''(u_{xx}^2)4u_{xx}^2 u_{xxx}^2 + g'(u_{xx}^2)6u_{xxx}^2\right) \\
&\quad - u_{xxxx}\left(g'(u_{xx}^2)2u_{xx}^2 + g(u_{xx}^2)\right)
\end{aligned} \tag{3.25b}$$

In the first equation (3.25a) we see that fourth order diffusion is forward only, as it merely depends on the diffusivity, which is always positive. Furthermore, we are in presence of odd order derivatives for the second order term, so we cannot distinguish cases for forward or backward second order diffusion.

In the second equation (3.25b) we have both second order and fourth order diffusion solely depending on the diffusivity function and its derivatives. All other terms appear squared and therefore are positive, so we can determine the sign of the leading terms by analyzing the derivatives of the diffusivity function.

One can conclude that using the (higher order) Frobenius norm of the Hessian harmonizes better with the overall higher order nature of the model than the gradient magnitude.

Furthermore, in the anisotropic setting we will need a method to find the principal directions of a higher order structure. The first idea that comes to mind is that the structure tensor for low order methods was a unique construct that we don't even need anymore: we could use the Hessian matrix and apply PAT to it directly. We however choose to stay with the traditional structure tensor, as we want to recognize structure starting at order one — the Hessian matrix could not do that, see Section 2.6.2. Sometimes, higher order is not synonymous to "better". However, we choose the Frobenius norm of the Hessian as the argument to diffusivity, as we would only like to stop diffusion starting at order two, i.e. linear variations in the image will be kept.

3.3.3 Discretization

(3.21) more verbosely written out yields

$$\begin{aligned}\partial_t u &= -\begin{pmatrix}\partial_{xx} & \partial_{xy}\\ \partial_{yx} & \partial_{yy}\end{pmatrix} : \left(\begin{pmatrix}a & b\\ b & c\end{pmatrix}\begin{pmatrix}u_{xx} & u_{xy}\\ u_{yx} & u_{yy}\end{pmatrix}\begin{pmatrix}a & b\\ b & c\end{pmatrix}\right)\\ &= -\partial_{xx}\left(a^2u_{xx} + ab(u_{xy}+u_{yx}) + b^2u_{yy}\right)\\ &\quad - \partial_{xy}\left(abu_{xx} + acu_{xy} + b^2u_{yx} + bcu_{yy}\right)\\ &\quad - \partial_{yx}\left(abu_{xx} + b^2u_{xy} + acu_{yx} + bcu_{yy}\right)\\ &\quad - \partial_{yy}\left(b^2u_{xx} + bc(u_{xy}+u_{yx}) + c^2u_{yy}\right)\ .\end{aligned} \tag{3.26}$$

Using the standard discretizations

$$u_{xx} \approx \frac{1}{h_x^2}\begin{array}{|c|c|c|}\hline 0 & 0 & 0\\ \hline 1 & -2 & 1\\ \hline 0 & 0 & 0\\ \hline\end{array}\cdot u\ , \qquad u_{yy} \approx \frac{1}{h_y^2}\begin{array}{|c|c|c|}\hline 0 & 1 & 0\\ \hline 0 & -2 & 0\\ \hline 0 & 1 & 0\\ \hline\end{array}\cdot u\ ,$$

as well as the special discretizations [LLT03]

$$u_{xy} \approx \frac{1}{2h_xh_y}\begin{array}{|c|c|c|}\hline 1 & -1 & 0\\ \hline -1 & 2 & -1\\ \hline 0 & -1 & 1\\ \hline\end{array}\cdot u\ , \qquad u_{yx} \approx \frac{1}{2h_yh_x}\begin{array}{|c|c|c|}\hline 0 & 1 & -1\\ \hline 1 & -2 & 1\\ \hline -1 & 1 & 0\\ \hline\end{array}\cdot u\ ,$$

that can be derived using Taylor's theorem (see Section 2.4.1), (3.26) becomes

$$\begin{aligned}\frac{\mathrm{d}u_{i,j}}{\mathrm{d}t} \approx &-\frac{1}{h_x^2}\Big(\left(a^2_{i+1,j}(u_{xx})_{i+1,j} + (ab)_{i+1,j}\left((u_{xy})_{i+1,j} + (u_{yx})_{i+1,j}\right) + b^2_{i+1,j}(u_{yy})_{i+1,j}\right)\\ &\quad - 2\left(a^2_{i,j}(u_{xx})_{i,j} + (ab)_{i,j}\left((u_{xy})_{i,j} + (u_{yx})_{i,j}\right) + b^2_{i,j}(u_{yy})_{i,j}\right)\\ &\quad + \left(a^2_{i-1,j}(u_{xx})_{i-1,j} + (ab)_{i-1,j}\left((u_{xy})_{i-1,j} + (u_{yx})_{i-1,j}\right) + b^2_{i-1,j}(u_{yy})_{i-1,j}\right)\Big)\\ &- \frac{1}{2h_xh_y}\Big(\big((ab)_{i-1,j-1}(u_{xx})_{i-1,j-1} + (ac)_{i-1,j-1}(u_{xy})_{i-1,j-1}\\ &\qquad\qquad + b^2_{i-1,j-1}(u_{yx})_{i-1,j-1} + (bc)_{i-1,j-1}(u_{yy})_{i-1,j-1}\big)\\ &\quad - \big((ab)_{i,j-1}(u_{xx})_{i,j-1} + (ac)_{i,j-1}(u_{xy})_{i,j-1} + b^2_{i,j-1}(u_{yx})_{i,j-1} + (bc)_{i,j-1}(u_{yy})_{i,j-1}\big)\\ &\quad - \big((ab)_{i-1,j}(u_{xx})_{i-1,j} + (ac)_{i-1,j}(u_{xy})_{i-1,j} + b^2_{i-1,j}(u_{yx})_{i-1,j} + (bc)_{i-1,j}(u_{yy})_{i-1,j}\big)\\ &\quad + 2\left((ab)_{i,j}(u_{xx})_{i,j} + (ac)_{i,j}(u_{xy})_{i,j} + b^2_{i,j}(u_{yx})_{i,j} + (bc)_{i,j}(u_{yy})_{i,j}\right)\\ &\quad - \big((ab)_{i+1,j}(u_{xx})_{i+1,j} + (ac)_{i+1,j}(u_{xy})_{i+1,j} + b^2_{i+1,j}(u_{yx})_{i+1,j} + (bc)_{i+1,j}(u_{yy})_{i+1,j}\big)\\ &\quad - \big((ab)_{i,j+1}(u_{xx})_{i,j+1} + (ac)_{i,j+1}(u_{xy})_{i,j+1} + b^2_{i,j+1}(u_{yx})_{i,j+1} + (bc)_{i,j+1}(u_{yy})_{i,j+1}\big)\\ &\quad + \big((ab)_{i+1,j+1}(u_{xx})_{i+1,j+1} + (ac)_{i+1,j+1}(u_{xy})_{i+1,j+1}\\ &\qquad\qquad + b^2_{i+1,j+1}(u_{yx})_{i+1,j+1} + (bc)_{i+1,j+1}(u_{yy})_{i+1,j+1}\big)\Big)\\ &- \frac{1}{2h_yh_x}\Big(\big((ab)_{i,j-1}(u_{xx})_{i,j-1} + b^2_{i,j-1}(u_{xy})_{i,j-1}\end{aligned}$$

$$
\begin{aligned}
&\qquad + (ac)_{i,j-1}(u_{yx})_{i,j-1} + (bc)_{i,j-1}(u_{yy})_{i,j-1}\big) \\
&- \big((ab)_{i+1,j-1}(u_{xx})_{i+1,j-1} + b^2_{i+1,j-1}(u_{xy})_{i+1,j-1} \\
&\qquad + (ac)_{i+1,j-1}(u_{yx})_{i+1,j-1} + (bc)_{i+1,j-1}(u_{yy})_{i+1,j-1}\big) \\
&+ \big((ab)_{i-1,j}(u_{xx})_{i-1,j} + b^2_{i-1,j}(u_{xy})_{i-1,j} + (ac)_{i-1,j}(u_{yx})_{i-1,j} + (bc)_{i-1,j}(u_{yy})_{i-1,j}\big) \\
&- 2\big((ab)_{i,j}(u_{xx})_{i,j} + b^2_{i,j}(u_{xy})_{i,j} + (ac)_{i,j}(u_{yx})_{i,j} + (bc)_{i,j}(u_{yy})_{i,j}\big) \\
&+ \big((ab)_{i+1,j}(u_{xx})_{i+1,j} + b^2_{i+1,j}(u_{xy})_{i+1,j} + (ac)_{i+1,j}(u_{yx})_{i+1,j} + (bc)_{i+1,j}(u_{yy})_{i+1,j}\big) \\
&- \big((ab)_{i-1,j+1}(u_{xx})_{i-1,j+1} + b^2_{i-1,j+1}(u_{xy})_{i-1,j+1} \\
&\qquad + (ac)_{i-1,j+1}(u_{yx})_{i-1,j+1} + (bc)_{i-1,j+1}(u_{yy})_{i-1,j+1}\big) \\
&+ \big((ab)_{i,j+1}(u_{xx})_{i,j+1} + b^2_{i,j+1}(u_{xy})_{i,j+1} + (ac)_{i,j+1}(u_{yx})_{i,j+1} + (bc)_{i,j+1}(u_{yy})_{i,j+1}\big)\Big) \\
&- \frac{1}{h_y^2}\Big(\big(b^2_{i,j+1}(u_{xx})_{i,j+1} + b_{i,j+1}c_{i,j+1}\left((u_{xy})_{i,j+1} + (u_{yx})_{i,j+1}\right) + c^2_{i,j+1}(u_{yy})_{i,j+1}\big) \\
&- 2\big(b^2_{i,j}(u_{xx})_{i,j} + b_{i,j}c_{i,j}\left((u_{xy})_{i,j} + (u_{yx})_{i,j}\right) + c^2_{i,j}(u_{yy})_{i,j}\big) \\
&+ \big(b^2_{i,j-1}(u_{xx})_{i,j-1} + b_{i,j-1}c_{i,j-1}\left((u_{xy})_{i,j-1} + (u_{yx})_{i,j-1}\right) + c^2_{i,j-1}(u_{yy})_{i,j-1}\big)\Big)\,,
\end{aligned}
$$

which, after tedious rearrangement (see Appendix A), yields the stencil in Figure 3.4. A simplified stencil ($h_x = h_y = 1$) can also be found in Figure A.1 (page A–11).

Unfortunately, as with low order EED, we cannot be certain of the sign of some of the elements of the system matrix (cells of the stencil). If we look at the traditional structure tensor

$$
D = \begin{pmatrix} a & b \\ b & c \end{pmatrix} = \left(\begin{array}{c|c} g v_{1_x}^2 + v_{2_x}^2 & g v_{1_x} v_{1_y} + v_{2_x} v_{2_y} \\ \hline g v_{1_x} v_{1_y} + v_{2_x} v_{2_y} & g v_{1_y}^2 + v_{2_y}^2 \end{array}\right), \tag{3.27}
$$

we have several facts at our disposal:

1. The diffusivity function $g()$ is bounded by the interval $[0;1]$.
2. Eigenvectors $\mathbf{v_1}$ and $\mathbf{v_2}$ are normalized, i.e. $v_{1_x}^2 + v_{1_y}^2 = v_{2_x}^2 + v_{2_y}^2 = 1$.
3. Eigenvectors are perpendicular to each other: $\mathbf{v_1} \perp \mathbf{v_2}$, i.e. $\mathbf{v_2} = (\pm v_{1_y}, \mp v_{1_x})^\top$.

This results in a and c being bounded by the interval $[0;1]$.

$$
\begin{aligned}
a_{\max} &= g_{\max} \cdot v_{1_x}^2 + v_{2_x}^2 & c_{\max} &= g_{\max} \cdot v_{1_y}^2 + v_{2_y}^2 \\
&= 1 \cdot v_{1_x}^2 + (\pm v_{1_y})^2 & &= 1 \cdot v_{1_y}^2 + (\pm v_{1_x})^2 \\
&= 1 \cdot (v_{1_x}^2 + v_{1_y}^2) & &= 1 \cdot (v_{1_y}^2 + v_{1_x}^2) \\
&= 1 & &= 1
\end{aligned} \tag{3.28}
$$

Obviously, all terms are positive, therefore the lower bound.

For b, we have

$$\begin{aligned} b_{\max} &= g \cdot v_{1_x} v_{1_y} + v_{2_x} v_{2_y} \\ &= g \cdot v_{1_x} v_{1_y} - v_{1_y} v_{1_x} \\ &= \underbrace{(g-1)}_{\in[-1;0]} \cdot \underbrace{v_{1_x} v_{1_y}}_{\in[-0.5;0.5]} , \end{aligned} \tag{3.29}$$

where the bounds for $v_{1_x} v_{1_y}$ can be seen when expressing the vector $\mathbf{v_1}$ in polar coordinates

$$v_{1_x} = \cos(\varphi) \quad , \quad v_{1_y} = \sin(\varphi)$$

and examining the function $f(\varphi) = \cos(\varphi)\sin(\varphi)$, which has its minimum and maximum at -0.5 and 0.5, respectively. Therefore b is bounded by the interval $[-0.5; 0.5]$, which unfortunately includes negative values.

Because the sign of the mixed terms including b is not known a priori, one cannot guarantee non-negativity of the off-diagonal elements of the system matrix. For lower-order methods, there are ways to obtain a non-negative discretization, limiting the anisotropy at the same time [Wei98]. Since our method however does not succumb to a maximum-minimum principle, but only the weaker $\mathcal{L}_2$-stability, such a discretization cannot be obtained.

3.3.4 Interpretation as an Energy Functional

So far, we have worked with the divergence expression only. However we believe that this higher order evolution equation can be nicely recast as an energy functional.

In fact, the primary motivation of choosing a "symmetric" divergence expression was toward the goal of maybe finding an energy minimization formulation, which at least needs inner and outer derivatives of same order. As we have mentioned before, it is impossible to write the LCIS evolution equation (Section 3.2) as variational formulation exactly because of that reason.

Since we started with the divergence expression and there is no clear way to reverse engineer a functional, we found a corresponding functional by intuition and trial and error. At this point, we remind the reader about the Euler-Lagrange equation (2.42) introduced in Section 2.5. In our case, E_{u_x} and E_{u_y} both vanish and E_u contains e.g. the differentiated standard data term (squared difference)

$$E_u = 2(u-f)(1-0) = 2(u-f) \, , \tag{3.30}$$

which yields the time discretization together with the regularization parameter, which becomes a purely numerical parameter [RSW99].

Now, if we look closely at the remaining second order differentiations, it is not hard to see that we could write

$$\frac{\partial}{\partial xx} E_{u_{xx}} + \frac{\partial}{\partial xy} E_{u_{xy}} + \frac{\partial}{\partial yx} E_{u_{yx}} + \frac{\partial}{\partial yy} E_{u_{yy}} = \begin{pmatrix} \partial_{xx} & \partial_{xy} \\ \partial_{yx} & \partial_{yy} \end{pmatrix} : \begin{pmatrix} E_{u_{xx}} & E_{u_{xy}} \\ E_{u_{yx}} & E_{u_{yy}} \end{pmatrix} , \tag{3.31}$$

	$i-2$	$i-1$	i	$i+1$	$i+2$
$j-2$	$\frac{1}{4h_x^2h_y^2}\left(-(ac)_{i-1,j-1}\right)$	$\frac{1}{4h_x^2h_y^2}\left(-b_{i-1,j-1}^2-b_{i,j-1}^2+(ac)_{i-1,j-1}+(ac)_{i,j-1}\right)+\frac{1}{4h_xh_y^3}\left(-2(bc)_{i-1,j-1}-(bc)_{i,j-1}\right)$	$\frac{1}{4h_x^2h_y^2}\left(b_{i-1,j-1}^2+2b_{i,j-1}^2+b_{i+1,j-1}^2-2(ac)_{i,j-1}\right)+\frac{1}{h_y^4}\left(-c_{i,j-1}^2\right)$	$\frac{1}{4h_x^2h_y^2}\left(-b_{i+1,j-1}^2-b_{i,j-1}^2+(ac)_{i+1,j-1}+(ac)_{i,j-1}\right)+\frac{1}{4h_xh_y^3}\left(2(bc)_{i+1,j-1}+(bc)_{i,j-1}\right)$	$\frac{1}{4h_x^2h_y^2}\left(-(ac)_{i+1,j-1}\right)$
$j-1$	$\frac{1}{4h_x^3h_y}\left(-2(ab)_{i-1,j-1}-(ab)_{i-1,j}\right)+\frac{1}{4h_x^2h_y^2}\left(-b_{i-1,j-1}^2-b_{i-1,j}^2+(ac)_{i-1,j-1}+(ac)_{i-1,j}\right)$	$\frac{1}{4h_x^3h_y}\left(4(ab)_{i-1,j-1}+2(ab)_{i,j}\right)+\frac{1}{4h_x^2h_y^2}\left(2b_{i-1,j-1}^2-2b_{i,j-1}^2+2b_{i,j}^2-2(ac)_{i-1,j-1}-2(ac)_{i-1,j}-2(ac)_{i,j}\right)+\frac{1}{4h_xh_y^3}\left(4(bc)_{i-1,j-1}+2(bc)_{i,j}\right)$	$\frac{1}{4h_x^3h_y}\left(-2(ab)_{i-1,j-1}+2(ab)_{i+1,j-1}+(ab)_{i-1,j}-(ab)_{i+1,j}\right)+\frac{1}{4h_x^2h_y^2}\left(-b_{i-1,j-1}^2+4b_{i,j-1}^2-b_{i+1,j-1}^2-b_{i-1,j}^2+4b_{i,j}^2-b_{i+1,j}^2+(ac)_{i-1,j-1}+4(ac)_{i,j-1}+(ac)_{i+1,j-1}+(ac)_{i-1,j}+4(ac)_{i,j}+(ac)_{i+1,j}\right)+\frac{1}{h_y^4}\left(2c_{i,j-1}^2+2c_{i,j}^2\right)$	$\frac{1}{4h_x^3h_y}\left(-4(ab)_{i+1,j-1}-2(ab)_{i,j}\right)+\frac{1}{4h_x^2h_y^2}\left(2b_{i+1,j-1}^2-2b_{i,j-1}^2+2b_{i,j}^2-2(ac)_{i+1,j-1}-2(ac)_{i,j}-2(ac)_{i+1,j}+2(ac)_{i,j}\right)+\frac{1}{4h_xh_y^3}\left(-4(bc)_{i+1,j-1}-2(bc)_{i,j}\right)$	$\frac{1}{4h_x^3h_y}\left(2(ab)_{i+1,j-1}+(ab)_{i+1,j}\right)+\frac{1}{4h_x^2h_y^2}\left(-b_{i+1,j-1}^2-b_{i+1,j}^2+(ac)_{i+1,j-1}+(ac)_{i+1,j}\right)$
j	$\frac{1}{h_x^4}\left(-a_{i-1,j}^2\right)+\frac{1}{4h_x^2h_y^2}\left(b_{i-1,j-1}^2+2b_{i-1,j}^2+b_{i-1,j+1}^2-2(ac)_{i-1,j}\right)$	$\frac{1}{h_x^4}\left(2a_{i-1,j}^2+2a_{i,j}^2\right)+\frac{1}{4h_x^2h_y^2}\left(-b_{i-1,j-1}^2-b_{i,j-1}^2+4b_{i-1,j}^2+4b_{i,j}^2-b_{i-1,j+1}^2-b_{i,j+1}^2+(ac)_{i-1,j-1}+4(ac)_{i-1,j}+(ac)_{i-1,j+1}+(ac)_{i,j-1}+4(ac)_{i,j}+(ac)_{i,j+1}\right)+\frac{1}{4h_xh_y^3}\left(-2(bc)_{i-1,j-1}+(bc)_{i,j-1}+2(bc)_{i-1,j+1}-(bc)_{i,j+1}\right)$	$\frac{1}{h_x^4}\left(-a_{i-1,j}^2-4a_{i,j}^2-a_{i+1,j}^2\right)+\frac{1}{4h_x^2h_y^2}\left(2b_{i,j-1}^2+2b_{i-1,j}^2-2b_{i,j}^2+2b_{i,j+1}^2+2b_{i+1,j}^2-(ac)_{i-1,j-1}-2(ac)_{i,j-1}-(ac)_{i+1,j-1}-2(ac)_{i-1,j}-8(ac)_{i,j}-2(ac)_{i+1,j}-(ac)_{i-1,j+1}-2(ac)_{i,j+1}-(ac)_{i+1,j+1}\right)+\frac{1}{h_y^4}\left(-c_{i-1,j}^2-4c_{i,j}^2-c_{i+1,j}^2\right)$	$\frac{1}{h_x^4}\left(2a_{i+1,j}^2+2a_{i,j}^2\right)+\frac{1}{4h_x^2h_y^2}\left(-b_{i+1,j-1}^2-b_{i,j-1}^2+4b_{i+1,j}^2+4b_{i,j}^2-b_{i+1,j+1}^2-b_{i,j+1}^2+(ac)_{i+1,j-1}+4(ac)_{i+1,j}+(ac)_{i+1,j+1}+(ac)_{i,j-1}+4(ac)_{i,j}+(ac)_{i,j+1}\right)+\frac{1}{4h_xh_y^3}\left(-(bc)_{i,j-1}+2(bc)_{i+1,j-1}+(bc)_{i,j+1}-2(bc)_{i+1,j+1}\right)$	$\frac{1}{h_x^4}\left(-a_{i+1,j}^2\right)+\frac{1}{4h_x^2h_y^2}\left(b_{i+1,j-1}^2+2b_{i+1,j}^2+b_{i+1,j+1}^2-2(ac)_{i+1,j}\right)$
$j+1$	$\frac{1}{4h_x^3h_y}\left(2(ab)_{i-1,j+1}+(ab)_{i-1,j}\right)+\frac{1}{4h_x^2h_y^2}\left(-b_{i-1,j+1}^2-b_{i-1,j}^2+(ac)_{i-1,j+1}+(ac)_{i-1,j}\right)$	$\frac{1}{4h_x^3h_y}\left(-4(ab)_{i-1,j+1}-2(ab)_{i,j}\right)+\frac{1}{4h_x^2h_y^2}\left(2b_{i-1,j+1}^2-2b_{i,j+1}^2+2b_{i,j}^2-2(ac)_{i-1,j+1}-2(ac)_{i-1,j}-2(ac)_{i,j+1}+2(ac)_{i,j}\right)+\frac{1}{4h_xh_y^3}\left(-4(bc)_{i-1,j-1}-2(bc)_{i,j}\right)$	$\frac{1}{4h_x^3h_y}\left(2(ab)_{i-1,j+1}-2(ab)_{i+1,j+1}-(ab)_{i-1,j}+(ab)_{i+1,j}\right)+\frac{1}{4h_x^2h_y^2}\left(-b_{i-1,j+1}^2+4b_{i,j+1}^2-b_{i+1,j+1}^2-b_{i-1,j}^2+4b_{i,j}^2-b_{i+1,j}^2+(ac)_{i-1,j+1}+4(ac)_{i,j+1}+(ac)_{i+1,j+1}+(ac)_{i-1,j}+4(ac)_{i,j}+(ac)_{i+1,j}\right)+\frac{1}{h_y^4}\left(2c_{i,j+1}^2+2c_{i,j}^2\right)$	$\frac{1}{4h_x^3h_y}\left(4(ab)_{i+1,j+1}+2(ab)_{i,j}\right)+\frac{1}{4h_x^2h_y^2}\left(2b_{i+1,j+1}^2-2b_{i,j+1}^2+2b_{i,j}^2-2(ac)_{i+1,j+1}-2(ac)_{i,j+1}-2(ac)_{i+1,j}+2(ac)_{i,j}\right)+\frac{1}{4h_xh_y^3}\left(4(bc)_{i+1,j+1}+2(bc)_{i,j}\right)$	$\frac{1}{4h_x^3h_y}\left(-2(ab)_{i+1,j+1}-(ab)_{i+1,j}\right)+\frac{1}{4h_x^2h_y^2}\left(-b_{i+1,j+1}^2-b_{i+1,j}^2+(ac)_{i+1,j+1}+(ac)_{i+1,j}\right)$
$j+2$	$\frac{1}{4h_x^2h_y^2}\left(-(ac)_{i-1,j+1}\right)$	$\frac{1}{4h_x^2h_y^2}\left(-b_{i-1,j+1}^2-b_{i,j+1}^2+(ac)_{i-1,j+1}+(ac)_{i,j+1}\right)+\frac{1}{4h_xh_y^3}\left(2(bc)_{i-1,j+1}+(bc)_{i,j+1}\right)$	$\frac{1}{4h_x^2h_y^2}\left(b_{i-1,j+1}^2+2b_{i,j+1}^2+b_{i+1,j+1}^2-2(ac)_{i,j+1}\right)+\frac{1}{h_y^4}\left(-c_{i,j+1}^2\right)$	$\frac{1}{4h_x^2h_y^2}\left(-b_{i+1,j+1}^2-b_{i,j+1}^2+(ac)_{i+1,j+1}+(ac)_{i,j+1}\right)+\frac{1}{4h_xh_y^3}\left(-2(bc)_{i+1,j+1}-(bc)_{i,j+1}\right)$	$\frac{1}{4h_x^2h_y^2}\left(-(ac)_{i+1,j+1}\right)$

$\cdot u$

FIGURE 3.4: Stencil for the discretization of the anisotropic Frobenius model.

which already carries some resemblance to

$$\begin{aligned}&\begin{pmatrix}\partial_{xx} & \partial_{xy}\\ \partial_{yx} & \partial_{yy}\end{pmatrix} : \left(\sqrt{D}^{\top}\cdot\mathcal{H}(u)\cdot\sqrt{D}\right)\\ =&\begin{pmatrix}\partial_{xx} & \partial_{xy}\\ \partial_{yx} & \partial_{yy}\end{pmatrix} : \begin{pmatrix}a^2u_{xx}+ab(u_{xy}+u_{yx})+b^2u_{yy} & abu_{xx}+b^2u_{yx}+acu_{xy}+bcu_{yy}\\ abu_{xx}+b^2u_{yx}+acu_{xy}+bcu_{yy} & b^2u_{xx}+bc(u_{yx}+u_{xy})+c^2u_{yy}\end{pmatrix}.\end{aligned} \tag{3.32}$$

With $\sqrt{D}=\begin{pmatrix}a & b\\ b & c\end{pmatrix}$, we could set

$$E_{u_{xx}} = a^2u_{xx}+ab(u_{xy}+u_{yx})+b^2u_{yy}\,, \tag{3.33a}$$
$$E_{u_{xy}} = abu_{xx}+b^2u_{yx}+acu_{xy}+bcu_{yy}\,, \tag{3.33b}$$
$$E_{u_{yx}} = abu_{xx}+b^2u_{yx}+acu_{xy}+bcu_{yy}\,, \quad \text{and} \tag{3.33c}$$
$$E_{u_{yy}} = b^2u_{xx}+bc(u_{yx}+u_{xy})+c^2u_{yy}\,. \tag{3.33d}$$

Performing the differentiations as given in (3.31) would yield the following (rearranged) terms:

$$u_{xxxx}\cdot(a^2)+ \tag{3.34a}$$
$$u_{xxxy}\cdot(4ab)+ \tag{3.34b}$$
$$u_{xxyy}\cdot(4b^2+2ac)+ \tag{3.34c}$$
$$u_{xyyy}\cdot(4bc)+ \tag{3.34d}$$
$$u_{yyyy}\cdot(c^2) \tag{3.34e}$$

Now, it only takes a sharp look to see that this is exactly the sum of all second order derivatives of

$$(au_{xx}+bu_{xy}+bu_{yx}+cu_{yy})^2 = \left(\sqrt{D} : \mathcal{H}(u)\right)^2 \tag{3.35}$$

with regard to u. We have omitted a factor of 2 in (3.34), which is usually simplified out in the Euler-Lagrange equation $0 = \not{2}(u-f)+\ \not{2}\cdot\ldots$, since the left-hand side is zero.

Therefore, finally, the corresponding energy functional can be written as:

$$\boxed{\begin{aligned}E(u) &= \int_\Omega (u-f)^2+\alpha\left(\sqrt{D} : \mathcal{H}(u)\right)^2 dx\,dy\\ &= \int_\Omega (u-f)^2+\alpha\left(\operatorname{tr}\left(\sqrt{D}\cdot\mathcal{H}(u)\right)\right)^2 dx\,dy\end{aligned}}\,. \tag{3.36}$$

There is however an important detail: the energy functional as presented only models a *linear* anisotropic filter. In the nonlinear case, the entries of the tensor D change at each iteration and would thus be variable and needed to be differentiated too. Hence we can interpret this functional as an expression that needs to be minimized at each iteration so that the diffusion tensor arguments can be treated as constants.

3.3.5 Higher Order Boundary Conditions

As we have mentioned before, the exact boundary conditions (BC) for a certain method arise out of the discretization of the energy functional. Certain stencil items then comprise an additional indicator function that "switches" certain terms inside that item on or off in certain intervals, as we have e.g. done for methods in [Rot08]. The parabolic approach does not automatically include these conditions.

Intuition tells us that something must change in a higher order setting. In full analogy and as a continuation to Section 2.4.5, we propose to extend boundary conditions to higher order by not making the first derivatives, but the second derivatives zero across boundaries (second order boundary), which we call Frobenius boundary conditions

$$\frac{\partial^2}{\partial n^2} u_i = 0 \quad \text{for} \quad i \in \Gamma \,. \tag{3.37}$$

If second derivatives are discretized the standard way using $u'' \approx \frac{u_{i+1}-2u_i-u_{i-1}}{h^2}$ in either dimension, this comes down to setting pixels beyond e.g. the right border to

$$u(i,j) = 2u(i-1,j) - u(i-2,j)$$

and the left border to

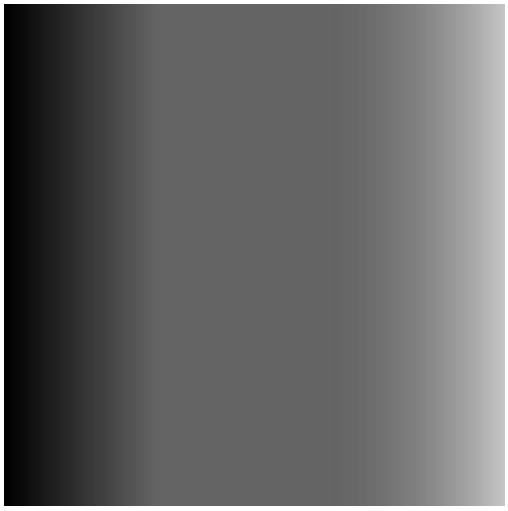
$$u(i,j) = 2u(i+1,j) - u(i+2,j) \,.$$

The image domain still is $\Omega = \{1,\dots,N\}$, but in contrast (and in accordance) to Section 2.4.5, the boundary must be enlarged to contain at least two (dummy) border pixels in each direction.

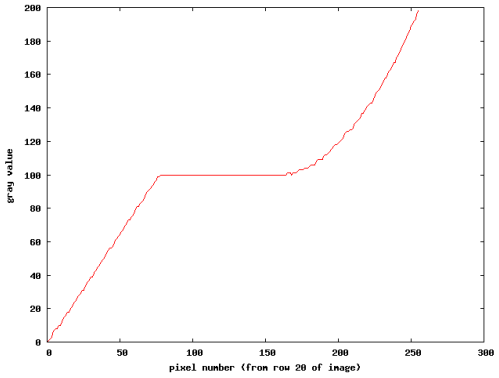
To illustrate the effects of boundary conditions, let us consider the test image in Figure 3.5(a) where the left part comprises a linear gradient, and the right part a square one. We extract one row of the image (all rows are the same) and display it as a 2-D plot in Figure 3.5(b).

Figure 3.5(c) then shows first order boundary conditions up to stopping time $t = 6250$ (in blue) using the nonlinear anisotropic Frobenius model and Perona/-Malik diffusivity with $\lambda = 1$. One can see that the further the evolution progresses (different colors in the plot), the more effect the boundary conditions have on the remaining image, we talk about propagation. Finally, Figure 3.5(d) shows the evolution of the test image using second order boundary conditions using the same parameters.

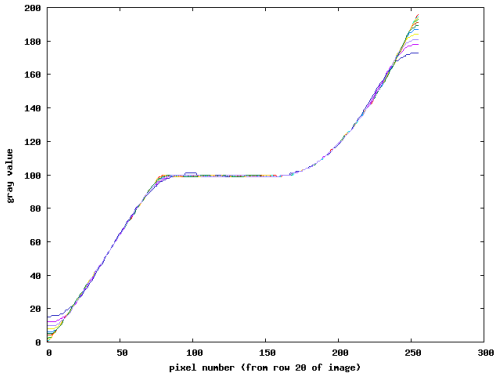
Why the right part of the image (the square gradient) is not evolved into a linear gradient is another question. This is probably due to the lack of backward diffusion in this case, since the linear gradient in the left part is not destroyed either, as would be the case for lower order methods using e.g. Perona/Malik: staircasing would occur.



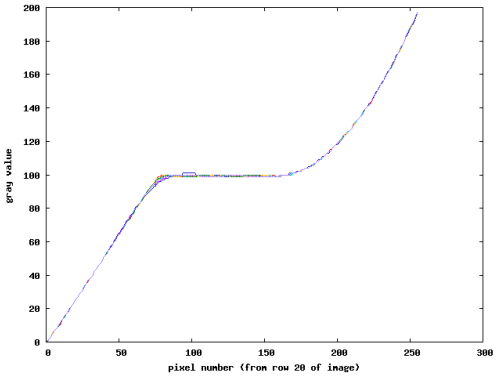
(a) Test image (Source:[Met08])



(b) Horizontal cut through test image



(c) Evolution using Neumann BC



(d) Evolution using Frobenius BC

FIGURE 3.5: Impact of Neumann and Frobenius boundary conditions on signal during evolution.

Chapter 4

Experiments

Thus far, we have seen a lot of theory, and Section 2.6 already showed a lot of model parameters. For a complete overview of all the methods we have mentioned so far, we refer to Table 4.2 (page 57). D always stands for the traditional structure tensor and the argument given to D stands for the argument that is given to the diffusivity function while constructing the diffusion tensor (see Section 2.2.2). Let us also recall the model parameters:

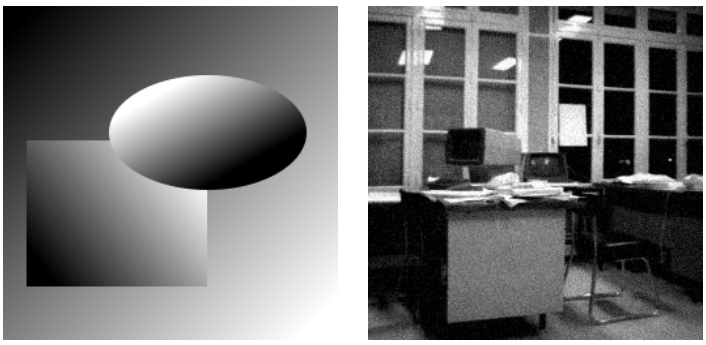
- σ: Standard deviation of Gaussian kernel used for convolution,
- $g(s)$: g: Diffusivity function, reduces diffusion in feature-rich regions; s: Argument to diffusivity function, feature indicator,
- λ: contrast parameter inside diffusivity function g; border between backward and forward diffusion[1]; intuitively, low lambda ($\lambda \ll 1$) makes g very small on a prominent feature (large s), high lambda ($\lambda \gg 1$) prevents g from getting too small when there is no feature (small s),
- D: Diffusion tensor; finds structure directions.

The stopping time $t = n \cdot \tau$ denotes the virtual time elapsed during the evolution, where n is the number of iterations and τ the time step size.

Table 4.2 already shows that there are many models to be tried — and there are many parameters listed just above. Note that we are not going to explore all those methods, we simply mention all of them to show that they exist and could be used in theory. We are going to explore those which we find most interesting, and of course those which have not been investigated – or even existed – before, to the best of our knowledge, like the anisotropic Frobenius model.

Difference in AAE units	Color Code
< 1	white
1–10	blue
10–50	green
50–150	yellow
> 150	red

TABLE 4.1: Color coding of difference images.



(a) `R&E` — synthetic image (b) `Office` — real world scene (noisy)

FIGURE 4.1: Synthetic and real world test images.

4.1 Error Measure and Difference Images

The average absolute error (AAE) in gray value units between the original image u_O and the result u_R after running an experiment is defined as

$$e_{\mathrm{AA}}(u_R, u_O) = \frac{1}{|\Omega_2|} \sum_{(x,y)\in\Omega_2} \left| u_R(x,y) - u_O(x,y) \right| , \tag{4.1}$$

where Ω_2 denotes the two-dimensional image domain and $|\Omega_2|$ consequently the number of pixels in the image.

Furthermore, we show difference images, in which white means "no change", whereas color shows the amount of change as indicated in Table 4.1. This has the advantage that we do not need to rescale a grayscale difference image for cases where the difference is small, which would make the comparison of the amount of change for different datasets less obvious.

[1] if the diffusivity function supports it

	Designation	Evolution Equation	OO	IO
Lower Order	HOM	$\partial_t u = \operatorname{div}(\nabla u) = \Delta u$	1	1
	ISO-L	$\partial_t u = \operatorname{div}(g_\nabla(f)\nabla u)$		
	ISO-NL	$\partial_t u = \operatorname{div}(g_\nabla(u)\nabla u)$		
	EED-L	$\partial_t u = \operatorname{div}(D_\nabla(f)\nabla u)$		
	EED-NL	$\partial_t u = \operatorname{div}(D_\nabla(u)\nabla u)$		
Wei	Wei-ISO-L	$\partial_t u = -\operatorname{div}(g_\nabla(f)\nabla\Delta u)$	1	3
	Wei-ISO-NL	$\partial_t u = -\operatorname{div}(g_\nabla(u)\nabla\Delta u)$		
	Wei-ANISO-L	$\partial_t u = -\operatorname{div}(D_\nabla(f)\nabla\Delta u)$		
	Wei-ANISO-NL	$\partial_t u = -\operatorname{div}(D_\nabla(u)\nabla\Delta u)$		
Tumblin/Turk	T/T-ISO-L	$\partial_t u = -\operatorname{div}(g_\mathcal{H}(f)\nabla\Delta u)$		
	T/T-ISO-NL	$\partial_t u = -\operatorname{div}(g_\mathcal{H}(u)\nabla\Delta u)$		
	T/T-ANISO-L	$\partial_t u = -\operatorname{div}(D_\mathcal{H}(f)\nabla\Delta u)$		
	T/T-ANISO-NL	$\partial_t u = -\operatorname{div}(D_\mathcal{H}(u)\nabla\Delta u)$		
Frobenius Model	FM1-ISO-L	$\partial_t u = -\mathcal{H} : (g_\nabla(f)\,\mathcal{H}(u))$	2	2
	FM1-ISO-NL	$\partial_t u = -\mathcal{H} : (g_\nabla(u)\,\mathcal{H}(u))$		
	FM1-ANISO-L	$\partial_t u = -\mathcal{H} : \left(\sqrt{D_\nabla(f)}\,\mathcal{H}(u)\,\sqrt{D_\nabla(f)}\right)$		
	FM1-ANISO-NL	$\partial_t u = -\mathcal{H} : \left(\sqrt{D_\nabla(u)}\,\mathcal{H}(u)\,\sqrt{D_\nabla(u)}\right)$		
	FM2-ISO-L	$\partial_t u = -\mathcal{H} : (g_\mathcal{H}(f)\,\mathcal{H}(u))$		
	FM2-ISO-NL	$\partial_t u = -\mathcal{H} : (g_\mathcal{H}(u)\,\mathcal{H}(u))$		
	FM2-ANISO-L	$\partial_t u = -\mathcal{H} : \left(\sqrt{D_\mathcal{H}(f)}\,\mathcal{H}(u)\,\sqrt{D_\mathcal{H}(f)}\right)$		
	FM2-ANISO-NL	$\partial_t u = -\mathcal{H} : \left(\sqrt{D_\mathcal{H}(u)}\,\mathcal{H}(u)\,\sqrt{D_\mathcal{H}(u)}\right)$		

TABLE 4.2: Overview of evolution equations. Abbreviations: ▶ **ISO/ANISO**: isotropic/anisotropic ▶ **L/NL**: linear/nonlinear ▶ **OO**: outer order ▶ **IO**: inner order ▶ $\varphi_\nabla(\cdot) := \varphi(|\nabla \cdot|^2)$, $\varphi_\mathcal{H}(\cdot) := \varphi(\|\mathcal{H}(\cdot)\|_{\mathrm{Fr}}^2)$

4.2 Experimental Setup

We perform our evaluation using two datasets. We call the first `R&E`, as abbreviation for "Rectangle and Ellipse", as seen in Figure 4.1(a). We put all higher order nonlinear methods for `R&E` side by side in Figures 4.3, 4.4, 4.6 and 4.7, for two parameter sets. Note that we use extreme stopping times, in order to illustrate the true behavior of those methods. We give the experiments in the linear setting in Figures 4.10 and 4.11 for the sake of completeness and also only for this non-noisy test image, as linear methods by design tend to keep the noise for a long time. We compare to results using lower order processes in Figure 4.9.

The second dataset is the (noisy) `Office` image seen in Figure 4.1(b). It is degraded with additive white Gaussian noise (AWGN) with a standard deviation of $\sigma = 10$. We omit linear evolution methods for `Office`, for previously stated reasons. Also, for noisy scenes, it is wise not to utilize a diffusivity that provides much backward diffusion, like Weickert diffusivity. Instead, we choose to employ Perona/Malik diffusivity for these tests. We show the results in Figures 4.13, 4.14, and compare to low order methods in Figure 4.16.

For all experiments, we also show the evolution plots which contain several values of the image throughout the evolution: minimum, maximum, mean and variance have been measured in regular intervals and are displayed in red, green, blue, and pink lines respectively. The variance is plotted against a second y-axis with a logarithmic scale. The plots can be found in Figures 4.5, 4.8, 4.12 for `R&E`, and 4.15 for `Office`, respectively, for the above mentioned experiments.

We show if $\mathcal{L}_2$-stability was adhered to as long as the evolution ran, and if not, at what iteration it was violated. Note that in some cases we indicate that it was not violated but nevertheless mention an iteration number; this is to say that the software reported increasing variance but we still believe the process to be stable (see the evolution plots) and that some quantization error or precision problem was encountered.

Suffice to say, some quantization errors are already present in the original image: if we look at e.g. Figure 4.3(c), we see the (scaled) Frobenius norm of the Hessian of the `R&E` image, which is close to zero, but not zero, as it should be in theory, except at the discontinuities. For that reason, these illustrations are sometimes more correctly called "approximations", as are in fact all discretizations (of the image, the derivatives, etc.). Another artifact of discretization and quantization can e.g. be seen at the right extremity of the ellipse: there seems to be a completely vertical line of about ten pixels; in reality, this would only be a spot of low but not zero curvature.

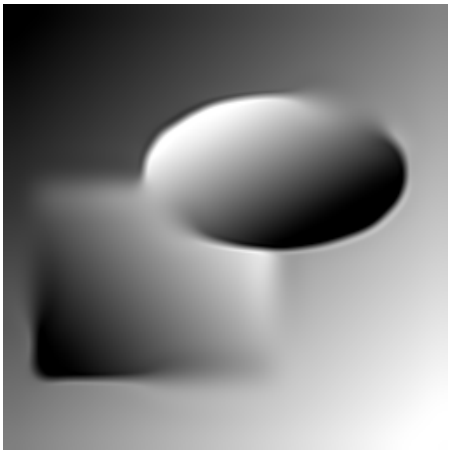


FIGURE 4.2: Result of applying FM2-ANISO-NL to `R&E` ($\lambda = 1, \sigma = 3, t = 10000$).

4.3 Evaluation

Test images featuring linear gradients and clear edges are a perfect testing ground for higher order methods. When we look at edge preservation in such a "higher order" scene like `R&E`, we conclude that for isotropic higher order processes the same overall behavior than for lower order ones applies: homogeneous areas are blurred, and when we increase the contrast parameter (if the diffusivity function supports it), important structures start to get blurred too. It is important to understand that for lower and higher order methods "homogeneous area" means different things. For a truly higher order method, the area is still homogeneous if it contains a linear gradient, whereas for a lower order method it certainly is not.

Higher order methods do not produce segmentations but rather create color gradients, if the appropriate higher order argument (namely the Frobenius norm of the Hessian) is used for diffusivity steering (see Figure 4.7, FM1-ISO-NL vs. FM2-ISO-NL). This applies to both isotropic Tumblin/Turk and Frobenius model.

As far as the anisotropic methods are concerned, we conclude that the Tumblin/Turk evolution equation, even though its anisotropic variant preserves discontinuities, produces too apparent over- and undershoots and speckle artifacts. Also, the linear gradients from the `R&E` test image have not been correctly preserved, as it should be for a true higher order method. The linear variants of this evolution equation even seem to be unstable, for unclear reasons. The questionable cases are Wei-ANISO-L and TT-ANISO-L, i.e. all linear anisotropic filters derived from the Tumblin/Turk evolution equation. The anisotropic Frobenius model works very well as expected, although discontinuities that are not parallel to the coordinate axes seem to suffer. The reason for this could be that the discretizations of the mixed derivatives are still not good enough, even though we have already used more complicated variants than usual.

Looking e.g. at Figure 4.2, we see that the points of interest are in the locations where the gray values of the outer gradient (background) and inner gradients

(square or ellipse, respectively) match. Also, at careful inspection of e.g. the upper left boundary of the ellipse in Figure 4.2, we see a halo, which is foreseeable for higher order methods: if light and dark areas collide, the darker area may become darker at the boundary, and the light area lighter. This effect however remains bounded for $\mathcal{L}_2$-stable methods. One could consider this as some form of contrast enhancement.

In Figure 4.9 we see, beside the obvious effect of staircasing created by lower order methods, that the low order variant essentially has to choose a certain direction (inward or outward) of a structure (segmentation-like behavior), the higher order variant treats the edges where inner and outer gray value become similar, in a way that one could call "bleeding edge": a gradient appears in the image instead of having to choose a direction for a segmentation.

Faced with the task noise removal (as in the experiments with the `Office` image starting at Figure 4.13 on page 71), anisotropic models with the correct parameters work very well, higher order methods make no exception and even keep image features more intact than lower order methods (for the same parameters at the same stopping time). Isotropic methods still keep noise at edges and anisotropic methods have more trouble cleaning out homogeneous areas — these deficiencies are however intrinsic to the respective methods.

We notice that the anisotropic filters produce a noise-free image much faster (about 50 times) than the isotropic variant of the filter using the same parameters. However, the quality of the result is inferior to the isotropic filters in this case, probably because the noise disturbs structure detection too much. Additionally, we see that the anisotropic variants of the Tumblin/Turk-based evolution equations do not provide nice denoising: speckles (unesthetic over- and undershoots) appear.

Furthermore, we see in Figure 4.16 that the anisotropic methods remove noise also close to edges very well, however they do not restore regions that were homogeneous in the original as well; here, the isotropic FM variants look better, and their result is a much sharper image than that of the lower order isotropic filter (ISO). Note that corners are less rounded at the same stopping time using the higher order method.

Thus, we conclude that for a noisy real-world scene, the Frobenius model works very well and, as for lower order methods, the isotropic variant keeps noise at edges, and the anisotropic variant rounds corners.

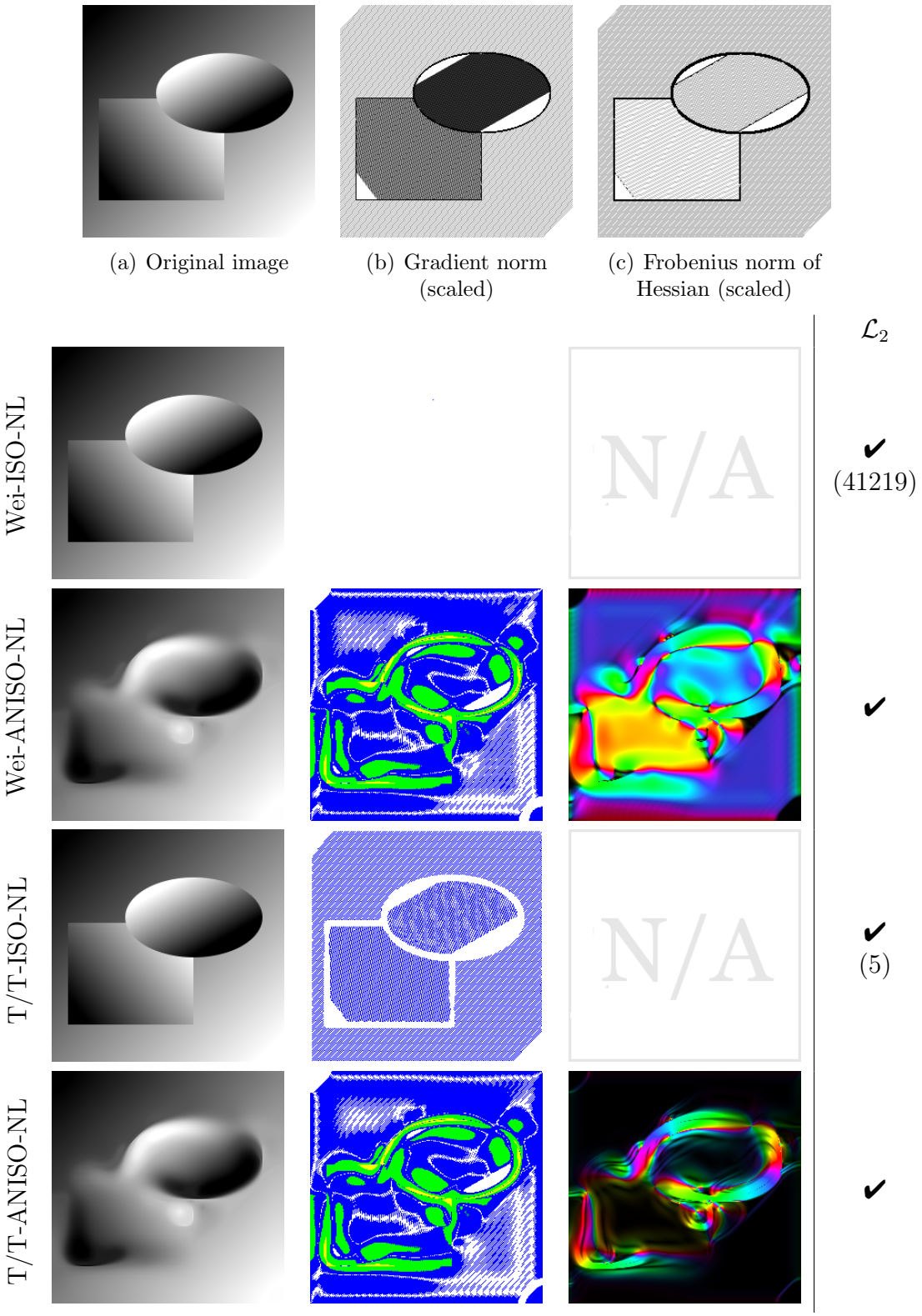


FIGURE 4.3: Nonlinear fourth order methods applied to R&E, Parameter Set I (1/2) ($\lambda = 0.01, \sigma = 1, t = 10000, \tau = 0.025$). ▶ **top**: original image and norms ALL BUT FIRST ROW: ▶ **left image**: resulting image ▶ **center image**: absolute difference to original ▶ **right image**: detected directions weighted by diffusivity (where applicable) ▶ **rightmost column**: adherence to $\mathcal{L}_2$-stability (✔=yes, ✖=no)

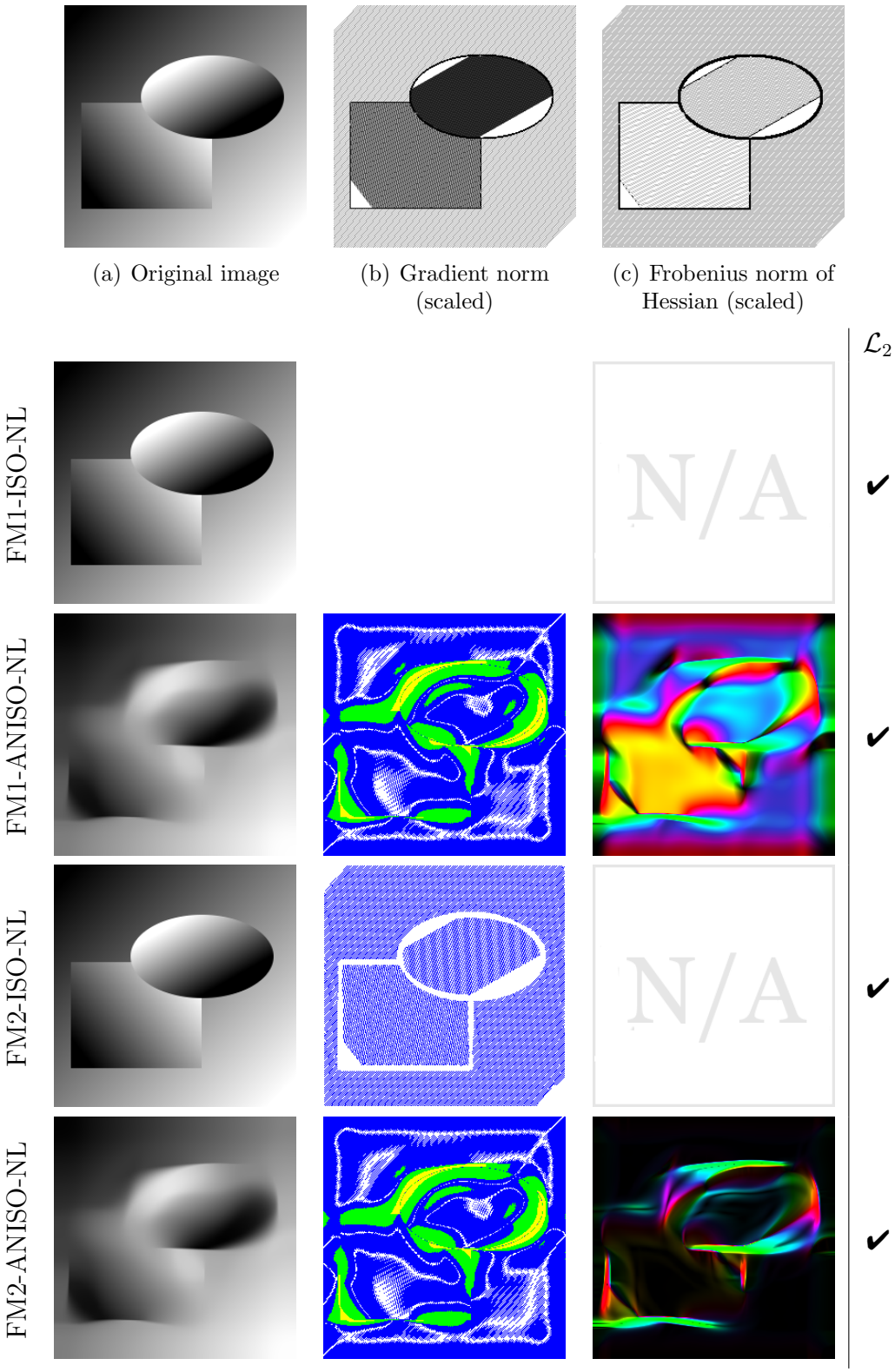
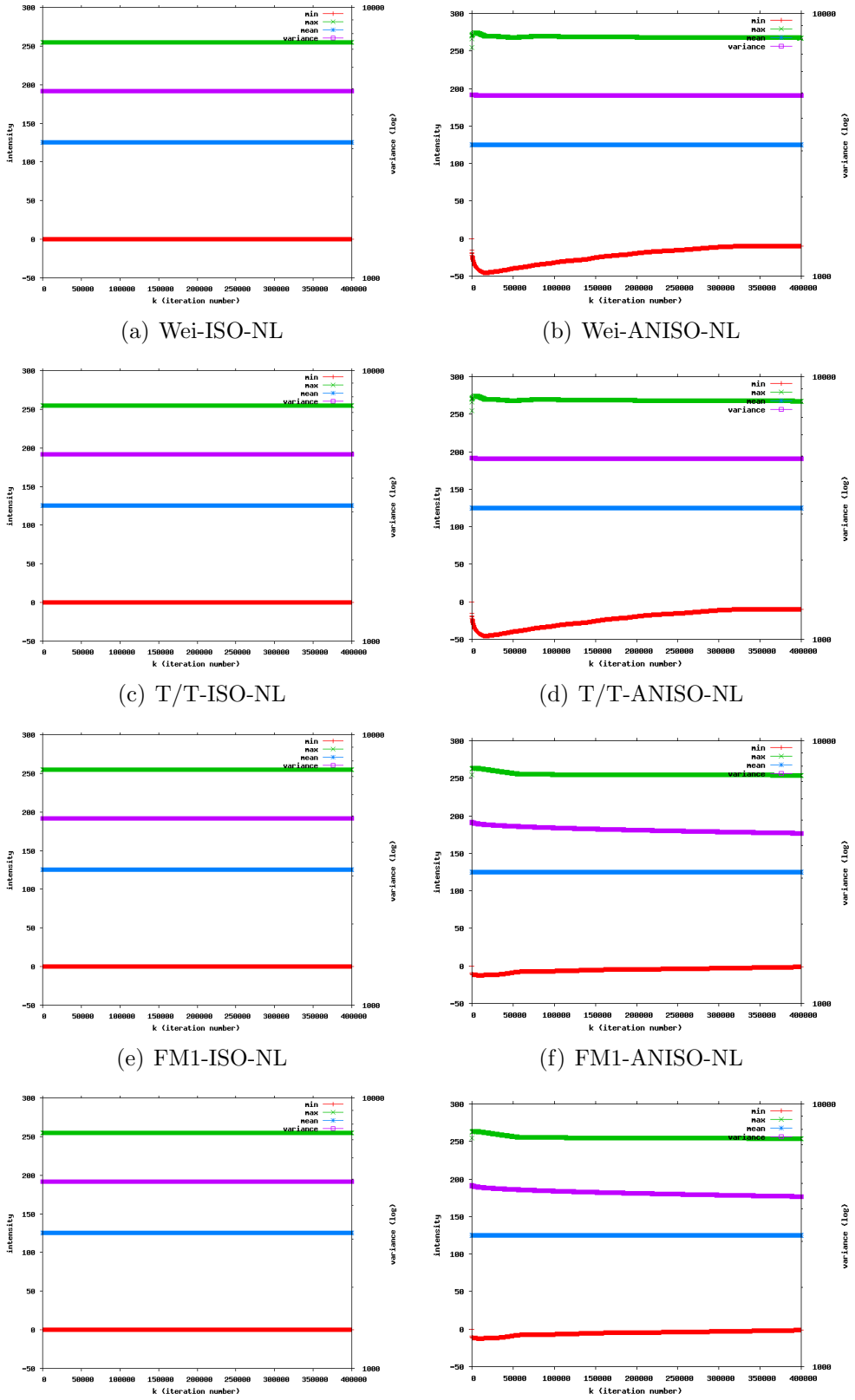


FIGURE 4.4: Nonlinear fourth order methods applied to R&E, Parameter Set I (2/2) ($\lambda = 0.01, \sigma = 1, t = 10000, \tau = 0.025$). ▶ **top**: original image and norms ALL BUT FIRST ROW: ▶ **left image**: resulting image ▶ **center image**: absolute difference to original ▶ **right image**: detected directions weighted by diffusivity (where applicable) ▶ **rightmost column**: adherence to $\mathcal{L}_2$-stability (✔=yes, ✖=no)



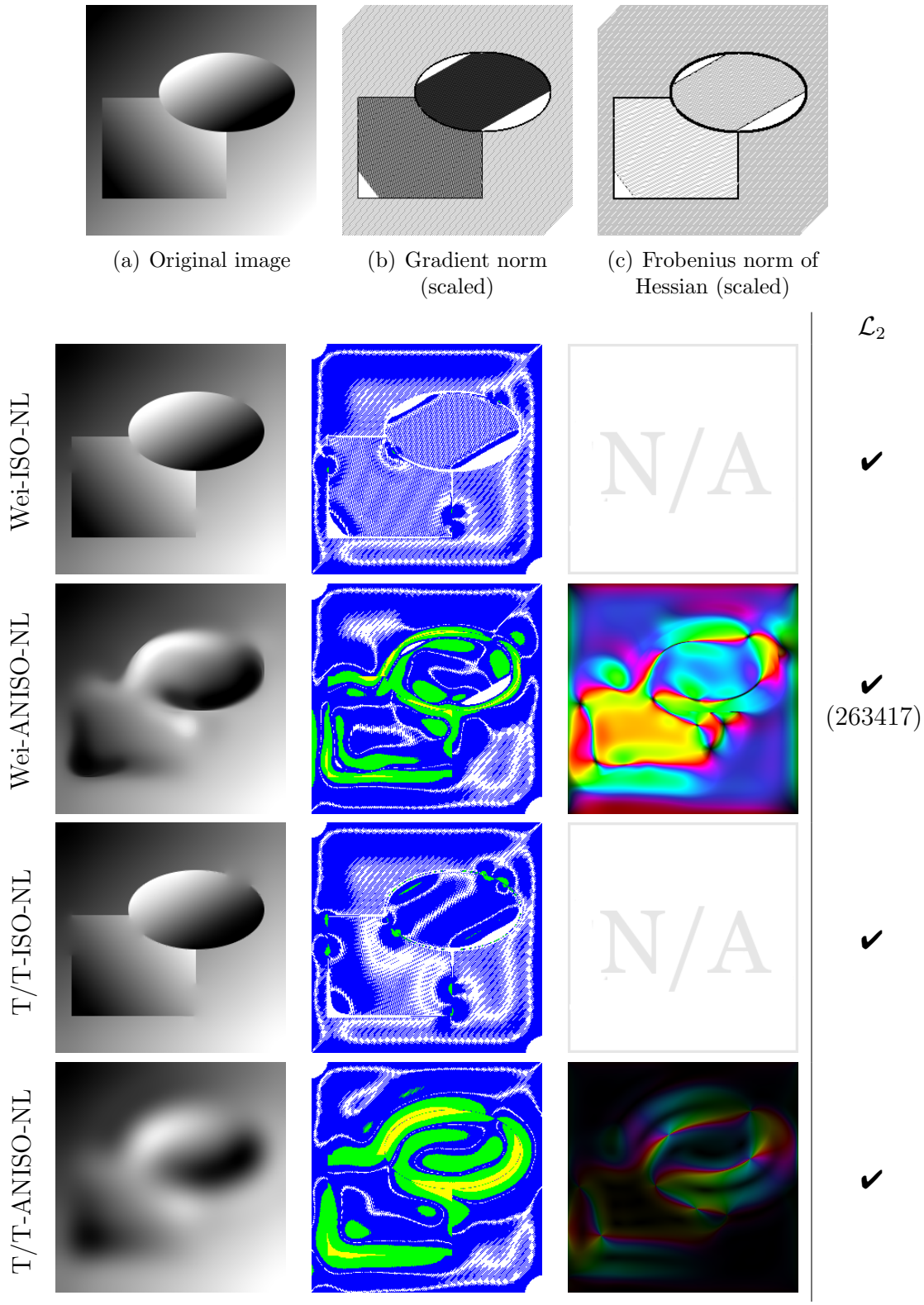
(a) Wei-ISO-NL (b) Wei-ANISO-NL

(c) T/T-ISO-NL (d) T/T-ANISO-NL

(e) FM1-ISO-NL (f) FM1-ANISO-NL

(g) FM2-ISO-NL (h) FM2-ANISO-NL

FIGURE 4.5: Evolution of nonlinear fourth order methods applied to R&E, Parameter Set I.



(a) Original image (b) Gradient norm (scaled) (c) Frobenius norm of Hessian (scaled)

FIGURE 4.6: Nonlinear fourth order methods applied to R&E, Parameter Set II (1/2) ($\lambda = 1, \sigma = 1, t = 10000, \tau = 0.025$). ▶ **top**: original image and norms ALL BUT FIRST ROW: ▶ **left image**: resulting image ▶ **center image**: absolute difference to original ▶ **right image**: detected directions weighted by diffusivity (where applicable) ▶ **rightmost column**: adherence to $\mathcal{L}_2$-stability (✔=yes, ✖=no)

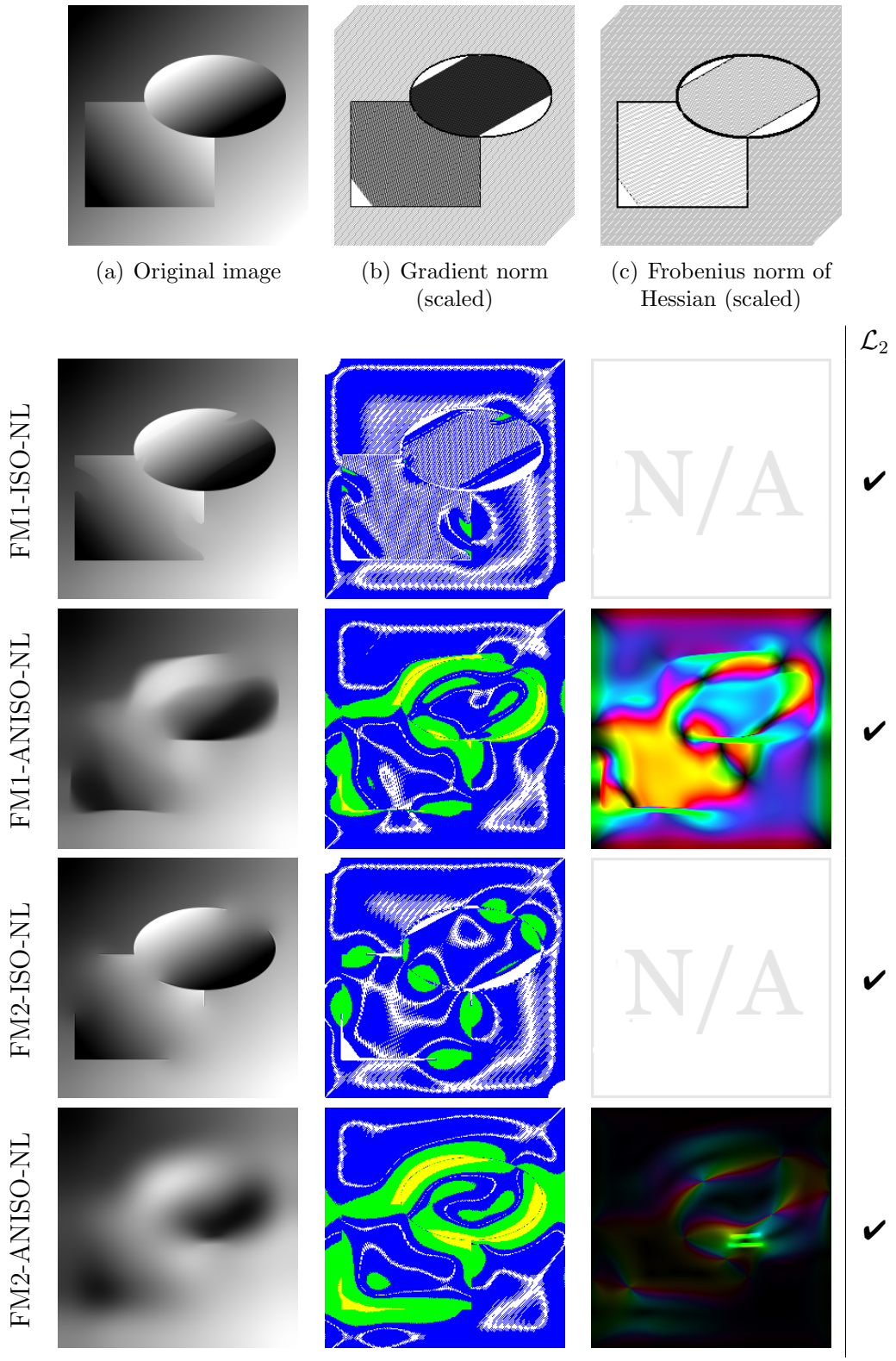
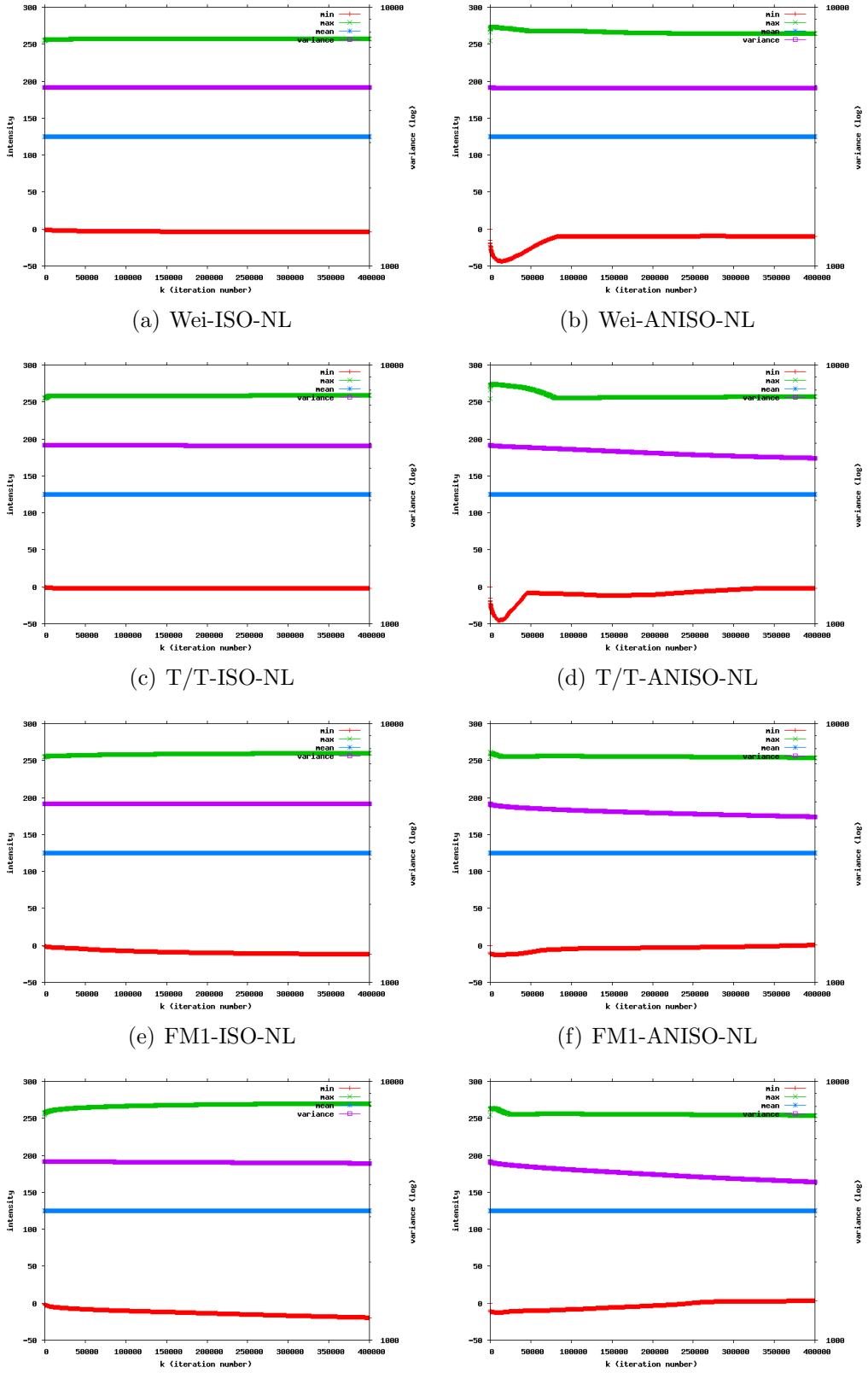


FIGURE 4.7: Nonlinear fourth order methods applied to R&E, Parameter Set II (2/2) ($\lambda = 1, \sigma = 1, t = 10000, \tau = 0.025$). ▶ **top**: original image and norms ALL BUT FIRST ROW: ▶ **left image**: resulting image ▶ **center image**: absolute difference to original ▶ **right image**: detected directions weighted by diffusivity (where applicable) ▶ **rightmost column**: adherence to $\mathcal{L}_2$-stability (✔=yes, ✖=no)



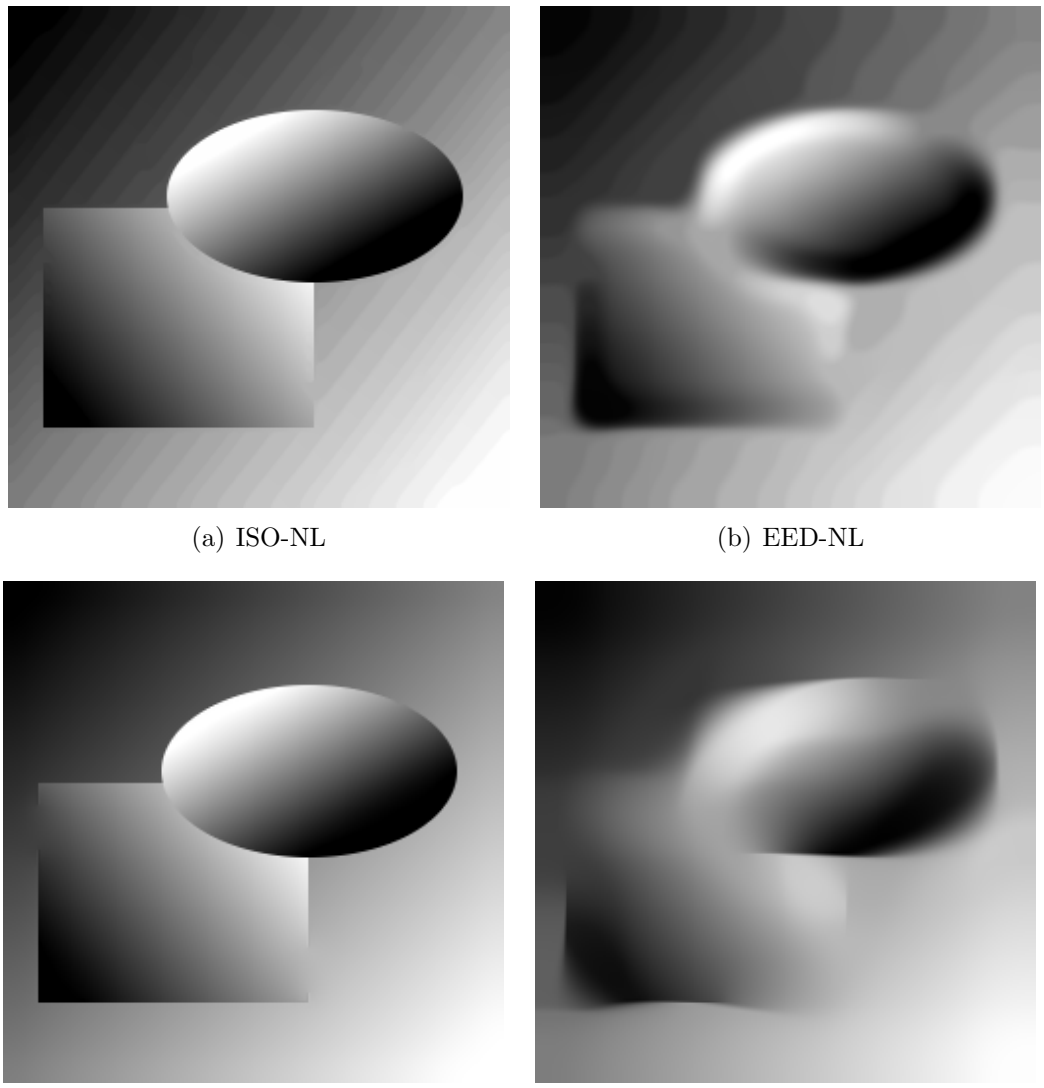
(a) Wei-ISO-NL (b) Wei-ANISO-NL

(c) T/T-ISO-NL (d) T/T-ANISO-NL

(e) FM1-ISO-NL (f) FM1-ANISO-NL

(g) FM2-ISO-NL (h) FM2-ANISO-NL

FIGURE 4.8: Evolution of nonlinear fourth order methods applied to R&E, Parameter Set II.



(a) ISO-NL (b) EED-NL

(c) FM2-ISO-NL (d) FM2-ANISO-NL

FIGURE 4.9: Nonlinear low and high order methods applied to `R&E` ($\lambda = 0.5$, $\sigma = 1$, $t = 1250$).

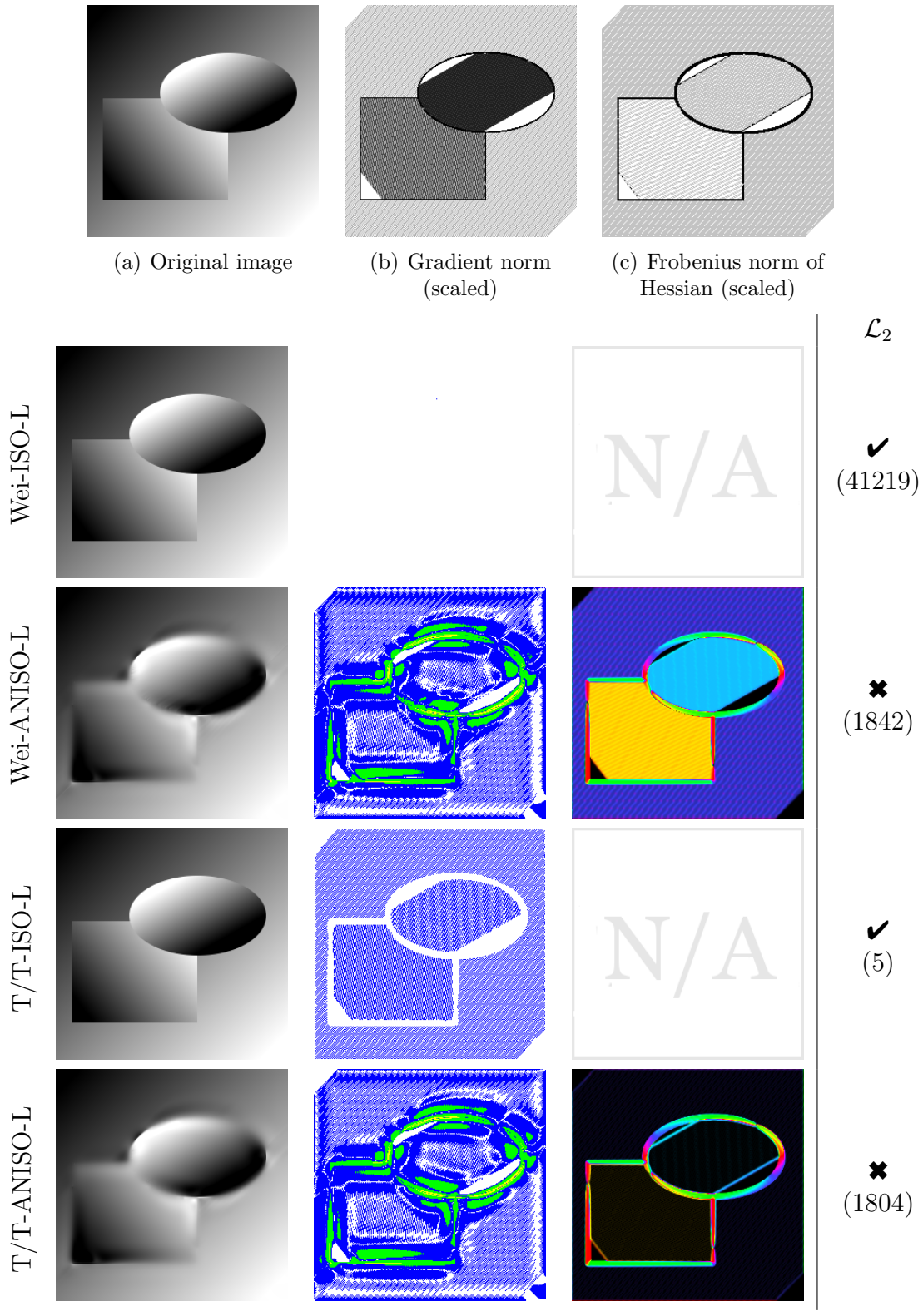


FIGURE 4.10: Linear fourth order methods applied to R&E (1/2) ($\lambda = 0.01, \sigma = 1, t = 1250, \tau = 0.025$). ▶ **top**: original image and norms ALL BUT FIRST ROW: ▶ **left image**: resulting image ▶ **center image**: absolute difference to original ▶ **right image**: detected directions weighted by diffusivity (where applicable) ▶ **rightmost column**: adherence to $\mathcal{L}_2$-stability (✔=yes, ✖=no)

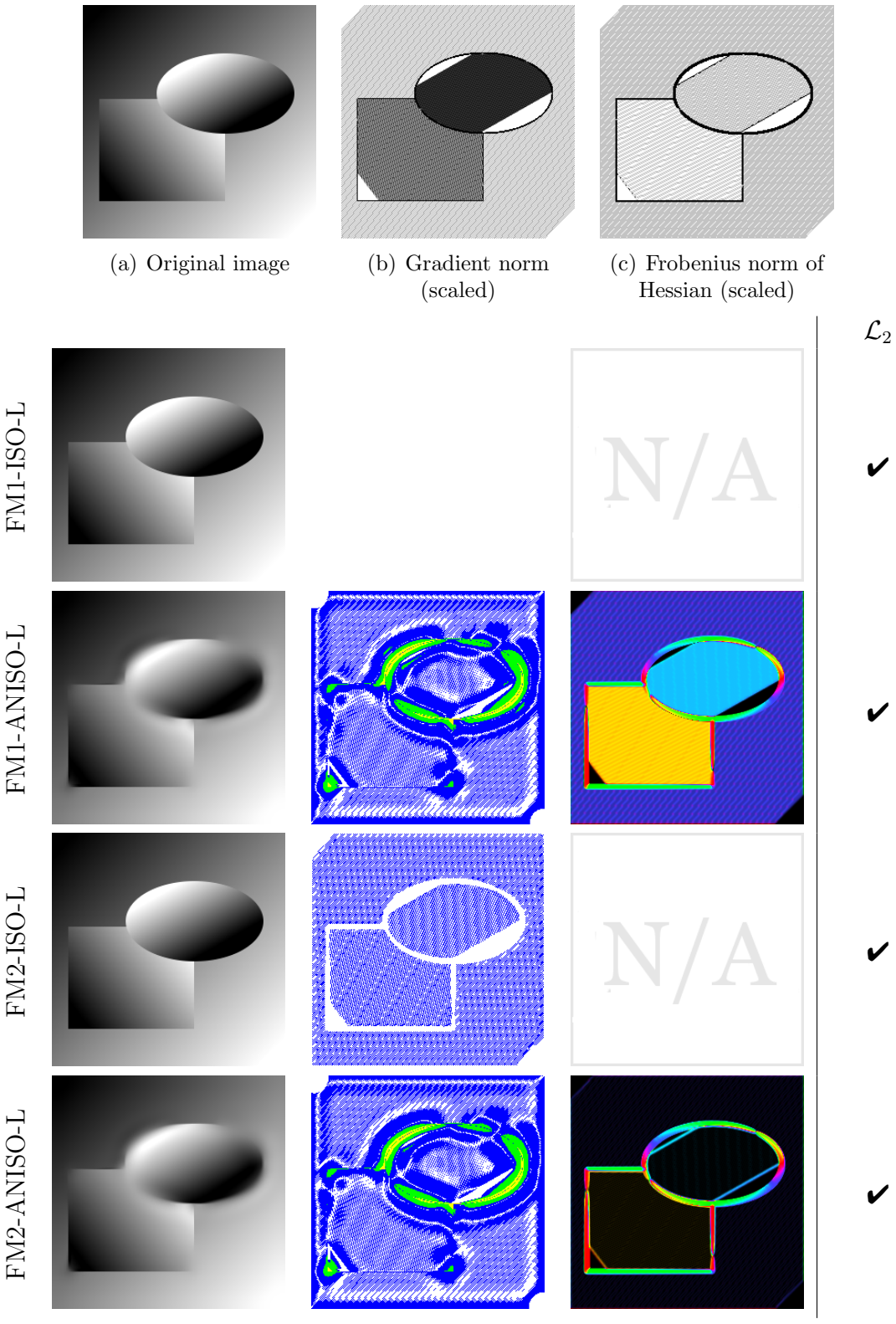
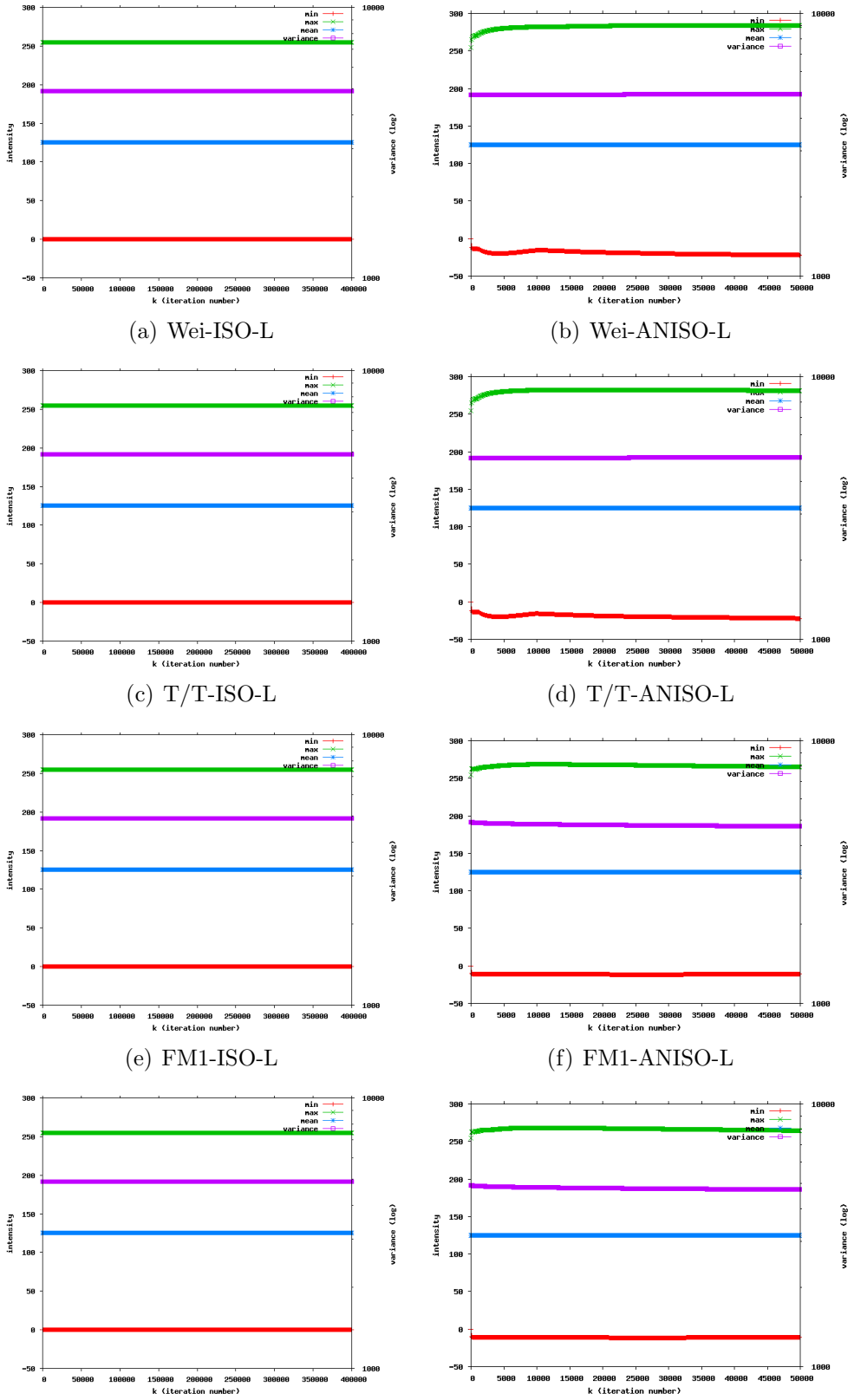


FIGURE 4.11: Linear fourth order methods applied to R&E (2/2) ($\lambda = 0.01, \sigma = 1, t = 1250, \tau = 0.025$). ▶ **top**: original image and norms ALL BUT FIRST ROW: ▶ **left image**: resulting image ▶ **center image**: absolute difference to original ▶ **right image**: detected directions weighted by diffusivity (where applicable) ▶ **rightmost column**: adherence to $\mathcal{L}_2$-stability (✔=yes, ✖=no)



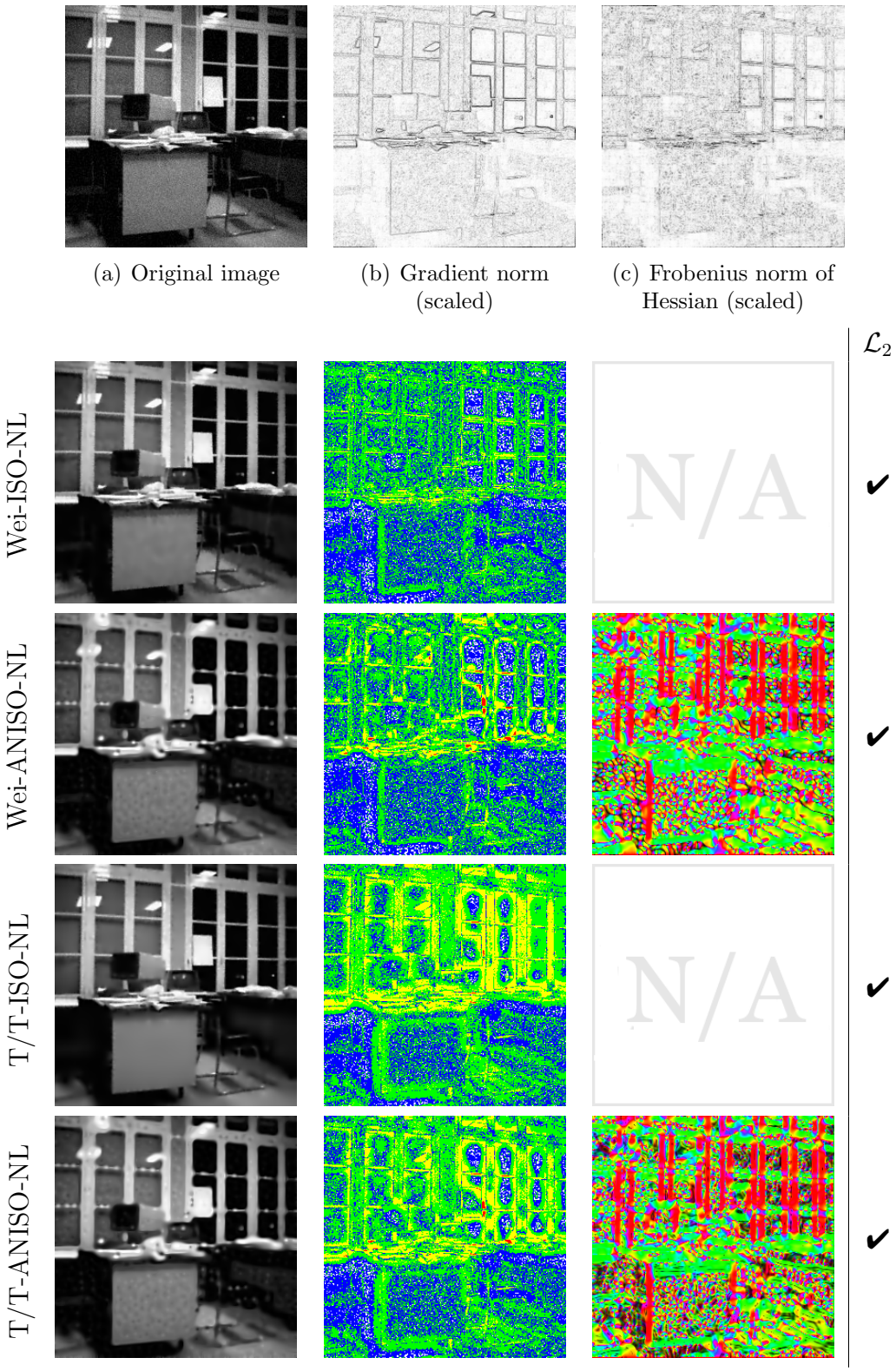
(a) Wei-ISO-L (b) Wei-ANISO-L

(c) T/T-ISO-L (d) T/T-ANISO-L

(e) FM1-ISO-L (f) FM1-ANISO-L

(g) FM2-ISO-L (h) FM2-ANISO-L

FIGURE 4.12: Evolution of linear fourth order methods applied to R&E.



(a) Original image (b) Gradient norm (scaled) (c) Frobenius norm of Hessian (scaled)

FIGURE 4.13: Nonlinear fourth order methods applied to `Office` (1/2) (Perona/Malik: $\lambda = 0.1$, $\sigma = 1$; ISO: $t = 1250$, ANISO: $t = 25$). ALL BUT FIRST ROW: ▶ **left image**: resulting image ▶ **center image**: absolute difference to original ▶ **right image**: detected directions weighted by diffusivity (where applicable) ▶ **rightmost column**: adherence to $\mathcal{L}_2$-stability (✔=yes, ✖=no)

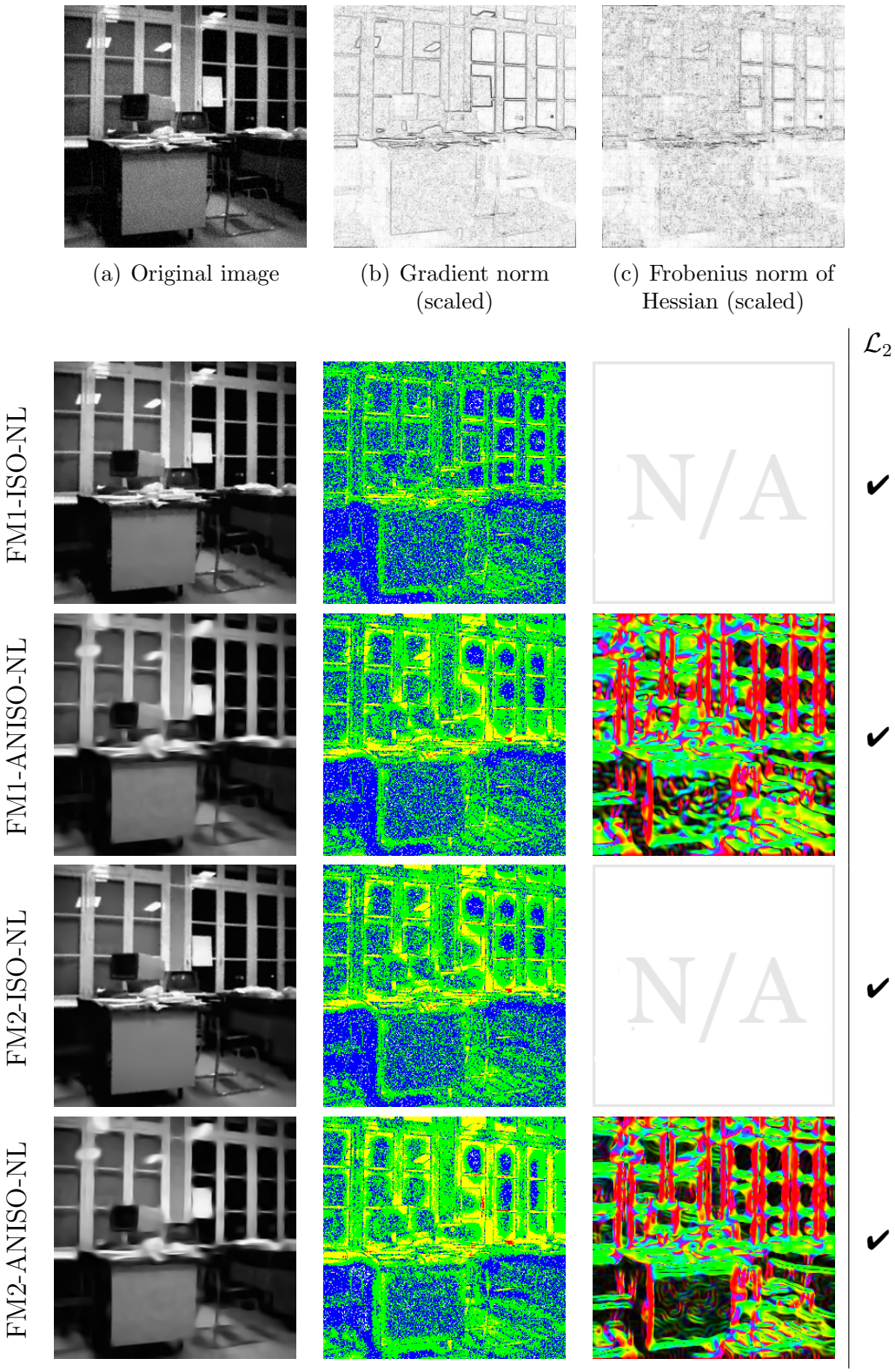
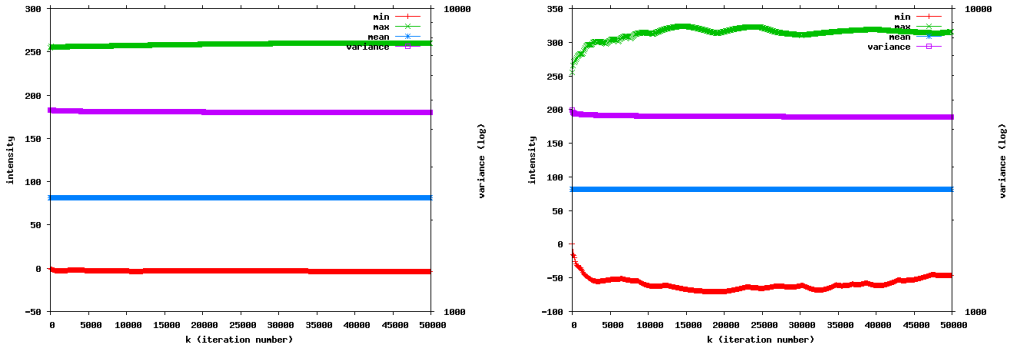
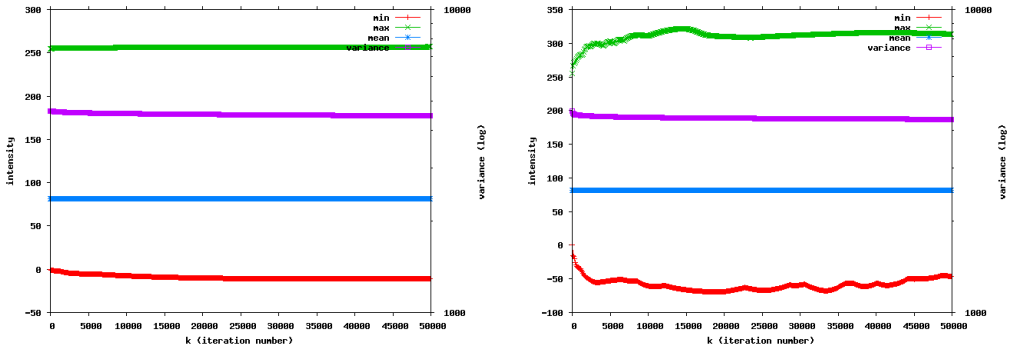


FIGURE 4.14: Nonlinear fourth order methods applied to `Office` (2/2) (Perona/Malik: $\lambda = 0.1$, $\sigma = 1$; ISO: $t = 1250$, ANISO: $t = 25$). ALL BUT FIRST ROW: ▶ **left image**: resulting image ▶ **center image**: absolute difference to original ▶ **right image**: detected directions weighted by diffusivity (where applicable) ▶ **rightmost column**: adherence to $\mathcal{L}_2$-stability (✔=yes, ✖=no)



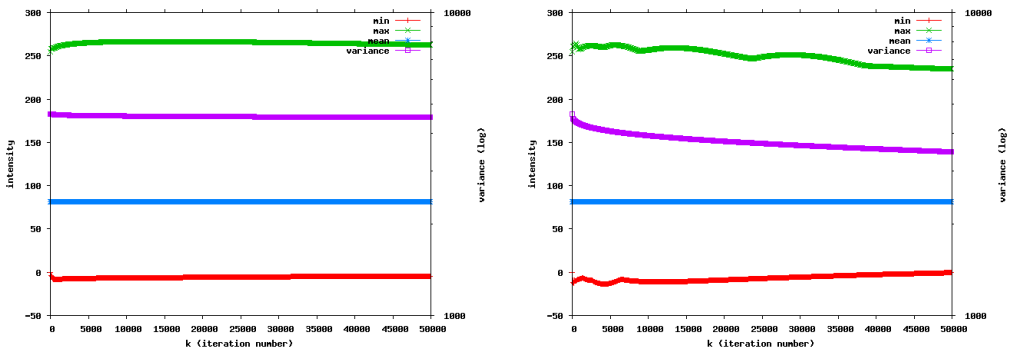
(a) Wei-ISO-NL

(b) Wei-ANISO-NL



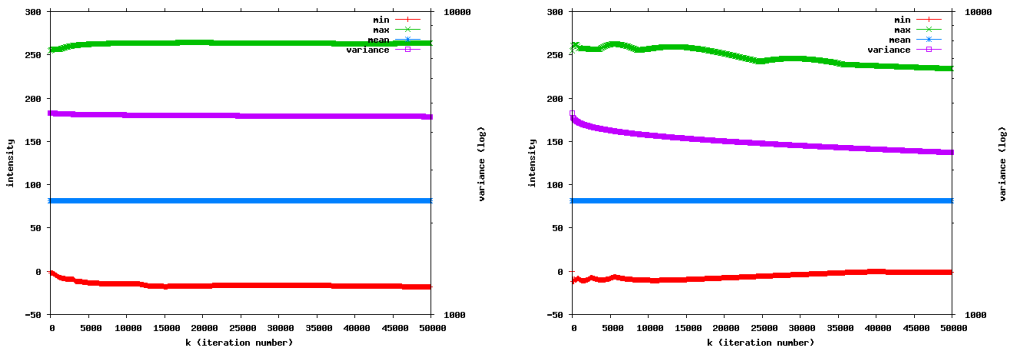
(c) T/T-ISO-NL

(d) T/T-ANISO-NL



(e) FM1-ISO-NL

(f) FM1-ANISO-NL



(g) FM2-ISO-NL

(h) FM2-ANISO-NL

FIGURE 4.15: Evolution of nonlinear fourth order methods applied to `Office`.

(a) ISO-NL (b) EED-NL

(c) FM2-ISO-NL (d) FM2-ANISO-NL

FIGURE 4.16: Low and high order methods applied to `Office` (Perona/Malik: $\lambda = 0.1$, $\sigma = 1$; ISO: $t = 1250$, ANISO: $t = 25$).

Chapter 5

Interpolation, Inpainting & Compression

We encounter interpolation problems in everyday life, like increasing the resolution of images (i.e. rescaling, zooming), such that no unesthetic (i.e. blocky) artifacts appear e.g. using high magnifications of digital cameras (so-called "digital zoom").

Even simple image processing tasks involve interpolation in less apparent scenarios like the rotation of digital images by nontrivial angles: the pixel grid (of the screen or printer) is fixed and cannot be rotated, therefore, after rotation, one point may have been attributed several color values. In this case, interpolation is used to create *one* suitable value from the different values proposed for that pixel.

5.1 PDE-Based Formulation

One can describe the interpolation/inpainting problem as an evolution

$$\partial_t u = c \cdot (u - f)^2 - (1 - c) \cdot Lu \tag{5.1}$$

where, as usual, f denotes the initial image, u the evolved/restored image and c a binary confidence function, i.e. for a 2-D dataset

$$c(x, y) := \begin{cases} 1 & \text{if } f(x, y) \text{ is to be trusted,} \\ 0 & \text{if } f(x, y) \text{ is to be filled in.} \end{cases} \tag{5.2}$$

The smoothness term Lu comprises an elliptic differential operator L, which stands for any interpolation method we would like (see next section).

The solution (i.e. steady state) of this evolution equation gives the desired interpolating function, which is exactly the values $f(x, y)$ where $c(x, y) = 1$ and should be close to the (unknown) values $f(x, y)$ where $c(x, y) = 0$. This means that $u(x, y, t) = f(x, y)$, for all t where $c(x, y) = 1$. Note that the trusted data points need not be on a regular grid and can very well be scattered.

5.2 Interpolation Methods

Interpolation[1] denotes any means of obtaining data at an unknown data point from known data points. Therefore, it is a special instance of curve fitting: the interpolated function must go exactly through trusted data points known a priori.

The simplest form of interpolation (which sometimes is not even called as such) is nearest-neighbor interpolation, where an unknown point simply inherits the values from the point closest to it.

The next simplest form of interpolation, linear interpolation, is to take the arithmetic mean of two points to find the data at the midway point: if we have a function $f : \mathbb{R} \to \mathbb{R}$, where the known values are e.g. $f(1) = 1$ and $f(2) = 3$, then linear interpolation would give $f(1.5) = 2$ [SB02]. In 1-D, linear interpolation is the same as homogeneous diffusion in 1-D, and in higher dimensions corresponds to interpolation using radial basis functions (functions, whose values solely depend on the distance to some center), like a Gaussian [Buh03].

Reconstructing data outside the first and last known elements x_0 and x_1 is called extrapolation, which we are not going to address here. Suffice to say, to reproduce the missing or unknown data, one extends the "behavior" of the closest data (this time, at *one end only*). Obviously, if there is knowledge about the process that is the source of the data to be reproduced, both inter- and extrapolation will recreate data close to the original with much higher certainty, and one need not only use the "last known" data.

In linear interpolation the interpolant obviously is a linear function. This can be generalized by replacing the interpolant with a function of higher degree, which yields polynomial interpolation.

The next better thing is spline interpolation, where data is represented by piecewise polynomials, i.e. a polynomial is inserted into each of the intervals of unknown data and the different pieces are combined such that they fit smoothly together [Boo78].

Finally, as far as general smoothing methods are concerned one can also use any method imaginable, like mean curvature motion (MCM) [AGLM93]

$$\partial_t u = \partial_{\xi\xi} u, \tag{5.3a}$$

absolute monotone Lipschitz extension (AMLE) [CMS98]

$$\partial_t u = \partial_{\eta\eta} u, \tag{5.3b}$$

or any other method that we have seen so far in this work, like EED. This is where we are going to proceed after a glimpse at how inpainting and compression are related.

[1] *to interpolate* (implied in interpolation): from Latin *interpolatus*, pp. of *interpolare* "alter, freshen up, falsify," from inter- "up" and *polare*, related to polire "to smoothe, polish."

5.3 Inpainting and Compression

The need for image compression is obvious: one wants to eliminate redundant data in order to be able to store more data using the same space or for example transmit the data in less time. All of this comes at a price: computation time. With nowadays' computers, computation time mostly is not an issue, but it needs to be said that this is a classical case where we trade space against time.[2]

Image compression can be considered as an extreme application of inpainting, which makes this kind of compression a lossy compression, as interpolation is involved. Usually, inpainting is used to interpolate small areas of an image that have been lost. Image compression tries to minimize the number of pixels that are *kept* from the original image, in order to reproduce a high-quality version thereof.

This introduces the question which pixels are relevant. Obviously, this depends on the decompression step, which uses the selected pixels and tries to restore the original image. Thus, compression and decompression had best be closely related to each other.

In [GWW+05], Galić et al. have proposed image compression using anisotropic diffusion. We would like to find out if the inclusion of a higher order diffusion method, namely our Frobenius model (more precisely, FM2-ANISO-NL), can reduce the error rate even further. This could be quantified in two ways: a better decompression result using the same compression rate as they did, or a similar decompression result at a higher compression rate.

They have made experiments using many methods, starting with plain and evident ones, then including many optimizations. While we cannot use or compare to their best method (for reasons stated later), let us now give a quick overview of the main points of procedure:

Sparsification of the original image using Binary Tree Triangular Coding (BTTC) [DNV97], which decomposes the image into smaller and smaller isosceles right triangles (using some error measure to decide whether to split or not, see below) and stores seed points at the remaining vertices.

Error Measure for Compression is linear interpolation or EED. EED has the drawback that it influences the entire image, not only the current triangle, and therefore, for the best results, one has to recompute at each triangle splitting. Since EED is already an expensive computation, recomputing at each step makes this special compression task tedious.

Huffmann Coding [Huf52] of the resulting binary tree structure is performed, i.e. the values that appear most often are encoded using the smallest representation, to save space.

Requantization is denoted by the prefix Q64+, which means pre-requantization (input image) to 64 gray values. Some of the more advanced methods use

[2] There still is no free lunch.

+Q32 (i.e. post-requantization to 32 gray values), however, as we said before, we stick to one of the simpler methods to base our comparisons on.

In the following, we will use denote the relative compression rate using bits per pixel (bpp). This means that for an uncompressed image with 256 possible different values per pixel one uses full 8 bpp. In that case, e.g. 0.8 bpp denotes an absolute compression rate of 1/10. The conversion is simple, if q denotes the possible quantizations per pixel (in the case of a multi-channel image, all channels summed up), then the absolute compression rate c_A for a relative compression rate c_R is

$$c_A = \frac{c_R}{\log_2(q)} \,. \tag{5.4}$$

In our previous example, that would be $\frac{0.8}{8} = 0.1 = \frac{1}{10}$.

5.4 Inpainting using Frobenius Model

For the actual implementation, we have one problem, namely the explicit scheme, which has severe restrictions on the time step size. For EED it is already not easy to obtain a positive semidefinite system matrix which allows the usage of an SOR solver and therefore eliminates the time step size problem. (It however also limits maximum anisotropy.) For higher order methods, it is simply not possible, as they no longer adhere to a maximum-minimum principle.

This has the second consequence that we of course cannot[3] use the same method in the coding and decoding step. Therefore, we show results for seed points generated using linear interpolation as error measure in the compression step, as well as for low order EED.

The first method is called Q64+BTTC(L) in [GWW+05]. We compare that method using classic EED as interpolant, and after that, we have used the Frobenius model (FM2-ANISO-NL, to be precise). We then repeat the same experiments using Q64+BTTC(EED) in the compression step. We denote the complete method by Q64+BTTC(*)-FM. We have used explicit schemes for both methods.

We show inpainting of `Trui` using Q64+BTTC(L)-* inpainting using different compression rates in in Figures 5.2 and 5.3, then using Q64+BTTC(EED)-* in 5.4. We repeat the experiment using Q64+BTTC(EED)-* for `Camera` in 5.6.

Table 5.1 summarizes the average absolute error values side by side. Note that for EED in the compression step we have used precomputed masks provided by the authors of [GWW+05] and that there is no such mask for `Trui` with 0.8 bpp compression rate. Thus, there still is a mismatch between compression and decompression, but we believe it alleviates that problem at least a little bit.

"Pure" Q64+BTTC(EED)-EED was not evaluated in [GWW+05] – we have computed it as a simple reference method. We show the evolution of the error during the interpolation in Figures 5.5 and 5.7 for the several experiments. For

[3] We could, but it would take months, if not years, to complete one single experiment.

(a) JPEG (b) JPEG 2000

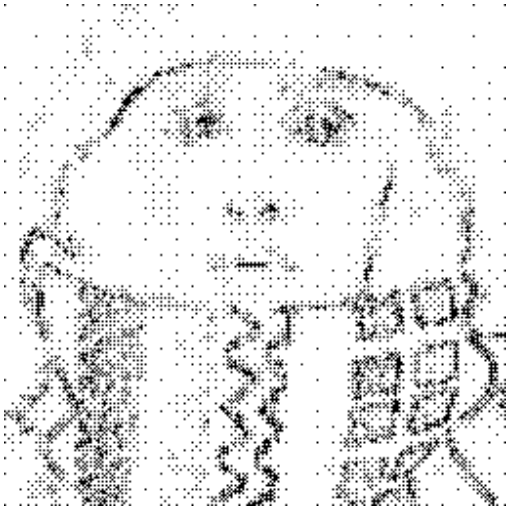
FIGURE 5.1: Compression of `Trui` using JPEG algorithms (0.2 bpp) (Source:[GWW+05]).

the sake of completeness we also show the principal diffusion directions in the final iteration — note that the directions are the same, but whether or not diffusion happens at all varies (because EED uses a different argument to diffusivity than FM). Here we see that for `Camera`, the sky is not smoothed, as it is smooth enough already (for the higher order method); in extreme cases lower order methods would produce staircasing here).

For all interpolation tasks, lots of forward diffusion (smoothing) is necessary, therefore we use Charbonnier diffusivity for all experiments. Parameters are $\lambda = 0.1$ and $\sigma = 0.8$, except otherwise noted. We have stopped both interpolations when the change in error between two subsequent measurements (one measurement each 500 iterations) got negligible or the error even started getting worse again. The results are listed in Table 5.1.

We have tried using mixed higher order methods like Tumblin/Turk or Wei (see Section 3.2) for the inpainting task — it however yielded no usable results.

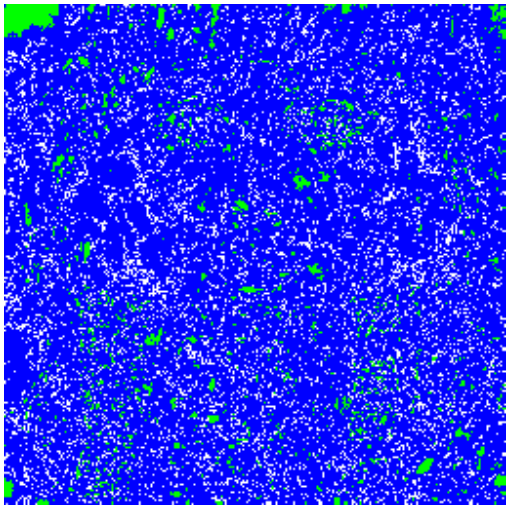
(a) `Trui`



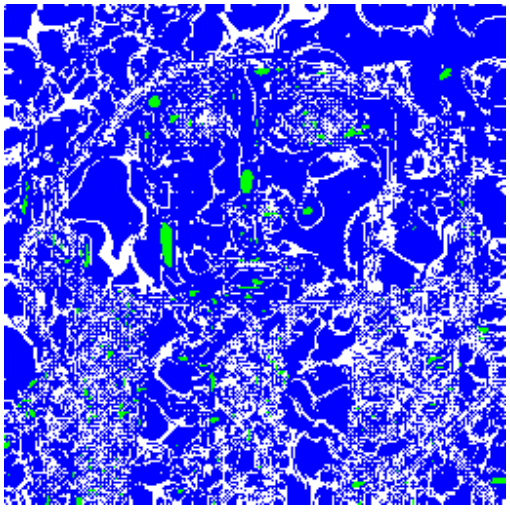
(b) Interpolation mask

(c) Q64+BTTC(L)-EED

(d) Q64+BTTC(L)-FM



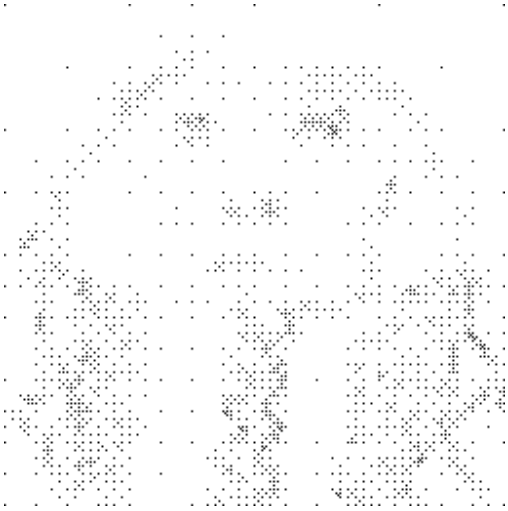
(e) Diff. Q64+BTTC(L)-FM vs. original



(f) Difference between the two interpolations

FIGURE 5.2: Interpolation of `Trui`: Q64+BTTC(L) compression at 0.8 bpp.

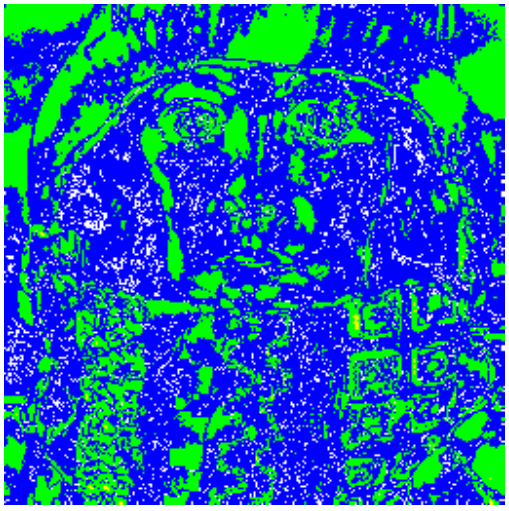
(a) Original image



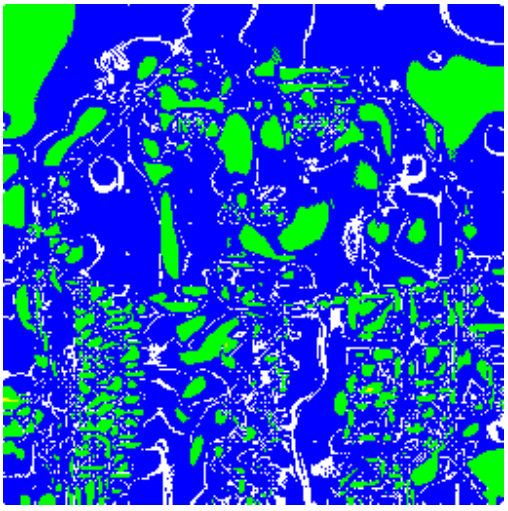
(b) Interpolation mask

(c) Q64+BTTC(L)-EED

(d) Q64+BTTC(L)-FM



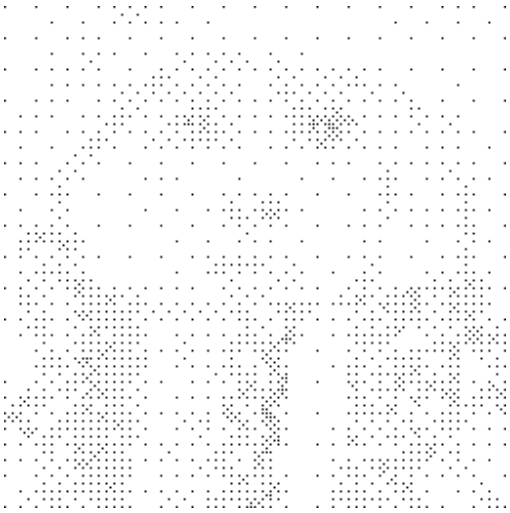
(e) Diff. Q64+BTTC(L)-FM vs. original



(f) Difference between the two interpolations

FIGURE 5.3: Interpolation of `Trui`: Q64+BTTC(L) compression at 0.2 bpp.

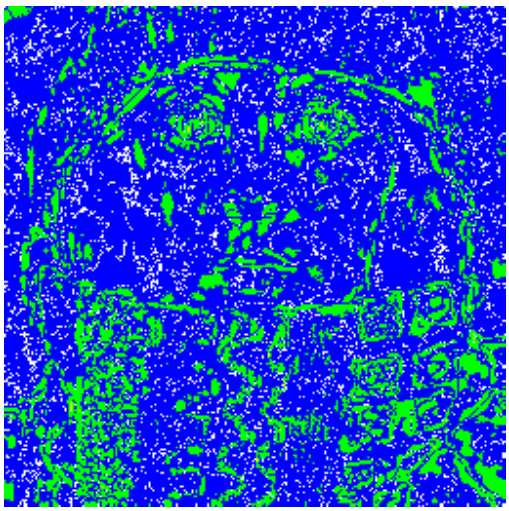
(a) Original image



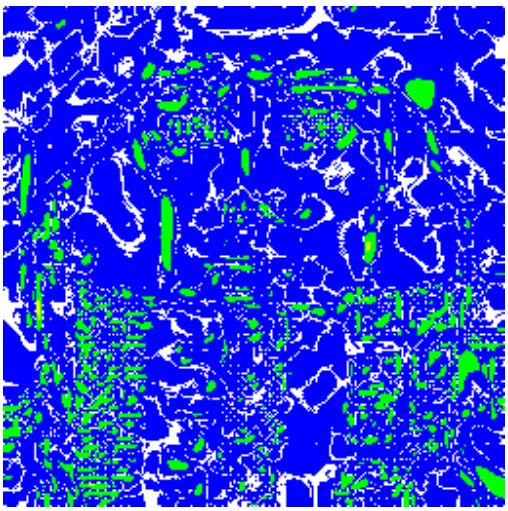
(b) Interpolation mask

(c) Q64+BTTC(EED)-EED

(d) Q64+BTTC(EED)-FM

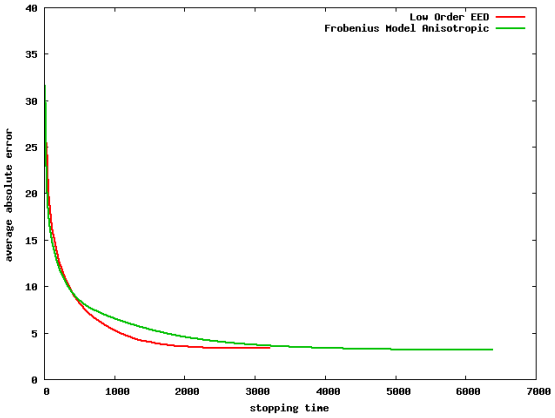


(e) Diff. Q64+BTTC(EED)-FM vs. original

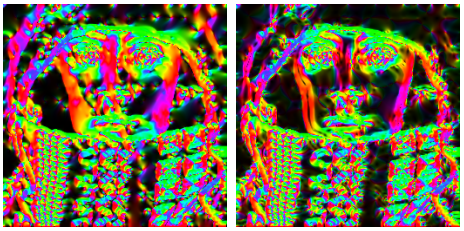


(f) Difference between the two interpolations

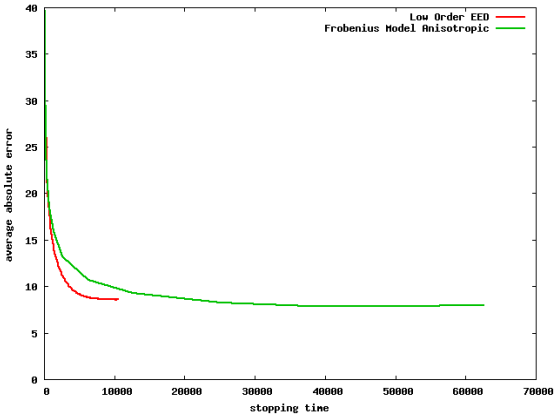
FIGURE 5.4: Interpolation of `Trui`: Q64+BTTC(EED) compression at 0.2 bpp



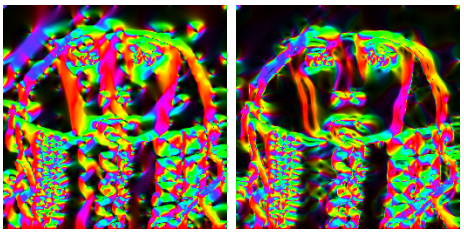
(a) Q64+BTTC(L)-* — 0.8 bpp



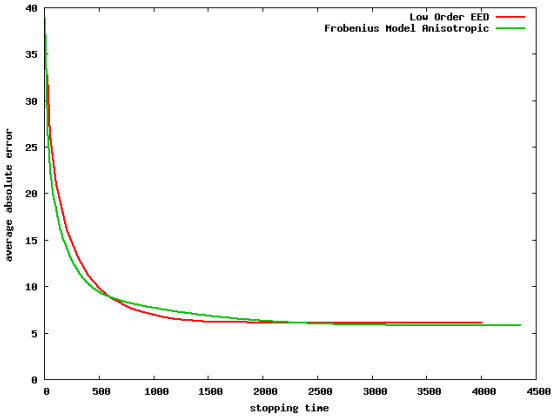
(b) Principal directions EED/FM



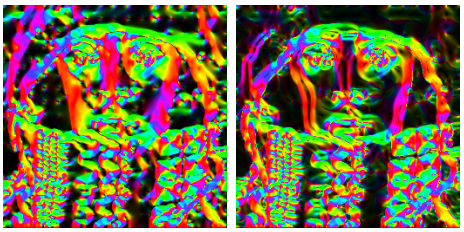
(c) Q64+BTTC(L)-* — 0.2 bpp



(d) Principal Directions EED/FM



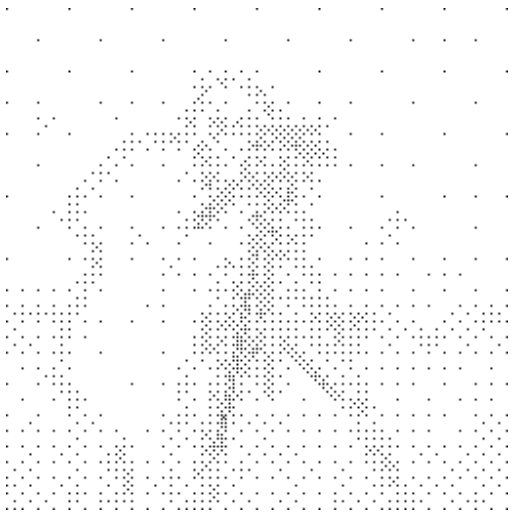
(e) Q64+BTTC(EED)-* — 0.2 bpp



(f) Principal Directions EED/FM

FIGURE 5.5: Data obtained during inpainting of `Trui`: ▶ **left**: evolution of the AAE and ▶ **right**: preferred smoothing direction in the last iteration.

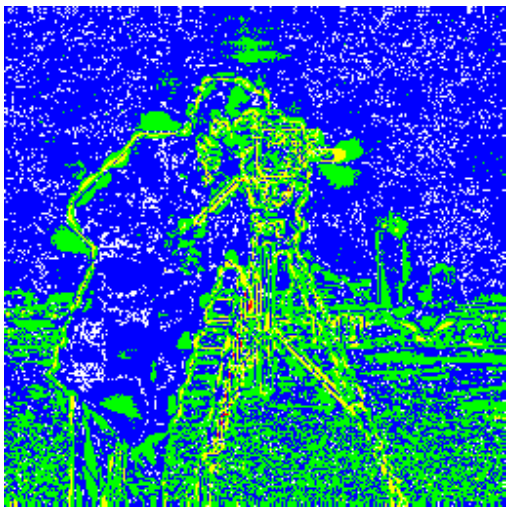
(a) Original image



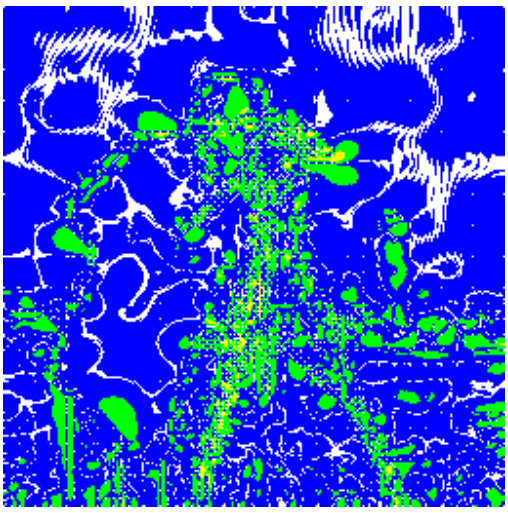
(b) Interpolation mask

(c) Q64+BTTC(EED)-EED

(d) Q64+BTTC(EED)-FM

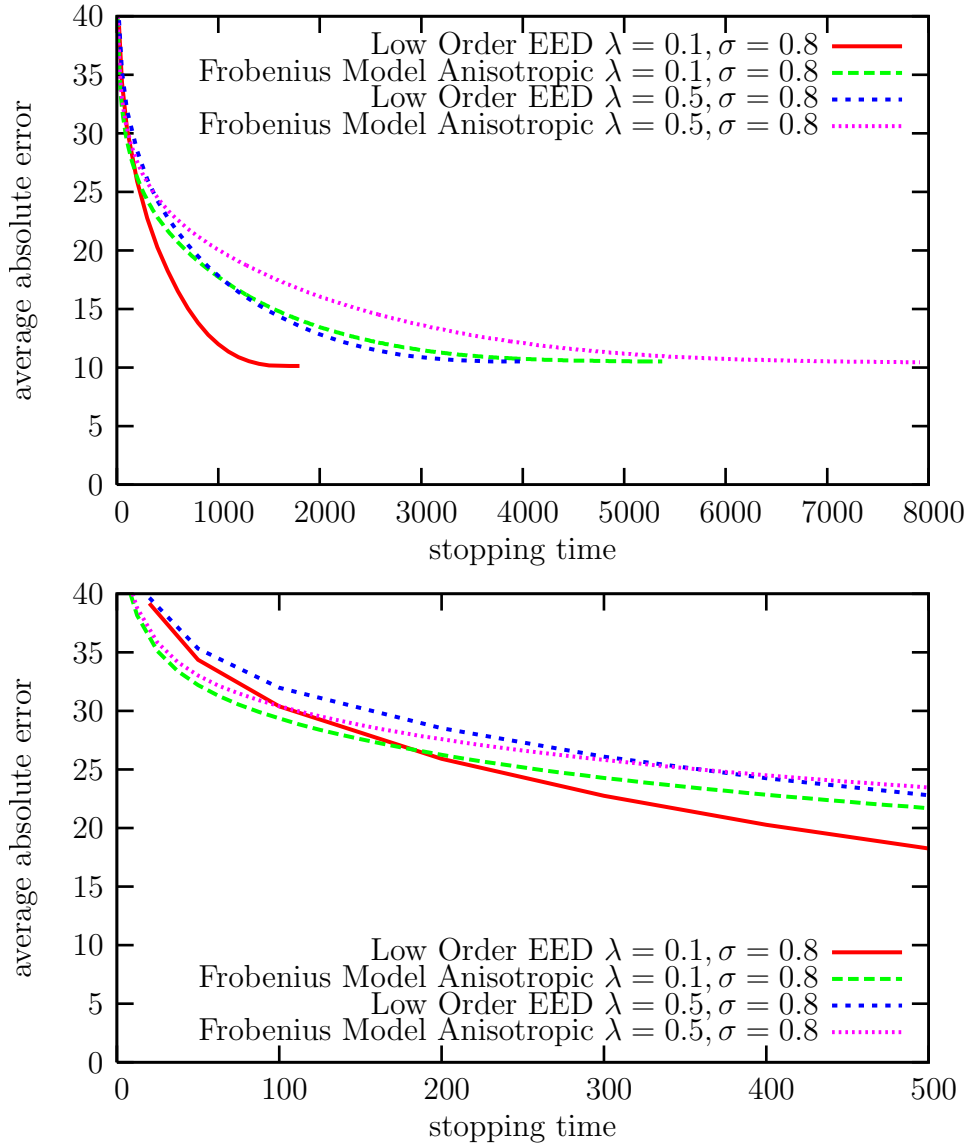


(e) Diff. Q64+BTTC(EED)-FM vs. original

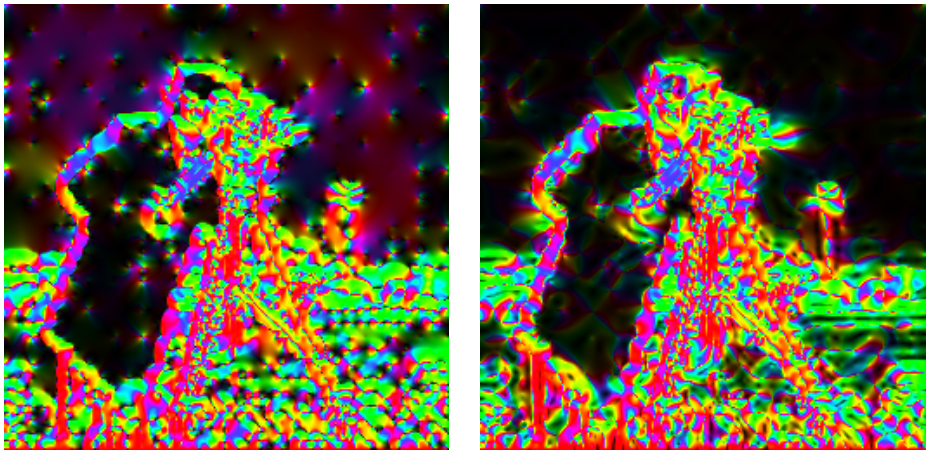


(f) Difference between the two interpolations

FIGURE 5.6: Interpolation of `Camera`: Q64+BTTC(EED) compression at 0.2 bpp.



(a) Reconstruction error of `Camera` inpainting using Q64+BTTC(EED)-* at 0.2 bpp
► **top**: overall ► **bottom**: zoom-in



(b) Principal directions — EED (c) Principal directions — FM

FIGURE 5.7: Data obtained during inpainting of `Camera`: ► **top**: evolution of the AAE and ► **bottom**: preferred smoothing directions in the last iteration.

Compression \ Decompression	EED	FM
`Trui` — **0.8 bpp**		
Q64+BTTC(L)	3.46 (t=3200)	3.30 (t=6362.5)
Q64+BTTC(EED)	—	—
`Trui` — **0.2 bpp**		
Q64+BTTC(L)	8.64 (t=10400)	7.98 (t=50000)
Q64+BTTC(EED)	6.20 (t=4000)	5.92 (t=4350)
`Camera` — **0.2 bpp**		
Q64+BTTC(L)	—	—
Q64+BTTC(EED) $\lambda = 0.5, \sigma = 0.8$	10.19 (t=1700)	10.58 (t=5375)
Q64+BTTC(EED) $\lambda = 0.1, \sigma = 0.8$	10.57 (t=4000)	10.55 (t=7912.5)

TABLE 5.1: Average absolute error values of inpainting results.

We conclude that, which one of both interpolants now is "better", is a matter of taste, and at least for low bitrates, probably also of the error measure in use. The evolution using the Frobenius model initially converges faster toward the original image than the classic EED interpolant but it has to run longer.

Additionally, it does not always provide slightly better quantitative results: for the `Camera` scene the result is at best equal or worse. We thus further conclude that using the Frobenius model works well with scenes that do not contain many minute details, of which `Camera` has a lot and `Trui` not so many.

We emphasize again, that if it were feasible to use FM already in the compression step, one might be able to generate seed points that would allow even better decompression at the same bitrates. Probably, the optimization of the model parameters (λ and σ) could also squeeze the one or other less AAE unit out of the error. To be concise, one by the way has to compute the error measure from the floating point result, which is usually 0.1-1 AAE unit better than e.g. the actual resulting image quantized to 256 integer values.

One could further argue that there is a certain link between the JPEG/-JPEG2000 methods (as seen in Figure 5.1) and low/high order EED respectively. Where JPEG produces blocky artifacts at high compression rates, JPEG2000 produces blurred areas. For our evolution processes this is similar: low order EED produces segmentation-like results whereas the Frobenius model keeps the structure intact but rather blurs its boundaries.

Chapter 6

Summary and Outlook

First, we have given an introduction into lower order diffusion methods, including discrete aspects and numerics, while occasionally divulging details not previously much discussed (to the best of our knowledge), like a formal presentation of stencil notation and analysis of anisotropic flux.

As a first step into higher order methods, we have explored the (isotropic) mixed order evolution equation by Tumbin and Turk, which we have used as part of our experiments. We have shown discretizations, including stencil notation and theoretical aspects. Afterward, we have introduced our novel *Frobenius model*, which includes a formulation that encompasses a whole class of fourth order image smoothers, including anisotropic ones. Unlike the Tumblin/Turk evolution equation, the Frobenius model can be recast in variational formulation, as we have shown. We have also introduced appropriate higher order boundary conditions.

In our experiments, we have juxtaposed Tumblin/Turk, Frobenius model and lower order methods, for several parameter sets. We have examined edge enhancement (or at least edge preservation) using extreme parameters on a standard test image to get a feeling for how and where they smoothen. We have also examined behavior with noisy data. In all cases, finding the right model parameters is not always easy and most of the time resorts in the principle of trial and error.

We have found that higher order methods do not produce staircasing or segmentation-like artifacts but rather create gradients; as such results seem more smooth. The anisotropic Tumblin/Turk evolution equation did not work well, but the anisotropic Frobenius model did. Higher order methods have also turned out competitive in denoising, compared to lower order methods.

As far as inpainting is concerned, results look smoother using higher order methods. Obviously, staircasing artifacts that would have been created by lower order methods do no longer appear. The "worst" case would be a linear gradient, which can be considered blurry by a human observer, but which is also unobtrusive. Whether this is good or bad finally depends on the scene to be reconstructed: subjectively, the `Camera` scene looks worse with higher order methods than lower order ones. For `Trui`, the opposite is the case.

Future Work

Most of the future work that comes into mind, builds up on one another. We have tried to show some theory about flux of lower order anisotropic filters (EED). Full understanding of that will make the analysis of the higher order model much easier, including what would be the actual maximum time step size allowed to use. We have made conservative guesses, using which our experiments ran stable.

Numerics need tuning, as higher order methods only allow for a much smaller time step size in an explicit scheme than lower order methods, and even there it is already not big, as we know. A method that would allow to use an arbitrary time step size would be nice to have.

Using even better discretized mixed derivatives should be worth a try, this was not so much a problem with lower order methods. Also, using higher but odd order derivatives (see the remarks in Section 2.6.2), at least as argument to diffusivity, should be checked out.

As far as the inpainting application is concerned, one should try to further optimize parameters, which we have mostly reused (those that we have known) from the excellent work using lower order by Galić et al. [GWW+08]. The previously mentioned optimizations on numerics would come in handy, as the search for optimal parameters and the analysis of their behavior could proceed much more quickly. An interleaved low/high order process could be interesting.

The compression/decompression mismatch in the inpainting application would improve the results of this process a lot. If one imagines a linear gradient in an image, a lower order method in theory needs to store each pixel. A higher order method only needs to store the beginning and ending pixel of the gradient to reproduce it later, since that is exactly what it is designed for: making functions (at worst) linear. Again, this depends on a fast implementation.

To be able to implement new models more quickly (and also be able to analyze its properties) computing a discretization, including stencil and system matrix, from an evolution equation would be a useful.

In our case, one iteration takes about 60 ms for a 256×256 image using the anisotropic Frobenius model and 40 ms for EED, with compiler optimizations but no heavily optimized code on a fairly standard 2.8 GHz PC with 1 MB L2 cache. An explicit computation for a large stopping time is thus a tedious task. One might consider a multi-threaded implementation (one computer, multiple cores, shared memory). The image is split equally between threads. This results in a theoretical speed-up of approximately the number of cores involved, minus the synchronization time between iterations. As such, this is an alternative to producing better numerics, but only a brute force one.

Last but not least, coherence enhancing diffusion as we have seen not only preserves features but can restore features. Therefore it is worth finding out more about higher order coherence enhancing evolutions, which should be a straightforward extension from the anisotropic Frobenius model. We feel that we have but scratched the surface and many improvements and extensions remain to be made.

Appendix A

Encore

A.1 Derivation of Stencil for Anisotropic Frobenius Model

This section gives the detailed derivation of the stencil for the anisotropic Frobenius Model from Section 3.3.

The first step writes down (3.26) (page 47) with discretized inner derivatives. The second step orders these terms by pixel grid size factors. The third step unifies some of those terms, and the fourth step performs many simplifications that present themselves.

$$
\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}t}u \approx & -\frac{a_{i+1,j}^2}{h_x^4}\left(u_{i,j} - 2u_{i+1,j} + u_{i+2,j}\right) \\
& - \frac{a_{i+1,j}b_{i+1,j}}{4h_x^3h_y}\left(u_{i,j-1} - u_{i+1,j-1} - u_{i,j} + 2u_{i+1,j} - u_{i+2,j} - u_{i+1,j+1} + u_{i+2,j+1}\right) \\
& - \frac{a_{i+1,j}b_{i+1,j}}{4h_x^3h_y}\left(u_{i+1,j-1} - u_{i+2,j-1} + u_{i,j} - 2u_{i+1,j} + u_{i+2,j} - u_{i,j+1} + u_{i+1,j+1}\right) \\
& - \frac{b_{i+1,j}^2}{h_x^2h_y^2}\left(u_{i+1,j-1} - 2u_{i+1,j} + u_{i+1,j+1}\right) \\
& + \frac{2a_{i,j}^2}{h_x^4}\left(u_{i-1,j} - 2u_{i,j} + u_{i+1,j}\right) \\
& + \frac{2a_{i,j}b_{i,j}}{4h_x^3h_y}\left(u_{i-1,j-1} - u_{i,j-1} - u_{i-1,j} + 2u_{i,j} - u_{i+1,j} - u_{i,j+1} + u_{i+1,j+1}\right) \\
& + \frac{2a_{i,j}b_{i,j}}{4h_x^3h_y}\left(u_{i,j-1} - u_{i+1,j-1} + u_{i-1,j} - 2u_{i,j} + u_{i+1,j} - u_{i-1,j+1} + u_{i,j+1}\right) \\
& + \frac{2b_{i,j}^2}{h_x^2h_y^2}\left(u_{i,j-1} - 2u_{i,j} + u_{i,j+1}\right) \\
& - \frac{a_{i-1,j}^2}{h_x^4}\left(u_{i-2,j} - 2u_{i-1,j} + u_{i,j}\right) \\
& - \frac{a_{i-1,j}b_{i-1,j}}{4h_x^3h_y}\left(u_{i-2,j-1} - u_{i-1,j-1} - u_{i-2,j} + 2u_{i-1,j} - u_{i,j} - u_{i-1,j+1} + u_{i,j+1}\right)
\end{aligned}
$$

$$
\begin{aligned}
&- \frac{a_{i-1,j}b_{i-1,j}}{4h_x^3h_y}\left(u_{i-1,j-1} - u_{i,j-1} + u_{i-2,j} - 2u_{i-1,j} + u_{i,j} - u_{i-2,j+1} + u_{i-1,j+1}\right)\\
&- \frac{b_{i-1,j}^2}{h_x^2h_y^2}\left(u_{i-1,j-1} - 2u_{i-1,j} + u_{i-1,j+1}\right)\\
&- \frac{a_{i-1,j-1}b_{i-1,j-1}}{2h_x^3h_y}\left(u_{i-2,j-1} - 2u_{i-1,j-1} + u_{i,j-1}\right)\\
&- \frac{a_{i-1,j-1}c_{i-1,j-1}}{4h_x^2h_y^2}\left(u_{i-2,j-2} - u_{i-1,j-2} - u_{i-2,j-1} + 2u_{i-1,j-1} - u_{i,j-1} - u_{i-1,j} + u_{i,j}\right)\\
&- \frac{b_{i-1,j-1}^2}{4h_x^2h_y^2}\left(u_{i-1,j-2} - u_{i,j-2} + u_{i-2,j-1} - 2u_{i-1,j-1} + u_{i,j-1} - u_{i-2,j} + u_{i-1,j}\right)\\
&- \frac{b_{i-1,j-1}c_{i-1,j-1}}{2h_xh_y^3}\left(u_{i-1,j-2} - 2u_{i-1,j-1} + u_{i-1,j}\right)\\
&+ \frac{a_{i,j-1}b_{i,j-1}}{2h_x^3h_y}\left(u_{i-1,j-1} - 2u_{i,j-1} + u_{i+1,j-1}\right)\\
&+ \frac{a_{i,j-1}c_{i,j-1}}{4h_x^2h_y^2}\left(u_{i-1,j-2} - u_{i,j-2} - u_{i-1,j-1} + 2u_{i,j-1} - u_{i+1,j-1} - u_{i,j} + u_{i+1,j}\right)\\
&+ \frac{b_{i,j-1}^2}{4h_x^2h_y^2}\left(u_{i,j-2} - u_{i+1,j-2} + u_{i-1,j-1} - 2u_{i,j-1} + u_{i+1,j-1} - u_{i-1,j} + u_{i,j}\right)\\
&+ \frac{b_{i,j-1}c_{i,j-1}}{2h_xh_y^3}\left(u_{i,j-2} - 2u_{i,j-1} + u_{i,j}\right)\\
&+ \frac{a_{i-1,j}b_{i-1,j}}{2h_x^3h_y}\left(u_{i-2,j} - 2u_{i-1,j} + u_{i,j}\right)\\
&+ \frac{a_{i-1,j}c_{i-1,j}}{4h_x^2h_y^2}\left(u_{i-2,j-1} - u_{i-1,j-1} - u_{i-2,j} + 2u_{i-1,j} - u_{i,j} - u_{i-1,j+1} + u_{i,j+1}\right)\\
&+ \frac{b_{i-1,j}^2}{4h_x^2h_y^2}\left(u_{i-1,j-1} - u_{i,j-1} + u_{i-2,j} - 2u_{i-1,j} + u_{i,j} - u_{i-2,j+1} + u_{i-1,j+1}\right)\\
&+ \frac{b_{i-1,j}c_{i-1,j}}{2h_xh_y^3}\left(u_{i-1,j-1} - 2u_{i-1,j} + u_{i-1,j+1}\right)\\
&- \frac{2a_{i,j}b_{i,j}}{2h_x^3h_y}\left(u_{i-1,j} - 2u_{i,j} + u_{i+1,j}\right)\\
&- \frac{2a_{i,j}c_{i,j}}{4h_x^2h_y^2}\left(u_{i-1,j-1} - u_{i,j-1} - u_{i-1,j} + 2u_{i,j} - u_{i+1,j} - u_{i,j+1} + u_{i+1,j+1}\right)\\
&- \frac{2b_{i,j}^2}{4h_x^2h_y^2}\left(u_{i,j-1} - u_{i+1,j-1} + u_{i-1,j} - 2u_{i,j} + u_{i+1,j} - u_{i-1,j+1} + u_{i,j+1}\right)\\
&- \frac{2b_{i,j}c_{i,j}}{2h_xh_y^3}\left(u_{i,j-1} - 2u_{i,j} + u_{i,j+1}\right)\\
&+ \frac{a_{i+1,j}b_{i+1,j}}{2h_x^3h_y}\left(u_{i,j} - 2u_{i+1,j} + u_{i+2,j}\right)\\
&+ \frac{a_{i+1,j}c_{i+1,j}}{4h_x^2h_y^2}\left(u_{i,j-1} - u_{i+1,j-1} - u_{i,j} + 2u_{i+1,j} - u_{i+2,j} - u_{i+1,j+1} + u_{i+2,j+1}\right)\\
&+ \frac{b_{i+1,j}^2}{4h_x^2h_y^2}\left(u_{i+1,j-1} - u_{i+2,j-1} + u_{i,j} - 2u_{i+1,j} + u_{i+2,j} - u_{i,j+1} + u_{i+1,j+1}\right)\\
&+ \frac{b_{i+1,j}c_{i+1,j}}{2h_xh_y^3}\left(u_{i+1,j-1} - 2u_{i+1,j} + u_{i+1,j+1}\right)\\
&+ \frac{a_{i,j+1}b_{i,j+1}}{2h_x^3h_y}\left(u_{i-1,j+1} - 2u_{i,j+1} + u_{i+1,j+1}\right)
\end{aligned}
$$

$$
\begin{aligned}
&+ \frac{a_{i,j+1}c_{i,j+1}}{4h_x^2h_y^2}\left(u_{i-1,j} - u_{i,j} - u_{i-1,j+1} + 2u_{i,j+1} - u_{i+1,j+1} - u_{i,j+2} + u_{i+1,j+2}\right)\\
&+ \frac{b_{i,j+1}^2}{4h_x^2h_y^2}\left(u_{i,j} - u_{i+1,j} + u_{i-1,j+1} - 2u_{i,j+1} + u_{i+1,j+1} - u_{i-1,j+2} + u_{i,j+2}\right)\\
&+ \frac{b_{i,j+1}c_{i,j+1}}{2h_xh_y^3}\left(u_{i,j} - 2u_{i,j+1} + u_{i,j+2}\right)\\
&- \frac{a_{i+1,j+1}b_{i+1,j+1}}{2h_x^3h_y}\left(u_{i,j+1} - 2u_{i+1,j+1} + u_{i+2,j+1}\right)\\
&- \frac{a_{i+1,j+1}c_{i+1,j+1}}{4h_x^2h_y^2}\left(u_{i,j} - u_{i+1,j} - u_{i,j+1} + 2u_{i+1,j+1} - u_{i+2,j+1} - u_{i+1,j+2} + u_{i+2,j+2}\right)\\
&- \frac{b_{i+1,j+1}^2}{4h_x^2h_y^2}\left(u_{i+1,j} - u_{i+2,j} + u_{i,j+1} - 2u_{i+1,j+1} + u_{i+2,j+1} - u_{i,j+2} + u_{i+1,j+2}\right)\\
&- \frac{b_{i+1,j+1}c_{i+1,j+1}}{2h_xh_y^3}\left(u_{i+1,j} - 2u_{i+1,j+1} + u_{i+1,j+2}\right)\\
&- \frac{a_{i,j-1}b_{i,j-1}}{2h_x^3h_y}\left(u_{i-1,j-1} - 2u_{i,j-1} + u_{i+1,j-1}\right)\\
&- \frac{b_{i,j-1}^2}{4h_x^2h_y^2}\left(u_{i-1,j-2} - u_{i,j-2} - u_{i-1,j-1} + 2u_{i,j-1} - u_{i+1,j-1} - u_{i,j} + u_{i+1,j}\right)\\
&- \frac{a_{i,j-1}c_{i,j-1}}{4h_x^2h_y^2}\left(u_{i,j-2} - u_{i+1,j-2} + u_{i-1,j-1} - 2u_{i,j-1} + u_{i+1,j-1} - u_{i-1,j} + u_{i,j}\right)\\
&- \frac{b_{i,j-1}c_{i,j-1}}{2h_xh_y^3}\left(u_{i,j-2} - 2u_{i,j-1} + u_{i,j}\right)\\
&+ \frac{a_{i+1,j-1}b_{i+1,j-1}}{2h_x^3h_y}\left(u_{i,j-1} - 2u_{i+1,j-1} + u_{i+2,j-1}\right)\\
&+ \frac{b_{i+1,j-1}^2}{4h_x^2h_y^2}\left(u_{i,j-2} - u_{i+1,j-2} - u_{i,j-1} + 2u_{i+1,j-1} - u_{i+2,j-1} - u_{i+1,j} + u_{i+2,j}\right)\\
&+ \frac{a_{i+1,j-1}c_{i+1,j-1}}{4h_x^2h_y^2}\left(u_{i+1,j-2} - u_{i+2,j-2} + u_{i,j-1} - 2u_{i+1,j-1} + u_{i+2,j-1} - u_{i,j} + u_{i+1,j}\right)\\
&+ \frac{b_{i+1,j-1}c_{i+1,j-1}}{2h_xh_y^3}\left(u_{i+1,j-2} - 2u_{i+1,j-1} + u_{i+1,j}\right)\\
&- \frac{a_{i-1,j}b_{i-1,j}}{2h_x^3h_y}\left(u_{i-2,j} - 2u_{i-1,j} + u_{i,j}\right)\\
&- \frac{b_{i-1,j}^2}{4h_x^2h_y^2}\left(u_{i-2,j-1} - u_{i-1,j-1} - u_{i-2,j} + 2u_{i-1,j} - u_{i,j} - u_{i-1,j+1} + u_{i,j+1}\right)\\
&- \frac{a_{i-1,j}c_{i-1,j}}{4h_x^2h_y^2}\left(u_{i-1,j-1} - u_{i,j-1} + u_{i-2,j} - 2u_{i-1,j} + u_{i,j} - u_{i-2,j+1} + u_{i-1,j+1}\right)\\
&- \frac{b_{i-1,j}c_{i-1,j}}{2h_xh_y^3}\left(u_{i-1,j-1} - 2u_{i-1,j} + u_{i-1,j+1}\right)\\
&+ \frac{2a_{i,j}b_{i,j}}{2h_x^3h_y}\left(u_{i-1,j} - 2u_{i,j} + u_{i+1,j}\right)\\
&+ \frac{2b_{i,j}^2}{4h_x^2h_y^2}\left(u_{i-1,j-1} - u_{i,j-1} - u_{i-1,j} + 2u_{i,j} - u_{i+1,j} - u_{i,j+1} + u_{i+1,j+1}\right)\\
&+ \frac{2a_{i,j}c_{i,j}}{4h_x^2h_y^2}\left(u_{i,j-1} - u_{i+1,j-1} + u_{i-1,j} - 2u_{i,j} + u_{i+1,j} - u_{i-1,j+1} + u_{i,j+1}\right)\\
&+ \frac{2b_{i,j}c_{i,j}}{2h_xh_y^3}\left(u_{i,j-1} - 2u_{i,j} + u_{i,j+1}\right)
\end{aligned}
$$

$$
\begin{aligned}
&- \frac{a_{i+1,j}b_{i+1,j}}{2h_x^3h_y}\left(u_{i,j} - 2u_{i+1,j} + u_{i+2,j}\right)\\
&- \frac{b_{i+1,j}^2}{4h_x^2h_y^2}\left(u_{i,j-1} - u_{i+1,j-1} - u_{i,j} + 2u_{i+1,j} - u_{i+2,j} - u_{i+1,j+1} + u_{i+2,j+1}\right)\\
&- \frac{a_{i+1,j}c_{i+1,j}}{4h_x^2h_y^2}\left(u_{i+1,j-1} - u_{i+2,j-1} + u_{i,j} - 2u_{i+1,j} + u_{i+2,j} - u_{i,j+1} + u_{i+1,j+1}\right)\\
&- \frac{b_{i+1,j}c_{i+1,j}}{2h_xh_y^3}\left(u_{i+1,j-1} - 2u_{i+1,j} + u_{i+1,j+1}\right)\\
&+ \frac{a_{i-1,j+1}b_{i-1,j+1}}{2h_x^3h_y}\left(u_{i-2,j+1} - 2u_{i-1,j+1} + u_{i,j+1}\right)\\
&+ \frac{b_{i-1,j+1}^2}{4h_x^2h_y^2}\left(u_{i-2,j} - u_{i-1,j} - u_{i-2,j+1} + 2u_{i-1,j+1} - u_{i,j+1} - u_{i-1,j+2} + u_{i,j+2}\right)\\
&+ \frac{a_{i-1,j+1}c_{i-1,j+1}}{4h_x^2h_y^2}\left(u_{i-1,j} - u_{i,j} + u_{i-2,j+1} - 2u_{i-1,j+1} + u_{i,j+1} - u_{i-2,j+2} + u_{i-1,j+2}\right)\\
&+ \frac{b_{i-1,j+1}c_{i-1,j+1}}{2h_xh_y^3}\left(u_{i-1,j} - 2u_{i-1,j+1} + u_{i-1,j+2}\right)\\
&- \frac{a_{i,j+1}b_{i,j+1}}{2h_x^3h_y}\left(u_{i-1,j+1} - 2u_{i,j+1} + u_{i+1,j+1}\right)\\
&- \frac{b_{i,j+1}^2}{4h_x^2h_y^2}\left(u_{i-1,j} - u_{i,j} - u_{i-1,j+1} + 2u_{i,j+1} - u_{i+1,j+1} - u_{i,j+2} + u_{i+1,j+2}\right)\\
&- \frac{a_{i,j+1}c_{i,j+1}}{4h_x^2h_y^2}\left(u_{i,j} - u_{i+1,j} + u_{i-1,j+1} - 2u_{i,j+1} + u_{i+1,j+1} - u_{i-1,j+2} + u_{i,j+2}\right)\\
&- \frac{b_{i,j+1}c_{i,j+1}}{2h_xh_y^3}\left(u_{i,j} - 2u_{i,j+1} + u_{i,j+2}\right)\\
&- \frac{b_{i,j+1}^2}{h_x^2h_y^2}\left(u_{i-1,j+1} - 2u_{i,j+1} + u_{i+1,j+1}\right)\\
&- \frac{b_{i,j+1}c_{i,j+1}}{4h_xh_y^3}\left(u_{i-1,j} - u_{i,j} - u_{i-1,j+1} + 2u_{i,j+1} - u_{i+1,j+1} - u_{i,j+2} + u_{i+1,j+2}\right)\\
&- \frac{b_{i,j+1}c_{i,j+1}}{4h_xh_y^3}\left(u_{i,j} - u_{i+1,j} + u_{i-1,j+1} - 2u_{i,j+1} + u_{i+1,j+1} - u_{i-1,j+2} + u_{i,j+2}\right)\\
&- \frac{c_{i,j+1}^2}{h_y^4}\left(u_{i,j} - 2u_{i,j+1} + u_{i,j+2}\right)\\
&+ \frac{2b_{i,j}^2}{h_x^2h_y^2}\left(u_{i-1,j} - 2u_{i,j} + u_{i+1,j}\right)\\
&+ \frac{2b_{i,j}c_{i,j}}{4h_xh_y^3}\left(u_{i-1,j-1} - u_{i,j-1} - u_{i-1,j} + 2u_{i,j} - u_{i+1,j} - u_{i,j+1} + u_{i+1,j+1}\right)\\
&+ \frac{2b_{i,j}c_{i,j}}{4h_xh_y^3}\left(u_{i,j-1} - u_{i+1,j-1} + u_{i-1,j} - 2u_{i,j} + u_{i+1,j} - u_{i-1,j+1} + u_{i,j+1}\right)\\
&+ \frac{2c_{i,j}^2}{h_y^4}\left(u_{i,j-1} - 2u_{i,j} + u_{i,j+1}\right)\\
&- \frac{b_{i,j-1}^2}{h_x^2h_y^2}\left(u_{i-1,j-1} - 2u_{i,j-1} + u_{i+1,j-1}\right)\\
&- \frac{b_{i,j-1}c_{i,j-1}}{4h_xh_y^3}\left(u_{i-1,j-2} - u_{i,j-2} - u_{i-1,j-1} + 2u_{i,j-1} - u_{i+1,j-1} - u_{i,j} + u_{i+1,j}\right)
\end{aligned}
$$

$$
\begin{aligned}
&- \frac{b_{i,j-1}c_{i,j-1}}{4h_x h_y^3}\left(u_{i,j-2} - u_{i+1,j-2} + u_{i-1,j-1} - 2u_{i,j-1} + u_{i+1,j-1} - u_{i-1,j} + u_{i,j}\right)\\
&- \frac{c_{i,j-1}^2}{h_y^4}\left(u_{i,j-2} - 2u_{i,j-1} + u_{i,j}\right)
\end{aligned}
$$

$$
\begin{aligned}
&= \frac{1}{h_x^4}\Big(-a_{i+1,j}^2\left(u_{i,j} - 2u_{i+1,j} + u_{i+2,j}\right)\\
&\qquad + 2a_{i,j}^2\left(u_{i-1,j} - 2u_{i,j} + u_{i+1,j}\right)\\
&\qquad - a_{i-1,j}^2\left(u_{i-2,j} - 2u_{i-1,j} + u_{i,j}\right)\Big)\\
&+\frac{1}{4h_x^3 h_y}\Big(-a_{i+1,j}b_{i+1,j}\left(u_{i,j-1} - u_{i+1,j-1} - u_{i,j} + 2u_{i+1,j} - u_{i+2,j} - u_{i+1,j+1} + u_{i+2,j+1}\right)\\
&\qquad - a_{i+1,j}b_{i+1,j}\left(u_{i+1,j-1} - u_{i+2,j-1} + u_{i,j} - 2u_{i+1,j} + u_{i+2,j} - u_{i,j+1} + u_{i+1,j+1}\right)\\
&\qquad - a_{i-1,j}b_{i-1,j}\left(u_{i-2,j-1} - u_{i-1,j-1} - u_{i-2,j} + 2u_{i-1,j} - u_{i,j} - u_{i-1,j+1} + u_{i,j+1}\right)\\
&\qquad - a_{i-1,j}b_{i-1,j}\left(u_{i-1,j-1} - u_{i,j-1} + u_{i-2,j} - 2u_{i-1,j} + u_{i,j} - u_{i-2,j+1} + u_{i-1,j+1}\right)\Big)\\
&+\frac{2}{4h_x^3 h_y}\Big(+a_{i,j}b_{i,j}\left(u_{i-1,j-1} - u_{i,j-1} - u_{i-1,j} + 2u_{i,j} - u_{i+1,j} - u_{i,j+1} + u_{i+1,j+1}\right)\\
&\qquad + a_{i,j}b_{i,j}\left(u_{i,j-1} - u_{i+1,j-1} + u_{i-1,j} - 2u_{i,j} + u_{i+1,j} - u_{i-1,j+1} + u_{i,j+1}\right)\Big)\\
&+\frac{1}{2h_x^3 h_y}\Big(-a_{i-1,j-1}b_{i-1,j-1}\left(u_{i-2,j-1} - 2u_{i-1,j-1} + u_{i,j-1}\right)\\
&\qquad + a_{i,j-1}b_{i,j-1}\left(u_{i-1,j-1} - 2u_{i,j-1} + u_{i+1,j-1}\right)\\
&\qquad + a_{i-1,j}b_{i-1,j}\left(u_{i-2,j} - 2u_{i-1,j} + u_{i,j}\right)\\
&\qquad + a_{i+1,j}b_{i+1,j}\left(u_{i,j} - 2u_{i+1,j} + u_{i+2,j}\right)\\
&\qquad + a_{i,j+1}b_{i,j+1}\left(u_{i-1,j+1} - 2u_{i,j+1} + u_{i+1,j+1}\right)\\
&\qquad - a_{i+1,j+1}b_{i+1,j+1}\left(u_{i,j+1} - 2u_{i+1,j+1} + u_{i+2,j+1}\right)\\
&\qquad - a_{i,j-1}b_{i,j-1}\left(u_{i-1,j-1} - 2u_{i,j-1} + u_{i+1,j-1}\right)\\
&\qquad + a_{i+1,j-1}b_{i+1,j-1}\left(u_{i,j-1} - 2u_{i+1,j-1} + u_{i+2,j-1}\right)\\
&\qquad - a_{i-1,j}b_{i-1,j}\left(u_{i-2,j} - 2u_{i-1,j} + u_{i,j}\right)\\
&\qquad - a_{i+1,j}b_{i+1,j}\left(u_{i,j} - 2u_{i+1,j} + u_{i+2,j}\right)\\
&\qquad + a_{i-1,j+1}b_{i-1,j+1}\left(u_{i-2,j+1} - 2u_{i-1,j+1} + u_{i,j+1}\right)\\
&\qquad - a_{i,j+1}b_{i,j+1}\left(u_{i-1,j+1} - 2u_{i,j+1} + u_{i+1,j+1}\right)\Big)\\
&+\frac{2}{2h_x^3 h_y}\Big(-a_{i,j}b_{i,j}\left(u_{i-1,j} - 2u_{i,j} + u_{i+1,j}\right)\\
&\qquad + a_{i,j}b_{i,j}\left(u_{i-1,j} - 2u_{i,j} + u_{i+1,j}\right)\Big)\\
&+\frac{1}{h_x^2 h_y^2}\Big(-b_{i,j+1}^2\left(u_{i-1,j+1} - 2u_{i,j+1} + u_{i+1,j+1}\right)\\
&\qquad - b_{i,j-1}^2\left(u_{i-1,j-1} - 2u_{i,j-1} + u_{i+1,j-1}\right)\\
&\qquad - b_{i+1,j}^2\left(u_{i+1,j-1} - 2u_{i+1,j} + u_{i+1,j+1}\right)\\
&\qquad - b_{i-1,j}^2\left(u_{i-1,j-1} - 2u_{i-1,j} + u_{i-1,j+1}\right)\Big)\\
&+\frac{2}{h_x^2 h_y^2}\Big(+b_{i,j}^2\left(u_{i-1,j} - 2u_{i,j} + u_{i+1,j}\right)
\end{aligned}
$$

$$
\begin{aligned}
&\quad + b_{i,j}^2\left(u_{i,j-1} - 2u_{i,j} + u_{i,j+1}\right)\Big) \\
&+\frac{1}{4h_x^2h_y^2}\Big(-a_{i-1,j-1}c_{i-1,j-1}\left(u_{i-2,j-2} - u_{i-1,j-2} - u_{i-2,j-1} + 2u_{i-1,j-1} - u_{i,j-1} - u_{i-1,j} + u_{i,j}\right) \\
&\quad - b_{i-1,j-1}^2\left(u_{i-1,j-2} - u_{i,j-2} + u_{i-2,j-1} - 2u_{i-1,j-1} + u_{i,j-1} - u_{i-2,j} + u_{i-1,j}\right) \\
&\quad + a_{i,j-1}c_{i,j-1}\left(u_{i-1,j-2} - u_{i,j-2} - u_{i-1,j-1} + 2u_{i,j-1} - u_{i+1,j-1} - u_{i,j} + u_{i+1,j}\right) \\
&\quad + b_{i,j-1}^2\left(u_{i,j-2} - u_{i+1,j-2} + u_{i-1,j-1} - 2u_{i,j-1} + u_{i+1,j-1} - u_{i-1,j} + u_{i,j}\right) \\
&\quad + a_{i-1,j}c_{i-1,j}\left(u_{i-2,j-1} - u_{i-1,j-1} - u_{i-2,j} + 2u_{i-1,j} - u_{i,j} - u_{i-1,j+1} + u_{i,j+1}\right) \\
&\quad + b_{i-1,j}^2\left(u_{i-1,j-1} - u_{i,j-1} + u_{i-2,j} - 2u_{i-1,j} + u_{i,j} - u_{i-2,j+1} + u_{i-1,j+1}\right) \\
&\quad + a_{i+1,j}c_{i+1,j}\left(u_{i,j-1} - u_{i+1,j-1} - u_{i,j} + 2u_{i+1,j} - u_{i+2,j} - u_{i+1,j+1} + u_{i+2,j+1}\right) \\
&\quad + b_{i+1,j}^2\left(u_{i+1,j-1} - u_{i+2,j-1} + u_{i,j} - 2u_{i+1,j} + u_{i+2,j} - u_{i,j+1} + u_{i+1,j+1}\right) \\
&\quad + a_{i,j+1}c_{i,j+1}\left(u_{i-1,j} - u_{i,j} - u_{i-1,j+1} + 2u_{i,j+1} - u_{i+1,j+1} - u_{i,j+2} + u_{i+1,j+2}\right) \\
&\quad + b_{i,j+1}^2\left(u_{i,j} - u_{i+1,j} + u_{i-1,j+1} - 2u_{i,j+1} + u_{i+1,j+1} - u_{i-1,j+2} + u_{i,j+2}\right) \\
&\quad - a_{i+1,j+1}c_{i+1,j+1}\left(u_{i,j} - u_{i+1,j} - u_{i,j+1} + 2u_{i+1,j+1} - u_{i+2,j+1} - u_{i+1,j+2} + u_{i+2,j+2}\right) \\
&\quad - b_{i+1,j+1}^2\left(u_{i+1,j} - u_{i+2,j} + u_{i,j+1} - 2u_{i+1,j+1} + u_{i+2,j+1} - u_{i,j+2} + u_{i+1,j+2}\right) \\
&\quad - b_{i,j-1}^2\left(u_{i-1,j-2} - u_{i,j-2} - u_{i-1,j-1} + 2u_{i,j-1} - u_{i+1,j-1} - u_{i,j} + u_{i+1,j}\right) \\
&\quad - a_{i,j-1}c_{i,j-1}\left(u_{i,j-2} - u_{i+1,j-2} + u_{i-1,j-1} - 2u_{i,j-1} + u_{i+1,j-1} - u_{i-1,j} + u_{i,j}\right) \\
&\quad + b_{i+1,j-1}^2\left(u_{i,j-2} - u_{i+1,j-2} - u_{i,j-1} + 2u_{i+1,j-1} - u_{i+2,j-1} - u_{i+1,j} + u_{i+2,j}\right) \\
&\quad + a_{i+1,j-1}c_{i+1,j-1}\left(u_{i+1,j-2} - u_{i+2,j-2} + u_{i,j-1} - 2u_{i+1,j-1} + u_{i+2,j-1} - u_{i,j} + u_{i+1,j}\right) \\
&\quad - b_{i-1,j}^2\left(u_{i-2,j-1} - u_{i-1,j-1} - u_{i-2,j} + 2u_{i-1,j} - u_{i,j} - u_{i-1,j+1} + u_{i,j+1}\right) \\
&\quad - a_{i-1,j}c_{i-1,j}\left(u_{i-1,j-1} - u_{i,j-1} + u_{i-2,j} - 2u_{i-1,j} + u_{i,j} - u_{i-2,j+1} + u_{i-1,j+1}\right) \\
&\quad - b_{i+1,j}^2\left(u_{i,j-1} - u_{i+1,j-1} - u_{i,j} + 2u_{i+1,j} - u_{i+2,j} - u_{i+1,j+1} + u_{i+2,j+1}\right) \\
&\quad - a_{i+1,j}c_{i+1,j}\left(u_{i+1,j-1} - u_{i+2,j-1} + u_{i,j} - 2u_{i+1,j} + u_{i+2,j} - u_{i,j+1} + u_{i+1,j+1}\right) \\
&\quad + b_{i-1,j+1}^2\left(u_{i-2,j} - u_{i-1,j} - u_{i-2,j+1} + 2u_{i-1,j+1} - u_{i,j+1} - u_{i-1,j+2} + u_{i,j+2}\right) \\
&\quad + a_{i-1,j+1}c_{i-1,j+1}\left(u_{i-1,j} - u_{i,j} + u_{i-2,j+1} - 2u_{i-1,j+1} + u_{i,j+1} - u_{i-2,j+2} + u_{i-1,j+2}\right) \\
&\quad - b_{i,j+1}^2\left(u_{i-1,j} - u_{i,j} - u_{i-1,j+1} + 2u_{i,j+1} - u_{i+1,j+1} - u_{i,j+2} + u_{i+1,j+2}\right) \\
&\quad - a_{i,j+1}c_{i,j+1}\left(u_{i,j} - u_{i+1,j} + u_{i-1,j+1} - 2u_{i,j+1} + u_{i+1,j+1} - u_{i-1,j+2} + u_{i,j+2}\right)\Big) \\
&+\frac{2}{4h_x^2h_y^2}\Big(-a_{i,j}c_{i,j}\left(u_{i-1,j-1} - u_{i,j-1} - u_{i-1,j} + 2u_{i,j} - u_{i+1,j} - u_{i,j+1} + u_{i+1,j+1}\right) \\
&\quad - b_{i,j}^2\left(u_{i,j-1} - u_{i+1,j-1} + u_{i-1,j} - 2u_{i,j} + u_{i+1,j} - u_{i-1,j+1} + u_{i,j+1}\right) \\
&\quad + b_{i,j}^2\left(u_{i-1,j-1} - u_{i,j-1} - u_{i-1,j} + 2u_{i,j} - u_{i+1,j} - u_{i,j+1} + u_{i+1,j+1}\right) \\
&\quad + a_{i,j}c_{i,j}\left(u_{i,j-1} - u_{i+1,j-1} + u_{i-1,j} - 2u_{i,j} + u_{i+1,j} - u_{i-1,j+1} + u_{i,j+1}\right)\Big) \\
&+\frac{1}{2h_xh_y^3}\Big(-b_{i-1,j-1}c_{i-1,j-1}\left(u_{i-1,j-2} - 2u_{i-1,j-1} + u_{i-1,j}\right) \\
&\quad + b_{i,j-1}c_{i,j-1}\left(u_{i,j-2} - 2u_{i,j-1} + u_{i,j}\right) \\
&\quad + b_{i-1,j}c_{i-1,j}\left(u_{i-1,j-1} - 2u_{i-1,j} + u_{i-1,j+1}\right) \\
&\quad + b_{i+1,j}c_{i+1,j}\left(u_{i+1,j-1} - 2u_{i+1,j} + u_{i+1,j+1}\right) \\
&\quad + b_{i,j+1}c_{i,j+1}\left(u_{i,j} - 2u_{i,j+1} + u_{i,j+2}\right) \\
&\quad - b_{i+1,j+1}c_{i+1,j+1}\left(u_{i+1,j} - 2u_{i+1,j+1} + u_{i+1,j+2}\right) \\
&\quad - b_{i,j-1}c_{i,j-1}\left(u_{i,j-2} - 2u_{i,j-1} + u_{i,j}\right) \\
&\quad + b_{i+1,j-1}c_{i+1,j-1}\left(u_{i+1,j-2} - 2u_{i+1,j-1} + u_{i+1,j}\right)
\end{aligned}
$$

$$
\begin{aligned}
&\quad - b_{i-1,j}c_{i-1,j}\left(u_{i-1,j-1} - 2u_{i-1,j} + u_{i-1,j+1}\right)\\
&\quad - b_{i+1,j}c_{i+1,j}\left(u_{i+1,j-1} - 2u_{i+1,j} + u_{i+1,j+1}\right)\\
&\quad + b_{i-1,j+1}c_{i-1,j+1}\left(u_{i-1,j} - 2u_{i-1,j+1} + u_{i-1,j+2}\right)\\
&\quad - b_{i,j+1}c_{i,j+1}\left(u_{i,j} - 2u_{i,j+1} + u_{i,j+2}\right)\Big)\\
&+\frac{2}{2h_xh_y^3}\Big(-b_{i,j}c_{i,j}\left(u_{i,j-1} - 2u_{i,j} + u_{i,j+1}\right)\\
&\quad + b_{i,j}c_{i,j}\left(u_{i,j-1} - 2u_{i,j} + u_{i,j+1}\right)\Big)\\
&+\frac{1}{4h_xh_y^3}\Big(-b_{i,j+1}c_{i,j+1}\left(u_{i-1,j} - u_{i,j} - u_{i-1,j+1} + 2u_{i,j+1} - u_{i+1,j+1} - u_{i,j+2} + u_{i+1,j+2}\right)\\
&\quad - b_{i,j+1}c_{i,j+1}\left(u_{i,j} - u_{i+1,j} + u_{i-1,j+1} - 2u_{i,j+1} + u_{i+1,j+1} - u_{i-1,j+2} + u_{i,j+2}\right)\\
&\quad - b_{i,j-1}c_{i,j-1}\left(u_{i-1,j-2} - u_{i,j-2} - u_{i-1,j-1} + 2u_{i,j-1} - u_{i+1,j-1} - u_{i,j} + u_{i+1,j}\right)\\
&\quad - b_{i,j-1}c_{i,j-1}\left(u_{i,j-2} - u_{i+1,j-2} + u_{i-1,j-1} - 2u_{i,j-1} + u_{i+1,j-1} - u_{i-1,j} + u_{i,j}\right)\Big)\\
&+\frac{2}{4h_xh_y^3}\Big(+b_{i,j}c_{i,j}\left(u_{i-1,j-1} - u_{i,j-1} - u_{i-1,j} + 2u_{i,j} - u_{i+1,j} - u_{i,j+1} + u_{i+1,j+1}\right)\\
&\quad + b_{i,j}c_{i,j}\left(u_{i,j-1} - u_{i+1,j-1} + u_{i-1,j} - 2u_{i,j} + u_{i+1,j} - u_{i-1,j+1} + u_{i,j+1}\right)\Big)\\
&\quad +\frac{1}{h_y^4}\Big(-c_{i,j+1}^2\left(u_{i,j} - 2u_{i,j+1} + u_{i,j+2}\right)\\
&\quad + 2c_{i,j}^2\left(u_{i,j-1} - 2u_{i,j} + u_{i,j+1}\right)\\
&\quad - c_{i,j-1}^2\left(u_{i,j-2} - 2u_{i,j-1} + u_{i,j}\right)\Big)
\end{aligned}
$$

$$
\begin{aligned}
&= \frac{1}{h_x^4}\Big(-a_{i+1,j}^2\left(u_{i,j} - 2u_{i+1,j} + u_{i+2,j}\right)\\
&\quad + 2a_{i,j}^2\left(u_{i-1,j} - 2u_{i,j} + u_{i+1,j}\right)\\
&\quad - a_{i-1,j}^2\left(u_{i-2,j} - 2u_{i-1,j} + u_{i,j}\right)\Big)\\
&+\frac{1}{4h_x^3h_y}\Big(-a_{i+1,j}b_{i+1,j}\left(u_{i,j-1} - u_{i+1,j-1} - u_{i,j} + 2u_{i+1,j} - u_{i+2,j} - u_{i+1,j+1} + u_{i+2,j+1}\right)\\
&\quad - a_{i+1,j}b_{i+1,j}\left(u_{i+1,j-1} - u_{i+2,j-1} + u_{i,j} - 2u_{i+1,j} + u_{i+2,j} - u_{i,j+1} + u_{i+1,j+1}\right)\\
&\quad + 2a_{i,j}b_{i,j}\left(u_{i-1,j-1} - u_{i,j-1} - u_{i-1,j} + 2u_{i,j} - u_{i+1,j} - u_{i,j+1} + u_{i+1,j+1}\right)\\
&\quad + 2a_{i,j}b_{i,j}\left(u_{i,j-1} - u_{i+1,j-1} + u_{i-1,j} - 2u_{i,j} + u_{i+1,j} - u_{i-1,j+1} + u_{i,j+1}\right)\\
&\quad - a_{i-1,j}b_{i-1,j}\left(u_{i-2,j-1} - u_{i-1,j-1} - u_{i-2,j} + 2u_{i-1,j} - u_{i,j} - u_{i-1,j+1} + u_{i,j+1}\right)\\
&\quad - a_{i-1,j}b_{i-1,j}\left(u_{i-1,j-1} - u_{i,j-1} + u_{i-2,j} - 2u_{i-1,j} + u_{i,j} - u_{i-2,j+1} + u_{i-1,j+1}\right)\Big)\\
&+\frac{1}{2h_x^3h_y}\Big(-a_{i-1,j-1}b_{i-1,j-1}\left(u_{i-2,j-1} - 2u_{i-1,j-1} + u_{i,j-1}\right)\\
&\quad + a_{i,j-1}b_{i,j-1}\left(u_{i-1,j-1} - 2u_{i,j-1} + u_{i+1,j-1}\right)\\
&\quad + a_{i-1,j}b_{i-1,j}\left(u_{i-2,j} - 2u_{i-1,j} + u_{i,j}\right)\\
&\quad + a_{i+1,j}b_{i+1,j}\left(u_{i,j} - 2u_{i+1,j} + u_{i+2,j}\right)\\
&\quad + a_{i,j+1}b_{i,j+1}\left(u_{i-1,j+1} - 2u_{i,j+1} + u_{i+1,j+1}\right)\\
&\quad - a_{i+1,j+1}b_{i+1,j+1}\left(u_{i,j+1} - 2u_{i+1,j+1} + u_{i+2,j+1}\right)\\
&\quad - a_{i,j-1}b_{i,j-1}\left(u_{i-1,j-1} - 2u_{i,j-1} + u_{i+1,j-1}\right)\\
&\quad + a_{i+1,j-1}b_{i+1,j-1}\left(u_{i,j-1} - 2u_{i+1,j-1} + u_{i+2,j-1}\right)
\end{aligned}
$$

$$
\begin{aligned}
&- a_{i-1,j}b_{i-1,j}\left(u_{i-2,j} - 2u_{i-1,j} + u_{i,j}\right)\\
&- a_{i+1,j}b_{i+1,j}\left(u_{i,j} - 2u_{i+1,j} + u_{i+2,j}\right)\\
&+ a_{i-1,j+1}b_{i-1,j+1}\left(u_{i-2,j+1} - 2u_{i-1,j+1} + u_{i,j+1}\right)\\
&- a_{i,j+1}b_{i,j+1}\left(u_{i-1,j+1} - 2u_{i,j+1} + u_{i+1,j+1}\right)\\
&- 2a_{i,j}b_{i,j}\left(u_{i-1,j} - 2u_{i,j} + u_{i+1,j}\right)\\
&+ 2a_{i,j}b_{i,j}\left(u_{i-1,j} - 2u_{i,j} + u_{i+1,j}\right)\Big)\\
+\frac{1}{h_x^2h_y^2}\Big(&-b_{i,j+1}^2\left(u_{i-1,j+1} - 2u_{i,j+1} + u_{i+1,j+1}\right)\\
&- b_{i,j-1}^2\left(u_{i-1,j-1} - 2u_{i,j-1} + u_{i+1,j-1}\right)\\
&+ 2b_{i,j}^2\left(u_{i-1,j} - 2u_{i,j} + u_{i+1,j}\right)\\
&+ 2b_{i,j}^2\left(u_{i,j-1} - 2u_{i,j} + u_{i,j+1}\right)\\
&- b_{i+1,j}^2\left(u_{i+1,j-1} - 2u_{i+1,j} + u_{i+1,j+1}\right)\\
&- b_{i-1,j}^2\left(u_{i-1,j-1} - 2u_{i-1,j} + u_{i-1,j+1}\right)\Big)\\
+\frac{1}{4h_x^2h_y^2}\Big(&-a_{i-1,j-1}c_{i-1,j-1}\left(u_{i-2,j-2} - u_{i-1,j-2} - u_{i-2,j-1} + 2u_{i-1,j-1} - u_{i,j-1} - u_{i-1,j} + u_{i,j}\right)\\
&- b_{i-1,j-1}^2\left(u_{i-1,j-2} - u_{i,j-2} + u_{i-2,j-1} - 2u_{i-1,j-1} + u_{i,j-1} - u_{i-2,j} + u_{i-1,j}\right)\\
&+ a_{i,j-1}c_{i,j-1}\left(u_{i-1,j-2} - u_{i,j-2} - u_{i-1,j-1} + 2u_{i,j-1} - u_{i+1,j-1} - u_{i,j} + u_{i+1,j}\right)\\
&+ b_{i,j-1}^2\left(u_{i,j-2} - u_{i+1,j-2} + u_{i-1,j-1} - 2u_{i,j-1} + u_{i+1,j-1} - u_{i-1,j} + u_{i,j}\right)\\
&+ a_{i-1,j}c_{i-1,j}\left(u_{i-2,j-1} - u_{i-1,j-1} - u_{i-2,j} + 2u_{i-1,j} - u_{i,j} - u_{i-1,j+1} + u_{i,j+1}\right)\\
&+ b_{i-1,j}^2\left(u_{i-1,j-1} - u_{i,j-1} + u_{i-2,j} - 2u_{i-1,j} + u_{i,j} - u_{i-2,j+1} + u_{i-1,j+1}\right)\\
&+ a_{i+1,j}c_{i+1,j}\left(u_{i,j-1} - u_{i+1,j-1} - u_{i,j} + 2u_{i+1,j} - u_{i+2,j} - u_{i+1,j+1} + u_{i+2,j+1}\right)\\
&+ b_{i+1,j}^2\left(u_{i+1,j-1} - u_{i+2,j-1} + u_{i,j} - 2u_{i+1,j} + u_{i+2,j} - u_{i,j+1} + u_{i+1,j+1}\right)\\
&+ a_{i,j+1}c_{i,j+1}\left(u_{i-1,j} - u_{i,j} - u_{i-1,j+1} + 2u_{i,j+1} - u_{i+1,j+1} - u_{i,j+2} + u_{i+1,j+2}\right)\\
&+ b_{i,j+1}^2\left(u_{i,j} - u_{i+1,j} + u_{i-1,j+1} - 2u_{i,j+1} + u_{i+1,j+1} - u_{i-1,j+2} + u_{i,j+2}\right)\\
&- a_{i+1,j+1}c_{i+1,j+1}\left(u_{i,j} - u_{i+1,j} - u_{i,j+1} + 2u_{i+1,j+1} - u_{i+2,j+1} - u_{i+1,j+2} + u_{i+2,j+2}\right)\\
&- b_{i+1,j+1}^2\left(u_{i+1,j} - u_{i+2,j} + u_{i,j+1} - 2u_{i+1,j+1} + u_{i+2,j+1} - u_{i,j+2} + u_{i+1,j+2}\right)\\
&- b_{i,j-1}^2\left(u_{i-1,j-2} - u_{i,j-2} - u_{i-1,j-1} + 2u_{i,j-1} - u_{i+1,j-1} - u_{i,j} + u_{i+1,j}\right)\\
&- a_{i,j-1}c_{i,j-1}\left(u_{i,j-2} - u_{i+1,j-2} + u_{i-1,j-1} - 2u_{i,j-1} + u_{i+1,j-1} - u_{i-1,j} + u_{i,j}\right)\\
&+ b_{i+1,j-1}^2\left(u_{i,j-2} - u_{i+1,j-2} - u_{i,j-1} + 2u_{i+1,j-1} - u_{i+2,j-1} - u_{i+1,j} + u_{i+2,j}\right)\\
&+ a_{i+1,j-1}c_{i+1,j-1}\left(u_{i+1,j-2} - u_{i+2,j-2} + u_{i,j-1} - 2u_{i+1,j-1} + u_{i+2,j-1} - u_{i,j} + u_{i+1,j}\right)\\
&- b_{i-1,j}^2\left(u_{i-2,j-1} - u_{i-1,j-1} - u_{i-2,j} + 2u_{i-1,j} - u_{i,j} - u_{i-1,j+1} + u_{i,j+1}\right)\\
&- a_{i-1,j}c_{i-1,j}\left(u_{i-1,j-1} - u_{i,j-1} + u_{i-2,j} - 2u_{i-1,j} + u_{i,j} - u_{i-2,j+1} + u_{i-1,j+1}\right)\\
&- b_{i+1,j}^2\left(u_{i,j-1} - u_{i+1,j-1} - u_{i,j} + 2u_{i+1,j} - u_{i+2,j} - u_{i+1,j+1} + u_{i+2,j+1}\right)\\
&- a_{i+1,j}c_{i+1,j}\left(u_{i+1,j-1} - u_{i+2,j-1} + u_{i,j} - 2u_{i+1,j} + u_{i+2,j} - u_{i,j+1} + u_{i+1,j+1}\right)\\
&+ b_{i-1,j+1}^2\left(u_{i-2,j} - u_{i-1,j} - u_{i-2,j+1} + 2u_{i-1,j+1} - u_{i,j+1} - u_{i-1,j+2} + u_{i,j+2}\right)\\
&+ a_{i-1,j+1}c_{i-1,j+1}\left(u_{i-1,j} - u_{i,j} + u_{i-2,j+1} - 2u_{i-1,j+1} + u_{i,j+1} - u_{i-2,j+2} + u_{i-1,j+2}\right)\\
&- b_{i,j+1}^2\left(u_{i-1,j} - u_{i,j} - u_{i-1,j+1} + 2u_{i,j+1} - u_{i+1,j+1} - u_{i,j+2} + u_{i+1,j+2}\right)\\
&- a_{i,j+1}c_{i,j+1}\left(u_{i,j} - u_{i+1,j} + u_{i-1,j+1} - 2u_{i,j+1} + u_{i+1,j+1} - u_{i-1,j+2} + u_{i,j+2}\right)\\
&- 2a_{i,j}c_{i,j}\left(u_{i-1,j-1} - u_{i,j-1} - u_{i-1,j} + 2u_{i,j} - u_{i+1,j} - u_{i,j+1} + u_{i+1,j+1}\right)\\
&+ 2a_{i,j}c_{i,j}\left(u_{i,j-1} - u_{i+1,j-1} + u_{i-1,j} - 2u_{i,j} + u_{i+1,j} - u_{i-1,j+1} + u_{i,j+1}\right)
\end{aligned}
$$

$$
\begin{aligned}
&\quad - 2b_{i,j}^2 \left(u_{i,j-1} - u_{i+1,j-1} + u_{i-1,j} - 2u_{i,j} + u_{i+1,j} - u_{i-1,j+1} + u_{i,j+1}\right) \\
&\quad + 2b_{i,j}^2 \left(u_{i-1,j-1} - u_{i,j-1} - u_{i-1,j} + 2u_{i,j} - u_{i+1,j} - u_{i,j+1} + u_{i+1,j+1}\right)\Big) \\
&+ \frac{1}{2h_x h_y^3}\Big(-b_{i-1,j-1}c_{i-1,j-1}\left(u_{i-1,j-2} - 2u_{i-1,j-1} + u_{i-1,j}\right) \\
&\quad + b_{i,j-1}c_{i,j-1}\left(u_{i,j-2} - 2u_{i,j-1} + u_{i,j}\right) \\
&\quad + b_{i-1,j}c_{i-1,j}\left(u_{i-1,j-1} - 2u_{i-1,j} + u_{i-1,j+1}\right) \\
&\quad + b_{i+1,j}c_{i+1,j}\left(u_{i+1,j-1} - 2u_{i+1,j} + u_{i+1,j+1}\right) \\
&\quad + b_{i,j+1}c_{i,j+1}\left(u_{i,j} - 2u_{i,j+1} + u_{i,j+2}\right) \\
&\quad - b_{i+1,j+1}c_{i+1,j+1}\left(u_{i+1,j} - 2u_{i+1,j+1} + u_{i+1,j+2}\right) \\
&\quad - b_{i,j-1}c_{i,j-1}\left(u_{i,j-2} - 2u_{i,j-1} + u_{i,j}\right) \\
&\quad + b_{i+1,j-1}c_{i+1,j-1}\left(u_{i+1,j-2} - 2u_{i+1,j-1} + u_{i+1,j}\right) \\
&\quad - b_{i-1,j}c_{i-1,j}\left(u_{i-1,j-1} - 2u_{i-1,j} + u_{i-1,j+1}\right) \\
&\quad - b_{i+1,j}c_{i+1,j}\left(u_{i+1,j-1} - 2u_{i+1,j} + u_{i+1,j+1}\right) \\
&\quad + b_{i-1,j+1}c_{i-1,j+1}\left(u_{i-1,j} - 2u_{i-1,j+1} + u_{i-1,j+2}\right) \\
&\quad - b_{i,j+1}c_{i,j+1}\left(u_{i,j} - 2u_{i,j+1} + u_{i,j+2}\right) \\
&\quad - 2b_{i,j}c_{i,j}\left(u_{i,j-1} - 2u_{i,j} + u_{i,j+1}\right) \\
&\quad + 2b_{i,j}c_{i,j}\left(u_{i,j-1} - 2u_{i,j} + u_{i,j+1}\right)\Big) \\
&+ \frac{1}{4h_x h_y^3}\Big(-b_{i,j+1}c_{i,j+1}\left(u_{i-1,j} - u_{i,j} - u_{i-1,j+1} + 2u_{i,j+1} - u_{i+1,j+1} - u_{i,j+2} + u_{i+1,j+2}\right) \\
&\quad - b_{i,j+1}c_{i,j+1}\left(u_{i,j} - u_{i+1,j} + u_{i-1,j+1} - 2u_{i,j+1} + u_{i+1,j+1} - u_{i-1,j+2} + u_{i,j+2}\right) \\
&\quad + 2b_{i,j}c_{i,j}\left(u_{i-1,j-1} - u_{i,j-1} - u_{i-1,j} + 2u_{i,j} - u_{i+1,j} - u_{i,j+1} + u_{i+1,j+1}\right) \\
&\quad + 2b_{i,j}c_{i,j}\left(u_{i,j-1} - u_{i+1,j-1} + u_{i-1,j} - 2u_{i,j} + u_{i+1,j} - u_{i-1,j+1} + u_{i,j+1}\right) \\
&\quad - b_{i,j-1}c_{i,j-1}\left(u_{i-1,j-2} - u_{i,j-2} - u_{i-1,j-1} + 2u_{i,j-1} - u_{i+1,j-1} - u_{i,j} + u_{i+1,j}\right) \\
&\quad - b_{i,j-1}c_{i,j-1}\left(u_{i,j-2} - u_{i+1,j-2} + u_{i-1,j-1} - 2u_{i,j-1} + u_{i+1,j-1} - u_{i-1,j} + u_{i,j}\right)\Big) \\
&+ \frac{1}{h_y^4}\Big(-c_{i,j+1}^2\left(u_{i,j} - 2u_{i,j+1} + u_{i,j+2}\right) \\
&\quad + 2c_{i,j}^2\left(u_{i,j-1} - 2u_{i,j} + u_{i,j+1}\right) \\
&\quad - c_{i,j-1}^2\left(u_{i,j-2} - 2u_{i,j-1} + u_{i,j}\right)\Big)
\end{aligned}
$$

$$
\begin{aligned}
&= \frac{1}{h_x^4}\Big(-a_{i+1,j}^2\left(u_{i,j} - 2u_{i+1,j} + u_{i+2,j}\right) \\
&\quad + 2a_{i,j}^2\left(u_{i-1,j} - 2u_{i,j} + u_{i+1,j}\right) \\
&\quad - a_{i-1,j}^2\left(u_{i-2,j} - 2u_{i-1,j} + u_{i,j}\right)\Big) \\
&+ \frac{1}{4h_x^3 h_y}\Big(-a_{i+1,j}b_{i+1,j}\left(u_{i,j-1} - u_{i+2,j-1} - u_{i,j+1} + u_{i+2,j+1}\right) \\
&\quad + 2a_{i,j}b_{i,j}\left(u_{i-1,j-1} - u_{i+1,j-1} - u_{i-1,j+1} + u_{i+1,j+1}\right) \\
&\quad - a_{i-1,j}b_{i-1,j}\left(u_{i-2,j-1} - u_{i,j-1} - u_{i-2,j+1} + u_{i,j+1}\right) \\
&\quad - 2a_{i-1,j-1}b_{i-1,j-1}\left(u_{i-2,j-1} - 2u_{i-1,j-1} + u_{i,j-1}\right) \\
&\quad + 2a_{i+1,j-1}b_{i+1,j-1}\left(u_{i,j-1} - 2u_{i+1,j-1} + u_{i+2,j-1}\right) \\
&\quad + 2a_{i-1,j+1}b_{i-1,j+1}\left(u_{i-2,j+1} - 2u_{i-1,j+1} + u_{i,j+1}\right)
\end{aligned}
$$

$$
\begin{aligned}
&\quad - 2a_{i+1,j+1}b_{i+1,j+1}\left(u_{i,j+1} - 2u_{i+1,j+1} + u_{i+2,j+1}\right)\Big) \\
&+\frac{1}{4h_x^2h_y^2}\Big(-a_{i-1,j-1}c_{i-1,j-1}\left(u_{i-2,j-2} - u_{i-1,j-2} - u_{i-2,j-1} + 2u_{i-1,j-1} - u_{i,j-1} - u_{i-1,j} + u_{i,j}\right) \\
&\quad + a_{i-1,j+1}c_{i-1,j+1}\left(u_{i-1,j} - u_{i,j} + u_{i-2,j+1} - 2u_{i-1,j+1} + u_{i,j+1} - u_{i-2,j+2} + u_{i-1,j+2}\right) \\
&\quad + a_{i+1,j-1}c_{i+1,j-1}\left(u_{i+1,j-2} - u_{i+2,j-2} + u_{i,j-1} - 2u_{i+1,j-1} + u_{i+2,j-1} - u_{i,j} + u_{i+1,j}\right) \\
&\quad - a_{i+1,j+1}c_{i+1,j+1}\left(u_{i,j} - u_{i+1,j} - u_{i,j+1} + 2u_{i+1,j+1} - u_{i+2,j+1} - u_{i+1,j+2} + u_{i+2,j+2}\right) \\
&\quad + a_{i,j-1}c_{i,j-1}\left(u_{i-1,j-2} - 2u_{i,j-2} + u_{i+1,j-2} - 2u_{i-1,j-1} + 4u_{i,j-1} - 2u_{i+1,j-1} + u_{i-1,j} - 2u_{i,j} + u_{i+1,j}\right) \\
&\quad + a_{i-1,j}c_{i-1,j}\left(u_{i-2,j-1} - 2u_{i-1,j-1} + u_{i,j-1} - 2u_{i-2,j} + 4u_{i-1,j} - 2u_{i,j} + u_{i-2,j+1} - 2u_{i-1,j+1} + u_{i,j+1}\right) \\
&\quad + a_{i,j+1}c_{i,j+1}\left(u_{i-1,j} - 2u_{i,j} + u_{i+1,j} - 2u_{i-1,j+1} + 4u_{i,j+1} - 2u_{i+1,j+1} + u_{i-1,j+2} - 2u_{i,j+2} + u_{i+1,j+2}\right) \\
&\quad + a_{i+1,j}c_{i+1,j}\left(u_{i,j-1} - 2u_{i+1,j-1} + u_{i+2,j-1} - 2u_{i,j} + 4u_{i+1,j} - 2u_{i+2,j} + u_{i,j+1} - 2u_{i+1,j+1} + u_{i+2,j+1}\right) \\
&\quad + 2a_{i,j}c_{i,j}\left(-u_{i-1,j-1} + 2u_{i,j-1} - u_{i+1,j-1} + 2u_{i-1,j} - 4u_{i,j} + 2u_{i+1,j} - u_{i-1,j+1} + 2u_{i,j+1} - u_{i+1,j+1}\right) \\
&\quad + 2b_{i,j}^2\left(u_{i-1,j-1} - 2u_{i,j-1} + u_{i+1,j-1} - 2u_{i-1,j} + 4u_{i,j} - 2u_{i+1,j} + u_{i-1,j+1} - 2u_{i,j+1} + u_{i+1,j+1}\right) \\
&\quad - b_{i-1,j-1}^2\left(u_{i-1,j-2} - u_{i,j-2} + u_{i-2,j-1} - 2u_{i-1,j-1} + u_{i,j-1} - u_{i-2,j} + u_{i-1,j}\right) \\
&\quad - b_{i+1,j+1}^2\left(u_{i+1,j} - u_{i+2,j} + u_{i,j+1} - 2u_{i+1,j+1} + u_{i+2,j+1} - u_{i,j+2} + u_{i+1,j+2}\right) \\
&\quad + b_{i+1,j-1}^2\left(u_{i,j-2} - u_{i+1,j-2} - u_{i,j-1} + 2u_{i+1,j-1} - u_{i+2,j-1} - u_{i+1,j} + u_{i+2,j}\right) \\
&\quad + b_{i-1,j+1}^2\left(u_{i-2,j} - u_{i-1,j} - u_{i-2,j+1} + 2u_{i-1,j+1} - u_{i,j+1} - u_{i-1,j+2} + u_{i,j+2}\right) \\
&\quad + b_{i-1,j}^2\left(-u_{i-2,j-1} + 2u_{i-1,j-1} - u_{i,j-1} + 2u_{i-2,j} - 4u_{i-1,j} + 2u_{i,j} - u_{i-2,j+1} + 2u_{i-1,j+1} - u_{i,j+1}\right) \\
&\quad + b_{i+1,j}^2\left(-u_{i,j-1} + 2u_{i+1,j-1} - u_{i+2,j-1} + 2u_{i,j} - 4u_{i+1,j} + 2u_{i+2,j} - u_{i,j+1} + 2u_{i+1,j+1} - u_{i+2,j+1}\right) \\
&\quad + b_{i,j+1}^2\left(-u_{i-1,j} + 2u_{i,j} - u_{i+1,j} + 2u_{i-1,j+1} - 4u_{i,j+1} + 2u_{i+1,j+1} - u_{i-1,j+2} + 2u_{i,j+2} - u_{i+1,j+2}\right) \\
&\quad + b_{i,j-1}^2\left(-u_{i-1,j-2} + 2u_{i,j-2} - u_{i+1,j-2} + 2u_{i-1,j-1} - 4u_{i,j-1} + 2u_{i+1,j-1} - u_{i-1,j} + 2u_{i,j} - u_{i+1,j}\right) \\
&\quad - 4b_{i,j+1}^2\left(u_{i-1,j+1} - 2u_{i,j+1} + u_{i+1,j+1}\right) \\
&\quad - 4b_{i,j-1}^2\left(u_{i-1,j-1} - 2u_{i,j-1} + u_{i+1,j-1}\right) \\
&\quad + 8b_{i,j}^2\left(u_{i-1,j} - 2u_{i,j} + u_{i+1,j}\right) \\
&\quad + 8b_{i,j}^2\left(u_{i,j-1} - 2u_{i,j} + u_{i,j+1}\right) \\
&\quad - 4b_{i+1,j}^2\left(u_{i+1,j-1} - 2u_{i+1,j} + u_{i+1,j+1}\right) \\
&\quad - 4b_{i-1,j}^2\left(u_{i-1,j-1} - 2u_{i-1,j} + u_{i-1,j+1}\right)\Big) \\
&+\frac{1}{4h_xh_y^3}\Big(-b_{i,j+1}c_{i,j+1}\left(u_{i-1,j} - u_{i+1,j} - u_{i-1,j+2} + u_{i+1,j+2}\right) \\
&\quad + 2b_{i,j}c_{i,j}\left(u_{i-1,j-1} - u_{i+1,j-1} - u_{i-1,j+1} + u_{i+1,j+1}\right) \\
&\quad - b_{i,j-1}c_{i,j-1}\left(u_{i-1,j-2} - u_{i+1,j-2} - u_{i-1,j} + u_{i+1,j}\right) \\
&\quad - 2b_{i-1,j-1}c_{i-1,j-1}\left(u_{i-1,j-2} - 2u_{i-1,j-1} + u_{i-1,j}\right) \\
&\quad + 2b_{i+1,j-1}c_{i+1,j-1}\left(u_{i+1,j-2} - 2u_{i+1,j-1} + u_{i+1,j}\right) \\
&\quad + 2b_{i-1,j+1}c_{i-1,j+1}\left(u_{i-1,j} - 2u_{i-1,j+1} + u_{i-1,j+2}\right) \\
&\quad - 2b_{i+1,j+1}c_{i+1,j+1}\left(u_{i+1,j} - 2u_{i+1,j+1} + u_{i+1,j+2}\right)\Big) \\
&+\frac{1}{h_y^4}\Big(-c_{i,j+1}^2\left(u_{i,j} - 2u_{i,j+1} + u_{i,j+2}\right) \\
&\quad + 2c_{i,j}^2\left(u_{i,j-1} - 2u_{i,j} + u_{i,j+1}\right) \\
&\quad - c_{i,j-1}^2\left(u_{i,j-2} - 2u_{i,j-1} + u_{i,j}\right)\Big)
\end{aligned}
$$

$$\frac{1}{4}\cdot\mathcal{U}$$

	$i-2$	$i-1$	i	$i+1$	$i+2$
$j-2$	$-(ac)_{i-1,j-1}$	$-b^2_{i-1,j-1} - b^2_{i,j-1} + (ac)_{i,j-1} + (ac)_{i-1,j-1} - 2(bc)_{i-1,j-1} - (bc)_{i,j-1}$	$b^2_{i-1,j-1} + b^2_{i,j-1} - 2(ac)_{i,j-1} + 2b^2_{i,j-1} - 4c^2_{i,j-1} - 2(ab)_{i-1,j-1}$	$-b^2_{i+1,j-1} - b^2_{i,j-1} + (ac)_{i+1,j-1} + (ac)_{i,j-1} + 2(bc)_{i+1,j-1} + (bc)_{i,j-1}$	$-(ac)_{i+1,j-1}$
$j-1$	$-2(ab)_{i-1,j-1} - (ab)_{i-1,j-1} - b^2_{i-1,j-1} + (ac)_{i-1,j-1} + b^2_{i-1,j} - 2b^2_{i-1,j-1} + 2(ac)_{i-1,j-1} + (ac)_{i-1,j} - 2(ac)_{i,j-1} + 2(ac)_{i-1,j} - 2(ac)_{i,j} + 4(bc)_{i-1,j-1} + 2(bc)_{i,j}$	$4(ab)_{i-1,j-1} + 2(ab)_{i,j} - 2b^2_{i-1,j-1} - 2b^2_{i,j-1} + 2b^2_{i,j} + 2(ac)_{i-1,j-1} - 2(ac)_{i,j-1} - 2(ac)_{i-1,j} + 2(ac)_{i,j} + 4(bc)_{i-1,j-1} - 2(bc)_{i,j}$	$-2(ab)_{i-1,j-1} + (ab)_{i-1,j} + (ab)_{i+1,j-1} + (ab)_{i,j} - b^2_{i-1,j-1} - b^2_{i+1,j-1} - 4b^2_{i,j-1} + b^2_{i-1,j} - b^2_{i+1,j} + 4b^2_{i,j} - b^2_{i-1,j} - b^2_{i+1,j} + (ac)_{i-1,j-1} + 4(ac)_{i,j-1} + (ac)_{i-1,j} + (ac)_{i+1,j-1} + (ac)_{i,j} + 8c^2_{i,j-1} + 8c^2_{i,j}$	$-4(ab)_{i+1,j-1} - 2(ab)_{i,j} + 2b^2_{i+1,j-1} - 2b^2_{i,j-1} - 2b^2_{i+1,j} + 2b^2_{i,j} + 2(ac)_{i+1,j-1} - 2(ac)_{i,j-1} - 2(ac)_{i+1,j} + 2(ac)_{i,j} + 4(bc)_{i+1,j-1} - 2(bc)_{i,j}$	$2(ab)_{i+1,j-1} + (ab)_{i+1,j} - b^2_{i+1,j-1} - b^2_{i+1,j} + (ac)_{i+1,j-1} + (ac)_{i+1,j}$
j	$-4a^2_{i-1,j} + b^2_{i-1,j-1} + b^2_{i-1,j+1} + 2(ac)_{i-1,j}$	$8a^2_{i-1,j} + 8a^2_{i,j} - b^2_{i-1,j-1} - b^2_{i,j-1} + 4b^2_{i-1,j} + 4b^2_{i,j} - b^2_{i-1,j+1} - b^2_{i,j+1} + (ac)_{i-1,j-1} + 4(ac)_{i-1,j} + (ac)_{i,j-1} + (ac)_{i-1,j+1} + (ac)_{i,j+1} + 4(ac)_{i,j} + 2(bc)_{i-1,j-1} + (bc)_{i,j+1}$	$-4a^2_{i-1,j} - 16a^2_{i,j} + 4a^2_{i+1,j} + 2b^2_{i,j-1} + 2b^2_{i-1,j} - 24b^2_{i,j} + 2b^2_{i,j+1} + 2b^2_{i+1,j} - (ac)_{i-1,j-1} + 2(ac)_{i,j-1} - (ac)_{i+1,j-1} - 2(ac)_{i-1,j} - 8(ac)_{i,j} + 2(ac)_{i+1,j} - (ac)_{i-1,j+1} + 2(ac)_{i,j+1} + (ac)_{i+1,j+1} - 4c^2_{i-1,j} - 16c^2_{i,j} - 4c^2_{i+1,j}$	$8a^2_{i+1,j} + 8a^2_{i,j} - b^2_{i+1,j-1} - b^2_{i,j-1} + 4b^2_{i+1,j} + 4b^2_{i,j} - b^2_{i+1,j+1} - b^2_{i,j+1} + (ac)_{i+1,j-1} + 4(ac)_{i,j-1} + (ac)_{i+1,j} + (ac)_{i,j+1} + 2(bc)_{i,j-1} - 2(bc)_{i+1,j+1}$	$-4a^2_{i+1,j} + b^2_{i+1,j-1} + b^2_{i+1,j+1} + 2(ac)_{i+1,j}$
$j+1$	$2(ab)_{i-1,j+1} + (ab)_{i-1,j} - b^2_{i-1,j+1} - b^2_{i-1,j} + (ac)_{i-1,j+1} + (ac)_{i-1,j}$	$-4(ab)_{i-1,j+1} - 2(ab)_{i,j} + 2b^2_{i-1,j+1} - 2b^2_{i,j+1} + 2b^2_{i-1,j} - 2b^2_{i,j} + 2(ac)_{i-1,j+1} - 2(ac)_{i,j+1} + 2(ac)_{i-1,j} - 2(ac)_{i,j} + 4(bc)_{i-1,j-1} - 2(bc)_{i,j}$	$2(ab)_{i-1,j+1} - (ab)_{i-1,j} + (ab)_{i+1,j+1} - b^2_{i-1,j+1} + b^2_{i-1,j} + 4b^2_{i,j+1} - b^2_{i+1,j+1} - b^2_{i+1,j} + (ac)_{i-1,j+1} + 4(ac)_{i,j+1} + (ac)_{i+1,j+1} + (ac)_{i-1,j} + (ac)_{i+1,j} + 8c^2_{i,j+1} + 8c^2_{i,j}$	$4(ab)_{i+1,j+1} + 2(ab)_{i,j} + 2b^2_{i+1,j+1} - 2b^2_{i,j+1} + 2b^2_{i+1,j} - 2b^2_{i,j} + 2(ac)_{i+1,j+1} - 2(ac)_{i,j+1} + 2(ac)_{i+1,j} - 2(ac)_{i,j} + 4(bc)_{i+1,j+1} + 2(bc)_{i,j}$	$-2(ab)_{i+1,j+1} - (ab)_{i+1,j} - b^2_{i+1,j+1} - b^2_{i+1,j} + (ac)_{i+1,j+1} + (ac)_{i+1,j}$
$j+2$	$-(ac)_{i-1,j+1}$	$-b^2_{i-1,j+1} - b^2_{i,j+1} + (ac)_{i-1,j+1} + (ac)_{i,j+1} + 2(bc)_{i-1,j+1} + (bc)_{i,j+1}$	$b^2_{i-1,j+1} + b^2_{i,j+1} - 2(ac)_{i,j+1} + 4c^2_{i,j+1}$	$-b^2_{i+1,j+1} - b^2_{i,j+1} + (ac)_{i+1,j+1} + (ac)_{i,j+1} + 2(bc)_{i+1,j+1} + (bc)_{i,j+1}$	$-(ac)_{i+1,j+1}$

FIGURE A.1: Simplified stencil for the discretization of the anisotropic Frobenius model ($h_x = h_y = 1$)

A.2 Pseudocode I — Generic Evolutions

This pseudocode lists the sequence of steps of a generic implementation for diffusion-like evolutions. It is designed such that isotropic, anisotropic methods as well as all their parameters can be given to one binary that can perform all of those methods using modular support functions. This makes everything transparent and maintainable, especially what method is run with what parameters during experiments, which can be run in batch mode easily and output consistent (therefore parseable) debug data into resulting images (e.g. into PGM header).

Listing A.1: Pseudocode for generic evolution processes.

```
// current iter is u, old iter is f
DIFFUSE(u,f) {
  switch(diffusion_method) {
  case *:
    switch(discretization) {
    case *:
      for(i,j)
        u[i][j] = ... +f[.][.]+f[.][.]+ ...
        // using either g[.], or a[.], b[.] and c[.]
    }
  }
}

MAIN() {
  // request input data and parameters here

  for(k=0; k<=number_of_iterations; k++) {
    PRESMOOTH(u) // Gaussian convolution
    DUMMIES(u) // boundary conditions
    // compute diffusivities, needed in any case
    for(i,j) {
      switch(diffusivity) {
      case *:
        switch(diffusivity_argument) {
        case *:
          s = ... u[.] ...
        }
        g[i][j] = ... 1/s ...
      }
    }

    // compute diffusion tensor, if necessary
    if (ANISOTROPIC) {
      switch(structure_tensor) {
```

```
35        case *:
36          for(i,j) {
37            a[i][j] = ... u[.] ...
38            b[i][j] = ... u[.] ...
39            c[i][j] = ... u[.] ...
40          }
41        }
42        for(i,j) {
43          DIFFUSION_TENSOR(i,j) {
44            PAT_TRANS()
45            eigenvalue1 = ... g[.] ...
46            eigenvalue2 = 1
47            PAT_BACKTRANS()
48          }
49        }
50      }
51
52      ORIGINAL(u) // restore unsmoothed u
53      if (do_inpaint) {
54        COPY_A2B(u,v) // copies matrix u into v
55        DIFFUSE(v)
56        /* where the mask is black, let it evolve
57         * otherwise (mask is white), keep original data */
58        for (i,j)
59          if (mask[i][j]==0)
60            u[i][j] = v[i][j]
61       } else {
62         DIFFUSE(u);
63       }
64
65    } // end for(k)
66  }
```

A.3 Pseudocode II — Stencil Algebra

This section shows code for applying stencils, respectively an image and a stencil, to each other as mentioned in Section 2.4.4. Applying two stencils to each other yields a new stencil and can be useful to quickly find discretizations of combinations of discretized derivatives that are already present in stencil notation, as we have e.g. done in Section 3.2.2 (page 40). Applying a stencil to (consecutively all pixels of) an image yields a new image with updated pixel values, exactly as it happens with evolution equations in a computational sense, where they are an iterative process. Therefore, how the resulting array's data is to be interpreted, depends on what is given as input.

Listing A.2 is an excerpt of the most important things from a working implementation in C (we have omitted memory allocation and output routines). The arrays `A` and `B` represent the input data and the variables `[AB]s[xy]` their dimensions (e.g. `Asx`: size of array `A` in `x`-direction). One also has to give the center element of the stencil (`c_x` and `c_y`), otherwise one does not exactly know how they overlap; in this thesis, we have always displayed it using a gray background. In the end, the nested `for` loops do nothing more than implement the sum from (2.37) with the appropriate boundary conditions. In the implementation, this is especially important since one cannot simply address an array at e.g. location -1. Finally, the array `R` contains the resulting set of values.

Example output of the complete application follows; here we compute the approximation of $\partial_{yx}\left(\partial_{yx}(u)\right)$ using the stencil already presented on page 47.

```
Asx=3, Asy=3, Bsx=3, Bsy=3 ... c_x=1, c_y=1
=> Rsx=5, Rsy=5

/-----------------------\
|     0 |     1 |    -1 |
|-----------------------|
|     1 |    -2 |     1 |
|-----------------------|
|    -1 |     1 |     0 |
\-----------------------/
applied to
/-----------------------\
|     0 |     1 |    -1 |
|-----------------------|
|     1 |    -2 |     1 |
|-----------------------|
|    -1 |     1 |     0 |
\-----------------------/
results in
/---------------------------------------\
|     0 |     0 |     1 |    -2 |     1 |
|---------------------------------------|
|     0 |     2 |    -6 |     6 |    -2 |
|---------------------------------------|
|     1 |    -6 |    10 |    -6 |     1 |
|---------------------------------------|
|    -2 |     6 |    -6 |     2 |     0 |
|---------------------------------------|
|     1 |    -2 |     1 |     0 |     0 |
\---------------------------------------/

Sum of absolute values except center, useful for Gershgorin: 54
```

Listing A.2: Implementation of stencil algebra.

```
int main () {

  /* requires:
   * arrays (stencils): A, B (float)
   * dimensions of the arrays: Asx, Asy, Bsx, Bsy (int)
   * the center pixel (zero-based): c_x, c_y (int)
   * memory allocation for matrices A,B,R!
   */

  float **A, **B, **R;

  const int Rsx = Asx+Bsx-1;
  const int Rsy = Asy+Bsy-1;

  int i,j,k,l, Ax,Ay;
  float sum;
  for (i=0;i<Rsx;i++)
    for (j=0;j<Rsy;j++) {
      sum = 0;
      for (k=0;k<Bsx;k++)
        for (l=0;l<Bsy;l++) {
          Ax = i+k-2*c_x;
          Ay = j+l-2*c_y;
          if (!(Ax<0 || Ax>=Asx || Ay<0 || Ay>=Asy))
            sum += A[Ax][Ay]*B[k][l];
        }
      R[i][j] = sum;
    }

  stencil_out(B, Bsx, Bsy, 5);
  printf("applied to\n");
  stencil_out(A, Asx, Bsy, 5);
  printf("results in\n");
  stencil_out(R, Rsx, Rsy, 5);

  int gsum=0; // sum for Gershgorin RHS
  const int c_Rx = (int)(Rsx/2);
  const int c_Ry = (int)(Rsy/2);
  for (i=0;i<Rsx;i++)
    for (j=0;j<Rsy;j++)
      if (!(i==c_Rx && j==c_Ry))
        gsum+=abs(R[i][j]);

  return(0);
}
```

Bibliography

[AGLM93] L. Alvarez, F. Guichard, P. L. Lions and J. M. Morel. Axioms and fundamental equations of image processing. *Archive of Rational Mechanics*, 123, 1993.

[BDFW07] B. Burgeth, S. Didas, L. Florack and J. Weickert. A generic approach to diffusion filtering of matrix-fields. *Computing*, 81(2-3):pages 179–197, 2007.

[Boo78] C. de Boor. *A Practical Guide to Splines*. Springer-Verlag Berlin and Heidelberg GmbH & Co. KG, December 1978.

[Bru06] A. Bruhn. *Variational Optic Flow Computation – Accurate Modelling and Efficient Numerics*. Ph.D. thesis, Department of Mathematics and Comptuer Science, Saarland University, Germany, July 2006.

[BSCB00] M. Bertalmio, G. Sapiro, V. Caselles and C. Ballester. Image inpainting. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 417–424. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 2000.

[Buh03] M. D. Buhmann. *Radial Basis Functions*. Cambridge University Press, New York, NY, USA, 2003.

[CBFAB97] P. Charbonnier, L. Blanc-Féraud, G. Aubert and M. Barlaud. Deterministic edge-preserving regularization in computed imaging. *IEEE Transactions on Image Processing*, 6:pages 298–311, 1997.

[CK07] Y. Cha and S. Kim. PDE-Based Interpolation Methods for Image Super Resolution. In *FGCN '07: Proceedings of the Future Generation Communication and Networking (FGCN 2007)*, pages 214–219. IEEE Computer Society, Washington, DC, USA, 2007.

[CLMC92] F. Catte, P. L. Lions, J. M. Morel and T. Coll. Image selective smoothing and edge detection by nonlinear diffusion. *SIAM Journal on Numerical Analysis*, 29(1):pages 182–193, 1992.

[CMM00] T. Chan, A. Marquina and P. Mulet. High-Order Total Variation-Based Image Restoration. *SIAM Journal of Scientific Computation*, 22(2):pages 503–516, 2000.

[CMS98] V. Caselles, J.-M. Morel and C. Sbert. An axiomatic approach to image interpolation. *IEEE Transactions on Image Processing*, 7(3):pages 376–386, 1998.

[CZ98] R. Carmona and S. Zhong. Adaptive smoothing respecting feature directions. *IEEE Transactions on Image Processing*, 7(3):pages 353–358, March 1998.

[Did04] S. Didas. *Higher Order Variational Methods for Noise Removal in Signals and Images*. Diploma thesis, Department of Mathematics, Saarland University, Germany, April 2004.

[Did08] S. Didas. *Denoising and Enhancement of Digital Images – Variational Methods, Integrodifferential Equations, and Wavelets*. Ph.D. thesis, Department of Mathematics and Comptuer Science, Saarland University, Germany, 2008.

[DNV97] R. Distasi, M. Nappi and S. Vitulano. Image Compression by B-Tree Triangular Coding. *IEEE Transactions on Communications*, 45(9):pages 1095–1100, 1997.

[DWB05] S. Didas, J. Weickert and B. Burgeth. Stability and local feature enhancement of higher order nonlinear diffusion filtering. In W. Kropatsch, R. Sablatnig and A. Hanbury, editors, *Pattern Recognition*, volume 3663 of *Lecture Notes in Computer Science*, pages 251–258. Springer, Berlin, 2005.

[Eva98] L. C. Evans. *Partial Differential Equations*. American Mathematical Society, 1998.

[Far93] S. J. Farlow. *Partial Differential Equations for Scientists and Engineers*. Dover, 1993.

[FG87] W. Förstner and E. Gülch. A fast operator for detection and precise location of distinct points, corners and centres of circular features. In *Proceedings of the ISPRS Intercommission Conference on Fast Processing of Photogrammetric Data*, pages 281–305. Interlaken, Switzerland, June 1987.

[Fic55] A. Fick. On Liquid Diffusion. *Philosophical Magazine and Journal of Science*, 10:pages 31–39, 1855.

[FWBW06] C. Feddern, J. Weickert, B. Burgeth and M. Welk. Curvature-Driven PDE Methods for Matrix-Valued Images. *International Journal of Computer Vision*, 69(1):pages 93–107, 2006.

[GB04] J. B. Greer and A. L. Bertozzi. H^1 solutions of a class of fourth order nonlinear equations for image processing. In *Discrete and continuous dynamical systems*, volume 10, pages 349–366. 2004.

[GF00] I. M. Gelfand and S. V. Fomin. *Calculus of Variations*. Dover, New York, 2000.

[GKKJ92] G. Gerig, R. Kikinis, O. Kübler and F. Jolesz. Nonlinear Anisotropic Filtering of MRI Data. *IEEE Transactions on Medical Imaging*, 11(2):pages 221–232, June 1992.

[GWW+05] I. Galić, J. Weickert, M. Welk, A. Bruhn, A. Belyaev and H.-P. Seidel. Towards PDE-based image compression. In N. Paragios, O. Faugeras, T. Chan and C. Schnörr, editors, *Variational, Geometric and Level Set Methods in Computer Vision*, volume 3752 of *Lecture Notes in Computer Science*, pages 37–48. Springer, Berlin, 2005.

[GWW+08] I. Galić, J. Weickert, M. Welk, A. Bruhn, A. Belyaev and H.-P. Seidel. Image Compression with Anisotropic Diffusion. *Journal of Mathematical Imaging and Vision*, 31(2-3):pages 255–269, 2008.

[Har06] M. Hardy. Combinatorics of Partial Derivatives. *Electronic Journal of Combinatorics*, 13(1), 2006.

[HJ90] R. A. Horn and C. R. Johnson. *Matrix Analysis*. Cambridge University Press, 1990.

[Huf52] D. A. Huffmann. A Method for the Construction of Minimum-Redundancy Codes. *Proceedings of the Institute of Radio Engineers*, 40(9):pages 1098–1101, 1952.

[Iij59] T. Iijima. Basic theory of pattern observation. *Papers of Technical Group on Automata and Automatic Control*, December 1959. (in Japanese).

[Iij63] T. Iijima. Theory of Pattern Recognition. *Electronics and Communications in Japan*, pages 123–124, 1963.

[LLT03] M. Lysaker, A. Lundervold and X.-C. Tai. Noise removal using fourth-order partial differential equation with applications to medical magnetic resonance images in space and time. *IEEE Transactions on Image Processing*, 12:pages 1579–1590, 2003.

[LT06] M. Lysaker and X.-C. Tai. Iterative Image Restoration Combining Total Variation Minimization and a Second-Order Functional. *International Journal of Computer Vision*, 66(1):pages 5–18, 2006.

[Mei02] E. Meijering. A Chronology of Interpolation: From Ancient Astronomy to Modern Signal and Image Processing. In *Proceedings of the IEEE*, volume 90, pages 319–342. 2002.

[Met08] C. Metzner. *Extending the Method of Tumblin/Turk to Tensor-Valued Data*. Bachelor's Thesis, Department of Mathematics and Computer Science, Saarland University, Germany, October 2008.

[MM95] K. W. Morton and D. F. Mayers. *Numerical Solution of Partial Differential Equations*. Cambridge University Press, 1995.

[NFD97] M. Nielsen, L. Florack and R. Deriche. Regularization, scale-space, and edge detection filters. *Journal of Mathematical Imaging and Vision*, 7:pages 291–308, 1997.

[Phi05] J. Philibert. One and a Half Century of Diffusion: Fick, Einstein, before and beyond. *Diffusion Fundamentals*, 2(1):pages 1–10, 2005.

[PM90] P. Perona and J. Malik. Scale space and edge detection using anisotropic diffusion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12:pages 629–639, 1990.

[ROF92] L. I. Rudin, S. Osher and E. Fatemi. Nonlinear Total Variation Based Noise Removal Algorithms. *Physica D: Nonlinear Phenomena*, 60:pages 259–268, 1992.

[Rot08] J. M. Roth. *Stereo Correspondence for Slanted Surfaces: Local and Global Methods*. Bachelor's Thesis, Department of Mathematics and Computer Science, Saarland University, Germany, March 2008.

[RS91] A. R. Rao and B. G. Schunck. Computing oriented texture fields. *CVGIP: Graphical Model and Image Processing*, 53(2):pages 157–185, 1991.

[RSW99] E. Radmoser, O. Scherzer and J. Weickert. Scale-Space Properties of Regularization Methods. In *SCALE-SPACE '99: Proceedings of the Second International Conference on Scale-Space Theories in Computer Vision*, pages 211–222. Springer-Verlag, London, UK, 1999.

[SB02] J. Stoer and R. Bulirsch. *Introduction to Numerical Analysis*. Springer, 2002.

[Sch94] C. Schnörr. Unique Reconstruction of Piecewise Smooth Images by Minimizing Strictly Convex Non-Quadratic Functionals. *Journal of Mathematical Imaging and Vision*, 4:pages 189–198, 1994.

[SW00] O. Scherzer and J. Weickert. Relations between regularization and diffusion filtering. *Journal of Mathematical Imaging and Vision*, 12:pages 43–63, 2000.

[SWS09] T. Schultz, J. Weickert and H.-P. Seidel. A Higher-Order Structure Tensor. In D. H. Laidlaw and J. Weickert, editors, *Visualization and Processing of Tensor Fields*. Springer, 2009. (in print).

[Tik63] A. Tikhonov. Solution of incorrectly formulated problems and the regularization method. *Soviet Mathematical Doklady*, 4:pages 1035–1038, 1963.

[TT99] J. Tumblin and G. Turk. LCIS: A Boundary Hierarchy for Detail-Preserving Contrast Reduction. In A. Rockwood, editor, *SIGGRAPH 1999: Computer Graphics Proceedings*, pages 83–90. Addison Wesley Longman, Los Angeles, CA, USA, 1999.

[Tum99] J. J. E. Tumblin. *Three Methods of Detail-Preserving Contrast Reduction for Displayed Images*. Ph.D. thesis, College of Computing, Georgia Institute of Technology, Georgia, USA, December 1999.

[Wei98] J. Weickert. *Anisotropic Diffusion in Image Processing*. Teubner, Stuttgart, 1998.

[Wei99] G. W. Wei. Generalized Perona-Malik equation for image restoration. *IEEE Signal Processing Letters*, 6(7):pages 165–167, 1999.

[Whi23] E. T. Whittaker. On a New Method of Graduation. *Proceedings of the Edinborough Mathematical Society*, 41:pages 63–75, 1923.

[WII99] J. Weickert, S. Ishikawa and A. Imiya. Linear Scale-Space has First been Proposed in Japan. *Journal of Mathematical Imaging and Vision*, 10(3):pages 237–252, May 1999.

[Wit83] A. P. Witkin. Scale-Space Filtering. In *8th International Joint Conference on Artificial Intelligence*, volume 2, pages 1019–1022. August 1983.

[YK00] Y.-L. You and M. Kaveh. Fourth-order partial differential equations for noise removal. *IEEE Transactions on Image Processing*, 9(10):pages 1723–1730, 2000.

[Zim07] H. L. Zimmer. *PDE-based Image Compression using Corner Information*. Master's thesis, Department of Mathematics and Computer Science, Saarland University, Germany, September 2007.

List of Figures

List of Tables

Listings

Index