Contents

FAIR Listings

FAIR Examples

List of Figures

List of Tables

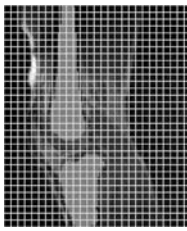
Preface

This book really shows how registration works: the flip-book appearing at the top right corners shows a registration of a human knee from bent to straight position (keeping bones rigid). Of course, the book also provides insight into concepts and practical tools. The presented framework exploits techniques from various fields such as image processing, numerical linear algebra, and optimization. Therefore, a brief overview of some preliminary literature in those fields is presented in the introduction (references [1–51]), and registration-specific literature is assembled at the end of the book (references [52–212]). Examples and results are based on the FAIR software, a package written in MATLAB. The FAIR software, as well as a PDF version of this entire book, can be freely downloaded from www.siam.org/books/fa06.

This book would not have been possible without the help of Bernd Fischer, Eldad Haber, Claudia Kremling, Jim Nagy, and the Safir Research Group from Lübeck: Sven Barendt, Björn Beuthin, Konstantin Ens, Stefan Heldmann, Sven Kabus, Janine Olesch, Nils Papenberg, Hanno Schumacher, and Stefan Wirtz. I'm also indebted to Sahar Alipour, Reza Heydarian, Raya Horesh, Ramin Mafi, and Bahram Marami Dizaji for improving the manuscript.

Thanks!

Jan Modersitzki
23. June 2009



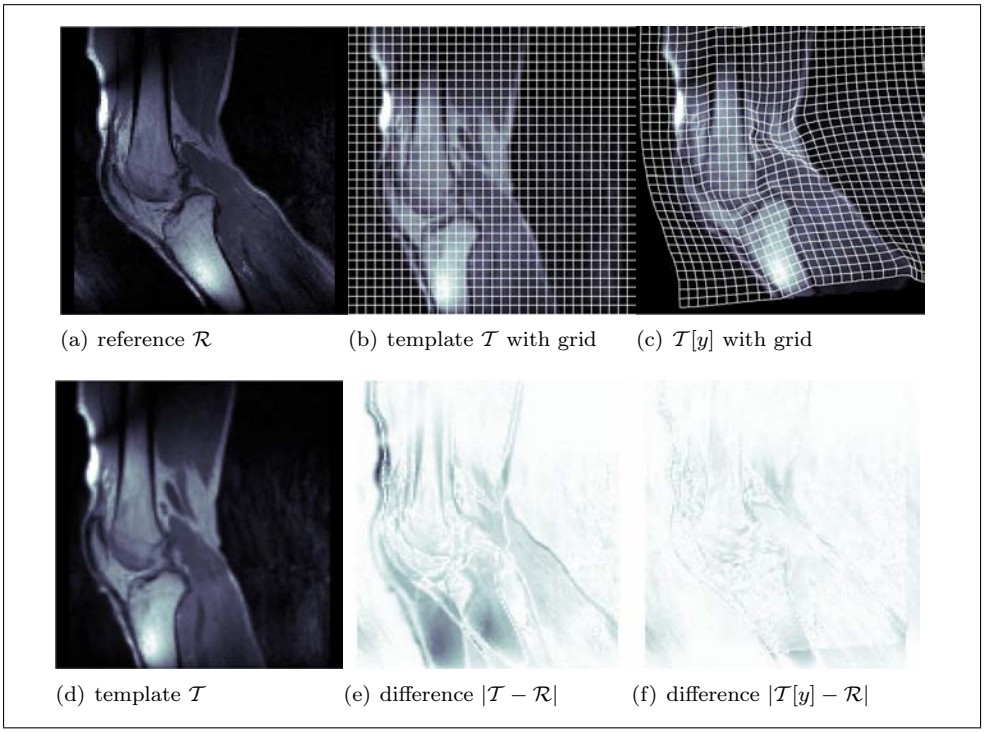
Chapter 1
Introduction

1.1 Image Registration

Image registration is one of the challenging problems in image processing. Given two images taken, for example, at different times, from different devices or perspectives, the goal is to determine a reasonable transformation, such that a transformed version of the first image is similar to the second one. A simplified registration example is illustrated in Figure 1.1. Given are two three-dimensional (3D) magnetic resonance scans of a human knee taken at two different times. The objective is to compensate image differences introduced by the different poses of the knee. Figure 1.1 also displays two corresponding slices of the image volumes: the template overlaid with a regular grid, and the difference before and after registration.

There is a large number of application areas which demand registration, including art, astronomy, astrophysics, biology, chemistry, criminology, genetics, physics, and basically any area involving imaging techniques. More specific examples include remote sensing (generating a global picture from different partial views), security (comparing current images with a data base), robotics (tracking of objects), and, in particular, medicine, where computational anatomy, computer-aided diagnosis, fusion of different modalities, intervention and treatment planning, monitoring of diseases, motion correction, radiation therapy, or treatment verification demand registration. Since imaging techniques, such as computer tomography (CT), magnetic resonance imaging (MRI), positron emission tomography (PET), single-photon emission computer tomography (SPECT), or ultrasound (US), have undergone remarkable, fascinating, and ongoing improvements, in the last decade, a tremendous increase in the utilization of the various modalities in medicine is taking place, and more is to be expected for the future.

Unfortunately, no unified treatment or general theory for image registration has been yet established. It appears that each application area has developed its own approaches and implementations. Depending on the particular application, the focus includes computing time (real-time applications in industrial inspection or tracking), image features (remote sensing), memory (high-resolution 3D images) and



(a) reference $\mathcal{R}$ (b) template $\mathcal{T}$ with grid (c) $\mathcal{T}[y]$ with grid

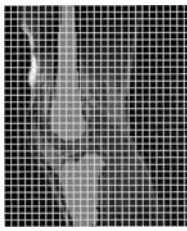
(d) template $\mathcal{T}$ (e) difference $|\mathcal{T} - \mathcal{R}|$ (f) difference $|\mathcal{T}[y] - \mathcal{R}|$

Figure 1.1: *Modified slices from CT scans of a human knee; image courtesy by Thomas Netsch, Philips Research Hamburg.*

accuracy of a model (medical treatment planing). Many of the solution strategies are designed by practitioners employing various heuristics for stabilization.

1.2 Scope and Aims of This Book

This book is about sound theoretical modeling, an efficient practical implementation, and a precise map connecting the two. Rather than taking an application-based perspective, this book aims to provide a unified approach to registration. For the modeling part, the book focuses on a variational approach because of its generality. It also allows the integration of many established techniques and enables a comparison and linking of different strategies. Moreover, the book limits itself to the so-called discretize-then-optimize approach, where the registration is viewed completely from an optimization perspective. For the algorithmic part, the book present, state-of-the-art numerical schemes. In contrast to the classical literature on digital image processing dealing with discrete objects and operations, pixels, and resolutions, this book deals with continuous objects and operations, families of discretization, and spatial domains. The book uses a toolbox of Flexible Algorithms for Image Registration (FAIR). FAIR collects state-of-the-art implementations of different building blocks which can then be combined in order to fit the specific de-



mands of particular applications. Its conceptual flexibility enables the integration of further features and the design of new fine-tuned schemes.

This book also aims to link *engineering* and *mathematics* to explain the *how* and *why* of image registration and to build a bridge between *image processing* and *numerics* as well as between *discrete* and *continuous* viewpoints. The book explains the *why* and provides tools for the *how* and thus strives to generate knowledge in *theory* and *practice*. The main goal of FAIR is to provide educational and research-oriented tools. It is not intended to provide software solutions for a clinical environment or the industrial standard. A new platform, an appropriate software engineering approach, smart C or C++ implementations, real-time code tuned for GPUs, appropriate visualization, theorems, proofs, or an exhaustive overview on the current literature are omitted. Instead, proper definitions of the modules and problems, explanation of basic concepts, examples and specifications, state-of-the-art implementations, and MATLAB® code enabling playing and exploration of options are provided.

1.3 Brief Outline

This section provides a brief overview of the topics presented in the following chapters.

Concepts Chapter 2 introduces the registration task as an optimization problem. The main ingredients such as images, transformations, distances, and regularizations are stated and briefly explained.

Interpolation An extended overview of interpolation techniques is outlined in Chapter 3. The chapter explains standard techniques such as d-linear and spline-based interpolation but also comments on multiscale and multilevel issues. The goal of interpolation is to provide a continuous image model, which is explored in the forthcoming chapters.

Transformations Parametric transformations are the topic of Chapter 4. A general model, including important classes such as rigid, affine, and spline-based transformations, is presented.

Landmarks Given the parametric transformation framework, simple landmark-based registration techniques are relatively easy to implement. Tools are presented in Chapter 5. Particular emphasis is given to the so-called thin-plate-spline registration.

Basic Optimization Tools Numerical schemes for the registration problem are discussed in Chapter 6, including discretization of integrals and transformations, numerical optimization, and multilevel strategies.

Distance Measures Chapter 7 introduces some of the common distance measures such as sum of squared differences (SSD), normalized cross correlation, mutual information, and normalized gradient fields. Particular emphasis is given to a proper discretization and solid derivatives.

Regularization Regularization is motivated, explained, and discussed in Chapter 8. Topics of interest are ill-posedness, L_2-norm-based regularizers such as elastic and curvature, discretization, and numerical implementations.

Nonparametric Image Registration Chapter 9 discusses the numerical schemes for solving the nonparametric image registration problem.

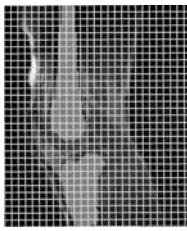
Outlook Chapter 10 addresses topics not covered in this book and scries the future of image registration.

1.4 Links to the Literature

The purpose of the following listing is to provide some starting points for an entry into areas related to this book. Note that this list is by no means complete or exhaustive. More specific references are provided in the following chapters.

1.4.1 (Medical) Image Registration

[1] L. G. Brown, *A survey of image registration techniques*, ACM Computing Surveys **24** (1992), 325–376.
[2] B. Fischer and J. Modersitzki, *Ill-posed medicine—an introduction to image registration*, Inverse Problems **24** (2008), 1–19.
[3] J. M. Fitzpatrick, D. L. G. Hill, and C. R. Maurer, Jr., *Image registration*, Handbook of Medical Imaging, Volume 2: Medical Image Processing and Analysis (M. Sonka and J. M. Fitzpatrick, eds.), SPIE, Bellingham, WA, 2000, pp. 447–513.
[4] C. Glasbey, *A review of image warping methods*, Journal of Applied Statistics **25** (1998), 155–171.
[5] A. A. Goshtasby, 2-*D and* 3-*D image registration*, Wiley, New York, 2005.
[6] J. Hajnal, D. Hawkes, and D. Hill, *Medical image registration*, CRC Press, Boca Raton, FL, 2001.
[7] R. Highnam and M. Brady, *Mammographic image analysis*, Kluwer Series on Medical Image Understanding, Kluwer Academic Publishers, New York, 1999.
[8] D. L. G. Hill, P. G. Batchelor, M. Holden, and D. J. Hawkes, *Medical image registration*, Physics in Medicine and Biology **46** (2001), R1–R45.
[9] H. Lester and R. Arridge, *A survey of hierarchical non-linear medical image registration*, Pattern Recognition **32** (1999), 129–149.
[10] J. B. A. Maintz and M. A. Viergever, *A survey of medical image registration*, Medical Image Analysis **2** (1998), 1–36.



[11] J. Modersitzki, *Numerical methods for image registration*, Oxford University Press, New York, 2004.
[12] J. P. W. Pluim, J. B. A. Maintz, and M. A. Viergever, *Mutual-information-based registration of medical images: A survey*, IEEE Transactions on Medical Imaging **22** (1999), 986–1004.
[13] K. Rohr, *Landmark-based image analysis*, Computational Imaging and Vision, Kluwer Academic Publishers, Dordrecht, 2001.
[14] O. Scherzer, *Mathematical models for registration and applications to medical imaging*, Springer, New York, 2006.
[15] A. Toga and P. Thompson, *The role of image registration in brain mapping*, Image and Vision Computing **19** (2001), 3–24.
[16] P. A. van den Elsen, E.-J. D. Pol, and M. A. Viergever, *Medical image matching—a review with classification*, IEEE Engineering in Medicine and Biology **12** (1993), 26–38.
[17] T. S. Yoo, *Insight into images: Principles and practice for segmentation, registration, and image analysis*, AK Peters Ltd., Wellesley, MA, 2004.
[18] B. Zitová and J. Flusser, *Image registration methods: A survey*, Image and Vision Computing **21** (2003), 977–1000.

1.4.2 Image Processing and Interpolation

[19] C. De Boor, *A practical guide to splines*, Springer, New York, 1978.
[20] T. F. Chan and J. Shen, *Image processing and analysis—variational, PDE, wavelet, and stochastic methods*, SIAM, Philadelphia, 2005.
[21] O. Faugeras, G. Aubert, and P. Kornprobst, *Mathematical problems in image processing*, Springer, New York, 2002.
[22] R. C. Gonzalez and R. E. Woods, *Digital image processing*, Addison-Wesley, Reading, PA, 1993.
[23] A. K. Jain, *Fundamentals of digital image processing*, Prentice–Hall, Englewood Cliffs, NJ, 1989.
[24] R. Kimmel, *Numerical geometry of images—theory, algorithms, and applications*, Springer, New York, 2004.
[25] L. Piegl and W. Tiller, *The NURBS book*, 2nd ed., Springer, New York, 1997.
[26] W. K. Pratt, *Digital image processing*, 4th ed., Wiley, Hoboken, NJ, 2007.
[27] M. Sonka, V. Hlavac, and R. Boyle, *Image processing, analysis, and machine vision*, 3rd ed., Thomson, Toronto, 2008.
[28] P. Thévenaz, T. Blu, and M. Unser, *Image interpolation and resampling*, in Handbook of Medical Imaging, Processing and Analysis (I. N. Bankman, ed.), Academic Press, San Diego, 2000, pp. 393–420.
[29] G. Wahba, *Spline models for observational data*, SIAM, Philadelphia, 1990.

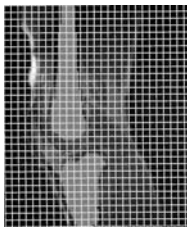
1.4.3 Numerics and Linear Algebra

[30] R. Barrett, M. Berry, T. F. Chan, J. W. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. van der Vorst, *Templates for the*

solution of linear systems: Building blocks for iterative methods, 2nd ed., Software, Environments, Tools, Vol. 43, SIAM, 1994.
[31] J. W. Brewer, *Kronecker products and matrix calculus in system theory*, IEEE Transactions on Circuits and Systems **25** (1978), 772–780.
[32] P. G. Ciarlet, *Introduction to numerical linear algebra and optimisation*, Cambridge University Press, Cambridge, 1989.
[33] B. Dacorogna, *Direct methods in the calculus of variations*, Springer, New York, 1989.
[34] P. J. Davis, *Circulant matrices*, Chelsea Publishing, New York, 1979.
[35] G. H. Golub and C. F. van Loan, *Matrix computations*, 3rd ed., The Johns Hopkins University Press, Baltimore, 2000.
[36] W. Hackbusch, *Iterative solution of large sparse systems of equations*, Springer, New York, 1993.
[37] D. Kahaner, C. Moler, and S. Nash, *Numerical methods and software*, Prentice-Hall, Englewood Cliffs, NJ, 1989.
[38] D. Kincaid and W. Cheney, *Numerical analysis*, 3rd ed., Brooks/Cole, Pacific Grove, CA, 2002.
[39] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical recipes in C*, 2nd ed., Cambridge University Press, Cambridge, 1992.
[40] M. Schatzman, *Numerical analysis: A mathematical introduction*, Clarendon Press, Oxford, 2002.

1.4.4 Partial Differential Equations and Optimization

[41] R. A. Adams, *Sobolev spaces*, Pure and Applied Mathematics Vol. 65, Academic Press, New York, 1975.
[42] J. E. Dennis and R. B. Schnabel, *Numerical methods for unconstrained optimization and nonlinear equations*, Classics in Applied Mathematics, Vol. 16, SIAM, Philadelphia, 1996.
[43] E. DiBenedetto, *Partial differential equations*, Birkhäuser, Boston, 1995.
[44] L. C. Evans, *Partial differential equations*, Graduate Studies in Mathematics, vol. 19, American Mathematical Society, Providence, RI, 1999.
[45] R. Fletcher, *Practical methods of optimization*, Wiley, Chichester, 1987.
[46] G. B. Folland, *Introduction to partial differential equations*, 2nd ed., Princeton University Press, Princeton, NJ, 1995.
[47] P. E. Gill, W. Murray, and M. H. Wright, *Practical optimization*, Academic Press, London, 1981.
[48] M. E. Gurtin, *An introduction to continuum mechanics*, Academic Press, Orlando, 1981.
[49] W. Hackbusch, *Partial differential equations*, Teubner, Stuttgart, 1987.
[50] C. T. Kelley, *Iterative methods for optimization*, Frontiers in Applied Mathematics, Vol. 18, SIAM, Philadelphia, 1999.
[51] J. Nocedal and S. J. Wright, *Numerical optimization*, Springer, New York, 1999.



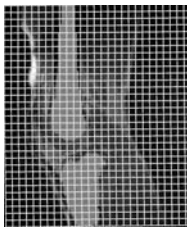
1.5 Further Links and Software

This book's website is www.siam.org/books/fa06.

MATLAB: Visit http://www.mathworks.com for tutorials, literature, and more.

Open Source Registration Software

- AIR: http://bishopw.loni.ucla.edu/AIR5/
- BIG: http://bigwww.epfl.ch/algorithms.html
- bUnwarpJ: http://biocomp.cnb.csic.es/~iarganda/bUnwarpJ/
- DROP: http://campar.in.tum.de/Main/Drop
- FAIR: http://www.cas.mcmaster.ca/~fair/index.shtml
- FLIRT: http://www.fmrib.ox.ac.uk/fsl/flirt/index.html
- Image Processing Toolbox: http://www.mathworks.com/products/image/
- IRT: http://www.eecs.umich.edu/~fessler/irt/
- ITK: http://www.itk.org/
- SAFIR-FLIRT: http://www.math.mu-luebeck.de/mitarbeiter/safir/software.shtml
- Slicer: http://www.slicer.org/



Chapter 2
FAIR Concepts

This chapter provides a brief overview on the topics to be discussed in the forthcoming chapters. The main ingredients of image registration and a formulation as an optimization problem are summarized in FAIR 1 (p.11). The functional or continuous setting and a solid numerical approach based on a nested sequence of discretizations are outlined. A few general comments on the MATLAB implementations are given.

2.1 FAIR Theory

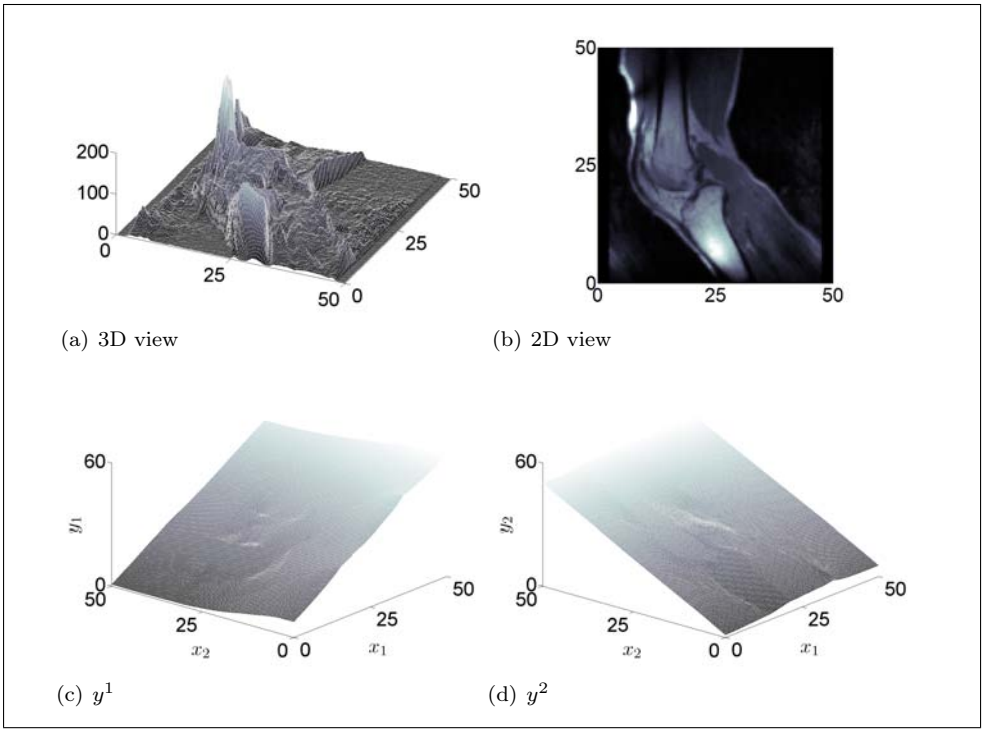
The registration problem may be phrased as follows. *Find a reasonable transformation such that a transformed version of a template image is similar to a reference image.* In this chapter, this task is formalized, and in the forthcoming chapters the ingredients are given life. The above formulation suggests an optimization framework

$$\mathcal{J}[y] = \mathcal{D}[\mathcal{T}[y], \mathcal{R}] + \alpha \mathcal{S}[y - y^{\text{ref}}] \xrightarrow{y} \min, \tag{2.1}$$

where $\mathcal{T}$ and $\mathcal{R}$ denote template and reference images, $\mathcal{T}[y]$ is the transformed template, $\mathcal{D}$ measures image similarity, and $\mathcal{S}$ measures reasonability of the transform. The notation used in this book is summarized in FAIR 1 (p.11).

An important FAIR feature is the use of a continuous setting; i.e., all variables are functions. Calligraphic characters such as $\mathcal{J}$ and $\mathcal{T}$ and square brackets are used for functionals. Figure 2.1 shows a visualization of an image and a transformation. A major advantage of the functional setting is that it enables fast and consistent multiscale and multilevel approaches; see Chapter 3.

There are three major reasons for using a continuous model. The first reason is philosophical. In general, the discrete image is a measurement of a continuous property of an object, for example, photon density. Therefore, modeling the object rather than the discrete measurement is an attractive option. The second reason is practical. Already a rather simple transformation such as a rotation of about 15 degrees requires a continuous image model since the transformed object does not align with a pixel grid; see Figure 2.2. Therefore, interpolation (in other words,



(a) 3D view (b) 2D view

(c) y^1 (d) y^2

Figure 2.1: *Visualizations of an image and a transformation* $y = (y^1(x^1, x^2), y^2(x^1, x^2))$.

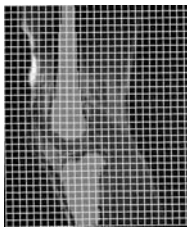
a continuous model) has to be used. Rather than mixing discrete and continuous viewpoints, a continuous model is used throughout the book from the very beginning. The third reason is efficiency. The numerical optimization schemes to be discussed exploit sequences on nested discretization of the very same functional; uppercase characters, such as J and T, and round brackets are used for functions. An approximate solution from a coarse level is refined on the finer level. This procedure is repeated until the finest level or the level with the desired accuracy is reached. Thus, on the expensive finest level, only a few correction steps are to be expected.

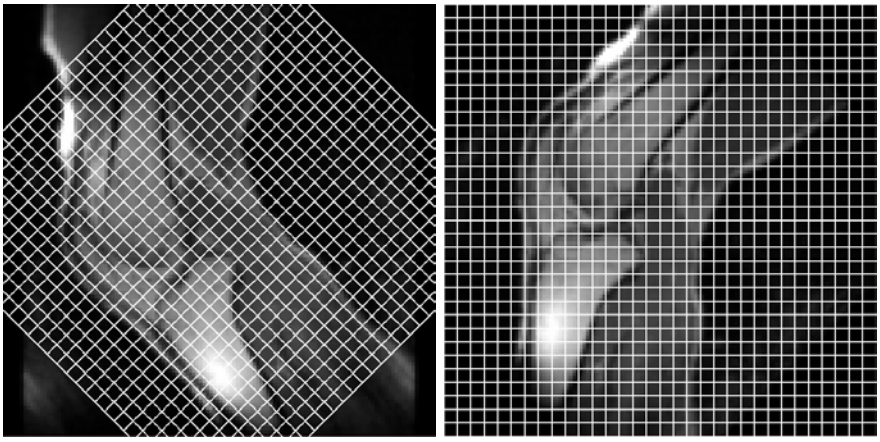
2.1.1 Images and Transformations

Images are considered as continuous mappings from a domain into the real numbers; see Chapter 3 for a detailed discussion. The domain is denoted by $\Omega \subset \mathbb{R}^d$, where d denotes the spatial dimensionality of the given data,

$$\mathcal{T} : \Omega \to \mathbb{R}, \quad \Omega \subset \mathbb{R}^d, \quad d = \text{spatial dimension}.$$

Typically $d = 3$; however, most of the visualization in this book is for $d = 2$. For any spatial points, a gray value is assigned. Figure 2.1 illustrates the functional





(a) $\mathcal{T}$ with transformed grid (b) $\mathcal{T}[y]$ with grid

Figure 2.2: *Transforming images.*

FAIR 1: Flexible Algorithms for Image Registration (FAIR)

Given two images $\mathcal{R}, \mathcal{T} : \Omega \subset \mathbb{R}^d \to \mathbb{R}$, find a transformation $y : \mathbb{R}^d \to \mathbb{R}^d$ such that

$$\mathcal{J}[y] = \mathcal{D}[\mathcal{T}[y], \mathcal{R}] + \mathcal{S}[y] \xrightarrow{y} \min$$

FAIR abbreviations	
d	spatial dimension
$\Omega \subset \mathbb{R}^d$	region of interest
$\mathcal{T},\ \mathcal{R} : \Omega \to \mathbb{R}$	template and reference images
$y : \Omega \to \mathbb{R}^d$	transformation
$\mathcal{T}[y]$	transformed image, $\mathcal{T}[y](x) = \mathcal{T}(y(x))$
$\mathcal{D}$	distance measure
$\mathcal{S}$	regularizer
$\mathcal{J}$	joint functional

view with the gray value as a height as well as the image view, where the points are colored according to the gray value. Note that the labels on the x^1 and x^2 axes refer to a spatial domain $\Omega = (0, 50) \times (0, 50)$ rather then to a particular resolution.

Since images are modeled continuously, image transformation can easily be phrased. The transformed image is denoted by $\mathcal{T}[y]$, where $y : \mathbb{R}^d \to \mathbb{R}^d$ denotes the transformation and

$$\mathcal{T}[y](x) = \mathcal{T}(y(x)).$$

Note that this so-called Eulerian approach transforms the domain. As a result, the corresponding transformation of the images is counterintuitive. Figure 2.2 gives an example of a counterclockwise rotation of 45 degrees. Note that the grid is rotated

counterclockwise, which results in a transformed image which looks like a clockwise-rotated copy of the template. More details are provided in Chapter 3. The so-called Lagrangian framework, which is to follow tissue points, is not discussed here.

Although the power of the techniques to be discussed in this book is best visible for nonparametric problems, they can also be used in the restricted parametric setting. Chapter 4 explains the handling of parameterized transformations $y = \mathcal{Q}w$, where $\mathcal{Q}$ is a collection of basis functions and w is a collection of parameters or coefficients. Important classes such as rigid, affine linear, and spline transformations are covered.

2.1.2 Distances and Regularization

So far, only the so-called forward problem has been addressed: given a transformation, how to compute the transformed image? The next step is to provide a quantitative measure for the quality of the transformation. This measure has two ingredients. The first ingredient is related to image similarity, and the second ingredient measures plausibility or regularity of the transformation. Feature- or landmark-based approaches are discussed in Chapter 5, and volumetric distance measures are the topic of Chapter 7. Figure 1.1 shows the difference image before and after registration. Here, the sum of squared differences (SSD) (measuring the energy of the difference image) is used as an intuitive example, where

$$\mathcal{D}[\mathcal{T}[y], \mathcal{R}] = \frac{1}{2} \int_{\Omega} (\mathcal{T}[y](x) - \mathcal{R}(x))^2 \, dx.$$

Since registration is an ill-posed problem, regularization becomes inevitable; see [96, 162]. Chapter 8 gives more details and explores the most common regularization choices such as the elastic potential of the transformation y, where

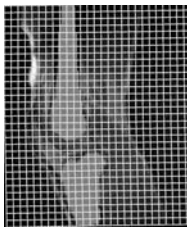
$$\mathcal{S}[y] = \text{Elastic Potential}[y - y^{\text{ref}}], \quad \text{where} \quad y^{\text{ref}}(x) = x.$$

2.2 FAIR Numerics

The numeric presented in this book is based on a discretize-then-optimize approach, where a nested sequence of discretizations is solved using plain vanilla Newton-type techniques.

2.2.1 Discretize-then-Optimize

Most of the registration problems do not allow for an analytic solution, and thus numerical solutions are to be provided. A feasible approach is to solve the discretized optimality condition for the continuous problem (2.1); see, e.g., [162]. This book, however, focuses on numerical optimization and therefore on optimality conditions for the discretized problem. To be more precise, a sequence $\{J^h\}$ of discretizations running from coarse to fine of the continuous functional $\mathcal{J}$ is considered. The idea is to capture the important features on a coarse presentation and to solve this



problem with relatively low computational costs. For finer representations, only corrections based on the added information are required. A key point is that all the discrete problems are linked by the same underlying continuous model, and thus the sequence of solutions y^h approximates the solution of the continuous problem. Another important point is that on each discretization level an optimization problem is to be solved. Thus, consistent line search techniques and automatic stopping criteria can be used.

2.2.2 A Family of Nested Approximations

Being able to play with consistent approximations of the registration problem is a major feature of FAIR. Multiscale and multilevel strategies for a particular problem are used for many good reasons. Using a coarse representation yields a smoother objective function, which prevents us from running into local minima, enables fast optimization techniques, and results in fewer unknowns, which is good for memory, computing time, and results.

Discretization is also used for approximation of integrals and derivatives. Using structured grids of cells with centers x_i and volume h, roughly speaking, we have

$$\int f(x)\ dx = h \sum f(x_i) + \mathcal{O}(h^2) \text{ and}$$
$$\partial f(x_{i+0.5}) = (f(x_{i+1}) - f(x_i))/h + \mathcal{O}(h^2),$$

where the Landau symbol $\mathcal{O}$ indicates errors of order h^2.

A visualization of f can be achieved by assigning the value $f(x_i)$ to a cell of volume h and displaying the piecewise constant. Note that in contrast to a pixel or voxel point of view, the discretization used by FAIR results from discretizations of (2.1) and is variable.

2.2.3 Numerical Optimization

An additional feature of FAIR is its focus on differentiable modules. Derivatives are the most important ingredient to most of the efficient numerical optimization techniques. Emphasis has also been given to proper line search and stopping criteria used by optimization schemes. The numerical optimization is explained in Chapter 6 for the parametric case and in Chapter 9 for the nonparametric case. Although other optimization techniques are discussed and considered in the exercises, the default scheme is a Gauss–Newton-type scheme with an Armijo line search.

2.3 FAIR MATLAB

The programs are developed using MATLAB software. This is certainly a limitation in terms of speed and memory, but MATLAB provides a fast, easy, and intuitive access to numerical computing, and in particular to sparse matrices. It has been chosen because it is easy to use for education and research.

2.3.1 Comments on Comments

The programs and their functionality are explained and commented on primarily in the text. To avoid confusion raised by different comments in the text and the code, the comments in the code are kept brief.

2.3.2 Notation and Conventions

Similar quantities are collected in arrays, and data structures that are adapted to the linear algebra are used as often as possible. For example, the size of the data is denoted by $m = [m^1, \dots, m^d]$. Collecting grid points is slightly more evolved and is discussed in Section 3.1 in detail. However, using vectorized data structures enables a direct and simple access to linear algebra and numerical optimization.

The typical LaTeX style (such as x or T) is used for variables in a mathematical context, whereas the `verbatim` style (such as `x` or `T`) is used for variables which are also used in programs. For integers (and arrays of integers) variable names are taken from {`i,j,k,l,m,n,p`}. Grids and transformed grids are denoted by `x*` and `y*`, respectively; for example, a starting guess is denoted by `y0`, the current iterate is denoted by `yc`, and the grid size is denoted by `h`. Matrices are denoted by uppercase characters. Derivatives are stored as derivatives, not as gradients: for $f : \mathbb{R}^n \to \mathbb{R}$, the derivative df is 1-by-n.

2.3.3 Coordinate System

Throughout the book a geometric right-handed (x^1, x^2, x^3) coordinate system is used. All coordinates are absolute and physical and thus changing the discretization does not affect positioning of data or scaling of the registration problem. For example, the plots shown in Figure 2.1 illustrate the functions $\mathcal{T}$, y^1, and y^2 on the domain $\Omega = (0, 50)^2$. A point $x = [x^1, x^2] \in \mathbb{R}^2$ refers to a physical point in this domain. Unfortunately, MATLAB stores two-dimensional (2D) arrays in an (i, j) coordinate systems and therefore imaging functions such as `image` should not be used directly; more details are provided in Chapter 3.

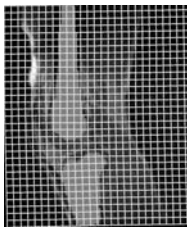
2.3.4 Arguments, Parameters, and Defaults

Although the distinction of variables and parameters is somehow arbitrary, the input list of programs is grouped into a number of variables and a number of parameters. The variables are mandatory, while the parameters are predefined and may be omitted in a function call.

Example 2.1 (Default Parameters)
An example is given by

```
plot(0,1,'color','r','marker','x').
```

The variables are `0` and `1` and a list of parameters is `{'color','r','marker','x'}`. Note that the parameters are predefined and need not be supplied.



2.3.5 Overwriting Default Parameters

In FAIR, default values for parameters are assigned in the beginning of a function, and these parameters are updated using an input list.

Example 2.2 (Overwriting Default Parameters)
A function, which is explained in detail in Chapter 6, can be called as follows:

```
yc = GaussNewtonArmijo(fctn,y0,'maxIter',5).
```

The mandatory variables are `fctn` and `y0`, a handle to the function to be minimized and a starting guess. In the code, defaults are assigned to the parameters, and the following for-loop overwrites the defaults using the values from the parameter list.

```
function [yc,his] = GaussNewtonArmijo(fctn,yc,varargin)

% parameter initialization ------------------------------------------------
maxIter      = 10;              % maximum number of iterations
tolF         = 1e-3;            % for stopping, objective function
tolX         = 1e-2;            %   - " -     , current value
tolG         = 1e-2;            %   - " -     , norm of gradient
LSMaxIter    = 10;              % maximum number of line search iterations
LSreduction  = 1e-4;            % minimal reduction in line search
vecNorm      = @norm;           % norm to be used for dJ and dY
solver       = regularizer;     % linear solver
yStop        = [];              % used for stopping in multilevel framework
Fstop        = [];              %
Plots        = @(iter,para) []; % for plots;

for k=1:2:length(varargin),     % overwrites default parameter
  eval([varargin{k},'=varargin{',int2str(k+1),'};']);
end;
% here starts the interesting stuff
```

If, for example, `varargin={'maxIter',5}`, the statement in the for-loop reads `eval(['maxIter=5;'])` and thus overwrites the default value `10` of `maxIter` by `5`.

2.3.6 Using the MATLAB "@" Constructor

Sometimes, it is quite helpful to apply Occam's razor and use the MATLAB "`@`" constructor to focus on essentials. Although the usage of this option is tempting, it is used sparsely to avoid over abstraction.

Example 2.3 (Using the MATLAB @ Constructor)
An objective function typically depends on many variables. For example, setting

```
objFctn = @(wc) PIRobjFctn(T,R,omega,m,beta,M,wRef,xc,wc)
```

enables a compact call `Jc=objFctn(wc)` and hides all variables not relevant for optimization.

2.3.7 FAIR Administration

Administration is tiresome and complicated but somehow a necessary and inevitable overhead. In FAIR, the main building blocks are administered by specific functions

with standardized input and rules. An administrative function `caller` for a generic task is parameterized on the basis of a list of persistent parameters `OPTN`. Each administrative functions handles one of the following tasks and is configured by a list of options (see also `options` in FAIR):

- `caller('reset','caller',method,name1,value1,...)`: clears all options and sets the method to be used to `method` and adds variables with `name1` and value `value1`, etc. to the persistent list of parameters
- `caller('set','name1',value1,'name2',value2,...)`: adds (or overwrites) `{'name1',value1,'name2',value2,...}` to the persistent parameters
- `caller('clear')`: clears persistent parameters
- `caller('disp')`: displays persistent parameters
- `[method,optn]=caller`: returns the specific methods and the persistent options
- `value=caller('get','name')`: returns the value of `name` or `[]` if not defined
- `[y1,y2,...]=caller(x1,x2,...)`: executes the function based on input variables and the persistent options.

Example 2.4 (Interpolation Administration)
In FAIR, any interpolation result depends on the data, the domain, and the points at which the interpolant is to be evaluated. A natural call thus reads `Tc= inter(T,omega,xc)`. Note that the current points are denoted by `xc`, the coefficients for the interpolation are denoted by `T`, and the interpolated values are called `Tc`. An initialization call defines the scheme to be used, and a check using `disp` verifies that the options have been set properly:

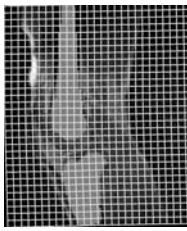
```
inter('reset','inter','linearInter3D'); inter('disp');
```

2.3.8 Memory Versus Clarity

In this book, clarity is the winner in the battle between clarity and efficiency. A higher-dimensional formula can often be derived using an appropriate combination of one-dimensional (1D) formulae. An important but often not very efficient tool is the so-called Kronecker product; see FAIR 2 (p.17).

Often, these Kronecker products involve an identity matrix I_n of appropriate size, where

$$I_n = \texttt{speye(n,n)} = \begin{bmatrix} 1 & & 0 \\ & \ddots & \\ 0 & & 1 \end{bmatrix} \in \mathbb{R}^{n,n}. \tag{2.2}$$



FAIR 2: Kronecker Products

Given two matrices $A \in \mathbb{R}^{p^1,p^2}$ and $B \in \mathbb{R}^{q^1,q^2}$, the Kronecker product is defined by

$$A \otimes B = \begin{bmatrix} a_{1,1}B & \cdots & a_{1,p^2}B \\ \vdots & \ddots & \vdots \\ a_{p^1,1}B & \cdots & a_{p^1,q^2}B \end{bmatrix} \in \mathbb{R}^{p^1q^1,p^2q^2}.$$

Example 2.5 (Kronecker Products)
As it will be explained in Chapter 4 in more detail, a 3D affine linear transformation $y = [y^1, y^2, y^3]$ can be phrased as

$$y^i(x) = [1, x^1, x^2, x^3] \cdot [w^{i+1}, w^{i+2}, w^{i+3}, w^{i+4}]^\top,$$

where $w \in \mathbb{R}^{12}$ denotes the parameters. A compact formulation $y = \mathcal{Q}(x)w$ is obtained by setting

$$\mathcal{Q}(x) = I_3 \otimes [1, x^1, x^2, x^3].$$

Although there are more memory-efficient ways of computing the transformed grid, FAIR stores a discretized version Q of $\mathcal{Q}$ and computes Qw using a simple matrix-vector multiplication.

2.4 FAIR Examples

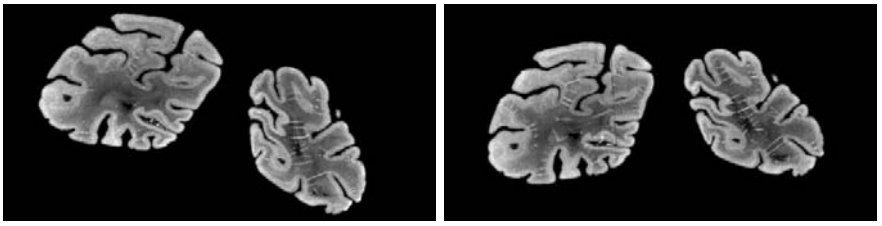
The performance of the algorithm is demonstrated on academic and real-life data. For later reference, a collection of this data is summarized below.

(a) US (b) template hand (c) reference hand

Figure 2.3: *Test images:* (a) *ultrasound image (Vibe Heldmann),* (b) *template and* (c) *reference data from human hands; images from S. Heldmann and Y. Amit* [59]; *see Examples* 2.6 *and* 2.7.

Example 2.6 (Ultrasound Data)
The data of size $m = [693, 385]$ shows Vibe Heldmann; see Figure 2.3 and `setupUSdata`. This data serves as an example for noisy data.

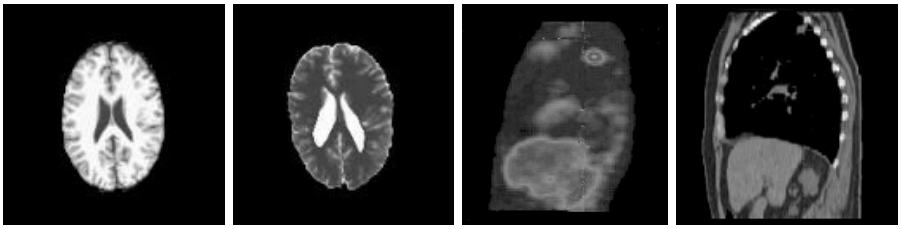
Example 2.7 (X-ray Hand Data)
The data of size $m = [128, 128]$ has been taken from [59]; see also `setupHandData` and Figure 2.3. This intuitive data provides a challenging test case and is used to illustrate problems with local minima.



(a) template section (b) reference section

Figure 2.4: *Test images: data from a histological serial section; images provided by O. Schmitt, Institute of Anatomy, University of Rostock, Germany* [180]; *cf. Example* 2.8.

Example 2.8 (Histological Serial Sectioning Data)
The data of size $m = [512, 256]$ has been provided by O. Schmitt, Institute of Anatomy, University of Rostock, Germany [180]; see also `setupHNSPdata` and Figure 2.4. It is generated in the Human NeuroScanning Project (HNSP) and serves as a prototype for a relatively easy registration problem.

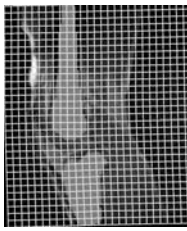


(a) T1 weighted MRI (b) T2 weighted MRI (c) PET (d) CT

Figure 2.5: *Test images:* (a) *T1 and* (b) *T2 weighted MRIs from BrainWeb* [89], *see Example* 2.9. (c) *PET and* (d) *CT images from R. Shekhat et al.* [184], *see Example* 2.10.

Example 2.9 (T1 and T2 Weighted MRIs)
The data of size $m = [128, 128]$ is taken from BrainWeb [89]; see also `setupMRIData` and Figure 2.5. It serves as a prototype for a registration problem with mildly different modalities.

Example 2.10 (CT/PET Data)
CT and PET images from the thorax serve as data for multimodal image registration; data taken from R. Shekar et al. [184]. See also Figure 2.5.



Chapter 3
Image Interpolation

This chapter introduces FAIR interpolation tools. The objective is to find a function $\mathcal{T}$ interpolating data $\texttt{dataT} \in \mathbb{R}^n$ given on a grid. More precisely, for given points $x_j \in \mathbb{R}^d$ the function $\mathcal{T}$ is required to satisfy the interpolation condition

$$\mathcal{T}(x_j) = \texttt{dataT}(j) \quad \text{for} \quad j = 1, \ldots, n.$$

As an additional convention, it is assumed that the function $\mathcal{T}$ vanishes outside the domain, i.e., $\mathcal{T}(x) = 0$ for $x \notin \Omega$.

After introducing basic notations and conventions used for data presentation in Section 3.1, the interpolation schemes are ready to be described. Although commonly used in image processing and therefore briefly discussed in Section 3.2, the *next neighbor* interpolation is not used in FAIR. The reason is that it results in a function whose derivative is either zero or undefined. This causes problems in the optimization process to be used later. Therefore, the emphasize in this book is on *linear* and *spline* interpolations; see Sections 3.3 and 3.4. More issues on interpolations are covered by the books listed in Section 1.4. The piecewise linear interpolant is differentiable almost everywhere, where the spline is as smooth as wanted. Section 3.5 illustrates how to return a derivative of an interpolation scheme, an essential issue in optimization.

For the more experienced reader, Section 3.6 introduces a technique to represent the data on different scales. Loosely speaking, a continuous parameter controls smoothness versus data proximity. Similar effects can be obtained by removing high frequencies in the image presentation.

In contrast to this multiscale approach, where a continuous parameter controls the amount of details of the data representation, the discrete multilevel approach discussed in Section 3.7 presents data on different levels. Finally, Section 3.8 summarizes the interpolation tools explained in this chapter.

Some general concepts are outlined, and details are given in the following sections. A parameterized continuous model based on the given data and interpolation or approximation preferences is generated. Note that only for very simple interpolation techniques will the data and the parameters of the model coincide.

The continuous model

$$\mathcal{T}(x) = \texttt{inter(T,omega,xc)}$$

can be evaluated not only for a particular point x but for any collection $\texttt{xc} = [x_j]_{j=1}^n$ of n points. Note that the parameters of the interpolation models are assembled using `T`. As mentioned before, these parameters do not necessarily coincide with the original data. Only for special cases such as linear interpolation are coefficients and data equal. If in addition the interpolation points coincide with the measurement locations, the result

$$T(\texttt{xc}) = [\mathcal{T}(x_j)]_{j=1}^n$$

also equals the given data, thus making the data redundant. However, more interesting scenarios are to be considered and discussed in the forthcoming chapters.

3.1 Cells, Grids, and Numbering

Let d denote the spatial dimension of the given data. In practice, $d = 2$ or $d = 3$, where for the first reading $d = 1$ might be helpful. It is assumed that given data with $\texttt{dataT}(j) \in \mathbb{R}$ is related to points

$$x_j = [x_j^1, \ldots, x_j^d] \in \mathbb{R}^d, \quad j = 1, \ldots, n,$$

and that these points are the cell-centers of a regular grid of a d-dimensional interval $\Omega = (\omega^1, \omega^2) \times \cdots \times (\omega^{2d-1}, \omega^{2d}) \subset \mathbb{R}^d$; see Figures 3.1 and 3.2 for examples. Superscript indices are used for components of vectors, and subscript indices are used for numbering. For the purposes of this book, it is safe to assume that $\omega^{2i-1} = 0$. However, for some problems such as data fusion it is beneficial to enable a more general description of the domain in the code.

A *grid* is a partitioning of the interval into a number of congruent cells or boxes. Thus, the ith component of the difference between two grid points is a multiple of a constant grid width h^i. For example, let the data size $m = [m^1, \ldots, m^d]$ be given and let

$$h^i = (\omega^{2i} - \omega^{2i-1})/m^i, \qquad h = [h^1, \ldots, h^d], \tag{3.1}$$

$$\xi_j^i = \omega^{2i-1} + (j - 0.5)h^i, \qquad \xi^i = [\xi_1^i, \ldots, \xi_{m^i}^i] \in \mathbb{R}^{m^i}. \tag{3.2}$$

Index vectors $j = [j^1, \ldots, j^d]$ are used for accessing elements of higher-dimensional arrays. Also the collection of points

$$x_j = [\xi_{j^1}^1, \ldots, \xi_{j^d}^d], \quad j^i = 1, \ldots, m^i, \quad i = 1, \ldots, d,$$

is called a *cell-centered grid*, and the d-dimensional intervals

$$\text{cell}_j = \{x \in \mathbb{R}^d \mid -h^i/2 < x^i - \xi_j^i < h^i/2\}$$

are called *cells*, where the *cell centers* are x_j.

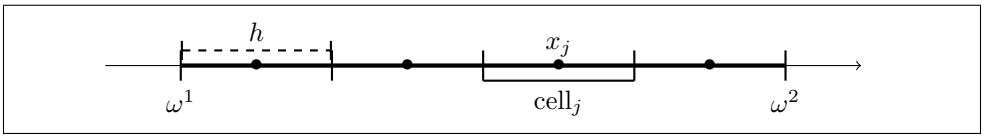


Figure 3.1: *Discretization of a 1D domain* $\Omega = (\omega^1, \omega^2) \subset \mathbb{R}$.

Remark 3.1
The variables used in the FAIR *package have similar names but a different font. For example,* `d` *is the* MATLAB *equivalent to spatial dimension* d. *The superscribed variables (like* m^i, h^i, ω^i*) are assembled as row vectors (like* `m`, `h`, `omega`*). Subscripted variables are collected as column vectors and variables with super- and subscripts as matrices, e.g.,* `xi(j,i)` $= \xi_j^i$.

The variable `omega` *can also be used to describe different domains for the template and reference image:* `omega(1,:)` *is used for the template and* `omega(end,:)` *is used for the reference. However, this* FAIR *option is not used in this book.*

Example 3.1 (Cells and Grids in One Dimension)
Figure 3.1 displays a 1D example. The interval $\Omega = (\omega^1, \omega^2)$ is divided into $m = 4$ cells of length $h = (\omega^2 - \omega^1)/m$, with center $x_j = \xi_j^1$. A MATLAB statement reads

```
h = (omega(2:2:end)-omega(1:2:end-1))./m,
x = omega(1)+h/2:h:omega(2)-h/2.
```

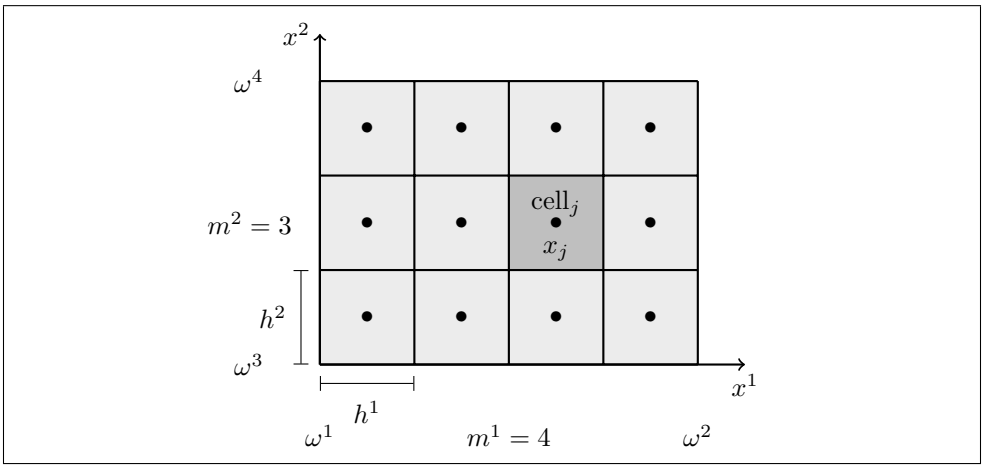


Figure 3.2: *Discretization of a 2D domain* $\Omega = (\omega^1, \omega^2) \times (\omega^3, \omega^4) \subset \mathbb{R}^2$.

In higher dimension, there is a conflict between the physical location of the points and their numbering. This is illustrated in the following example.

Example 3.2 (Cells and Grids in Two Dimensions)
Figure 3.2 displays a 2D example. The domain $\Omega = (0, 4) \times (0, 3)$ is divided into 4-by-3 cells; i.e., $m = (4, 3)$ and $h^1 = h^2 = 1$, $\xi^1 = [0.5, 1.5, 2.5, 3.5]$, and $\xi^2 =$

$[0.5, 1.5, 2.5]$. The cell centers are $x_j = [\xi^1_{j^1}, \xi^2_{j^2}]$, where $j = [j^1, j^2]$, $j^1 = 1, 2, 3, 4$, and $j^2 = 1, 2, 3$. The components x^1_j and x^2_j can be arranged in m^1-by-m^2 arrays,

$$X^1 = \begin{array}{|c|c|c|}\hline 0.5 & 0.5 & 0.5 \\ \hline 1.5 & 1.5 & 1.5 \\ \hline 2.5 & 2.5 & 2.5 \\ \hline 3.5 & 3.5 & 3.5 \\ \hline \end{array}, \qquad X^2 = \begin{array}{|c|c|c|}\hline 0.5 & 1.5 & 2.5 \\ \hline 0.5 & 1.5 & 2.5 \\ \hline 0.5 & 1.5 & 2.5 \\ \hline 0.5 & 1.5 & 2.5 \\ \hline \end{array}.$$

Note that the grid point for $j = [3, 2]$ is $X_j = [X^1_j, X^2_j] = [\xi^1_3, \xi^2_2] = [2.5, 1.5]$ and cell_j is the dark gray cell as shown in Figure 3.2.

Example 3.3 (Generating Cell-Centered Grids)
The function `getCenteredGrid` provides a convenient way of generating arbitrary cell-centered grids for domains of dimensions $d = 1, 2, 3$. As a convention, the output `xc` is a column vector of length `d*prod(m)`. If necessary, this vector can easily be reshaped to enable access to the different coordinates:

- `xc=reshape(xc,[],d)` returns a `prod(m)`-by-`d` array, where the `xc(j,:)` gives the jth grid point, and `reshape(xc(:,j),m)` returns the jth coordinate as a `d`-dimensional array of size `m`;
- `xc=reshape(xc,[m,d])` returns a `m(1)`-by-$\cdots$-by-`m(d)`-by-`d` array.

This file is `E3_getCenteredGrid.m`

```
%%%
omega = [0,6,0,4,0,8], m = [3,2,2]
xc = getCenteredGrid(omega(1:2),m(1));   xc = reshape(xc,1,[])
xc = getCenteredGrid(omega(1:4),m(1:2)); xc = reshape(xc,[m(1:2),2])
xc = getCenteredGrid(omega(1:6),m(1:3)); xc = reshape(xc,[m(1:3),3])
```

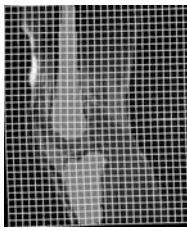
3.1.1 Right-Handed Coordinate System

Some visualization tools, e.g., `image` or `imagesc`, assume that the data is given in a left-handed (i, j) coordinate system, where i denotes the row (top to bottom) and j the column (left to right). In order to get the same ordering for the data and the cell centers, an initial conversion of the data into a right-handed format pays off. Let `Tij` be the given data in (i, j) coordinate system; the reoriented data is given by `dataT`:

1D: `dataT=reshape(Tij,[],1)`,

2D: `dataT=flipud(Tij)'`,

3D: `dataT=flipdim(flipdim(permute(Tij,[3,2,1]),3),1)`.



Example 3.4 (Changing Coordinate Systems in Two Dimensions)
This example continues Example 3.2. Let `Tij` be 3-by-4, `dataT=flipud(Tij)'`, with

$$\texttt{Tij} = \begin{array}{|c|c|c|c|}\hline t_{1,1} & t_{1,2} & t_{1,3} & t_{1,4} \\ \hline t_{2,1} & t_{2,2} & t_{2,3} & t_{2,4} \\ \hline t_{3,1} & t_{3,2} & t_{3,3} & t_{3,4} \\ \hline \end{array} = \begin{array}{|c|c|c|c|}\hline 1 & 2 & 3 & 4 \\ \hline 1 & 2 & 3 & 4 \\ \hline 4 & 4 & 4 & 4 \\ \hline \end{array}, \quad \texttt{dataT} = \begin{array}{|c|c|c|}\hline 4 & 1 & 1 \\ \hline 4 & 2 & 2 \\ \hline 4 & 3 & 3 \\ \hline 4 & 4 & 4 \\ \hline \end{array}.$$

For example, for $j = [3, 2]$, the data associated with $x_j = [X_j^1, X_j^2] = [2.5, 1.5]$ (dark gray cell in Figure 3.2) is `Tij(2,3)` $= 3$. On the other hand, for any j, the data `dataT`(j) is intuitively assigned to x_j.

Example 3.5 (Changing Coordinate Systems in Three Dimensions)
Try the following commands. Note that for $j = [1, 1, 1]$ (back, left, and bottom entry) `T(1)=T(j)=9` is assigned to $x_j = [x_j^1, x_j^2, x_j^3]$.

This file is `E3_ij2xy3D.m`

```
%%%
Tij=reshape(1:12,[3,2,2]),
disp(flipdim(flipdim(permute(Tij,[3,2,1]),3),1))
disp(['T(:)''= [ 9  3 12  6  8  2 11  5  7  1 10  4]'])
```

3.1.2 Lexicographical Ordering

In particular for the optimization schemes to be discussed later, it is convenient to rearrange d-dimensional arrays as long vectors. Here the lexicographical ordering is used. For $j = [j^1, \ldots, j^d]$,

1D: $j \mapsto j^1$,

2D: $j \mapsto j^1 + m^1(j^2 - 1)$,

3D: $j \mapsto j^1 + m^1(j^2 - 1) + m^1 m^2(j^3 - 1)$.

To reduce the notational overhead, the number on the right side is again denoted by $j \in \mathbb{N}$. Of course, any other ordering could be used as long as it is used consistently.

Example 3.6 (Lexicographical Ordering in Two Dimensions)
For the data of Examples 3.2 and 3.4 the vectorized quantities are

$$\begin{aligned} X^1(:) &= [\ 0.5;\ 1.5;\ 2.5;\ 3.5;\ \ 0.5;\ 1.5;\ 2.5;\ 3.5;\ \ 0.5;\ 1.5;\ 2.5;\ 3.5\ ], \\ X^2(:) &= [\ 0.5;\ 0.5;\ 0.5;\ 0.5;\ \ 1.5;\ 1.5;\ 1.5;\ 1.5;\ \ 2.5;\ 2.5;\ 2.5;\ 2.5\ ], \\ \texttt{dataT}(:) &= [\ \ 4;\ \ 4;\ \ 4;\ \ 4;\ \ \ 1;\ \ 2;\ \ 3;\ \ 4;\ \ \ 1;\ \ 2;\ \ 3;\ \ 4\ ]. \end{aligned}$$

Note that if `xc= getCenteredGrid(omega,m)`, then `xc` $= [X^1(:); X^2(:)]$.

The grid points x_j, $j = 1, \ldots, n$, are collected in a vector `xc` of length dn. In MATLAB, vectorized versions of arrays of size `m` can be obtained by setting `Tvector=Tarray(:)`. On the other hand, `reshape(Tvector,m)` returns the array of size `m`. Therefore, standard procedures for visualization can be used. For example, `viewImage2D` enables the graphical output of vectorized 2D data. A reasonable visualization of 3D data is beyond the scope of this book. However, a poor man's version is the topic of Exercise 3.3.

3.2 Next Neighbor Interpolation

Having finished the basic notations and conventions, the interpolation schemes are ready to be described. Although commonly used in image processing, next neighbor interpolation is not used in FAIR. The reason is that the interpolant is not continuous which causes trouble in the optimization schemes to be discussed later. However, the procedure is defined by setting $\mathcal{T}^{\text{nn}}(x) = 0$ for $x \notin \Omega$ and $\mathcal{T}^{\text{nn}}(x) := \texttt{dataT}(j)$, where j is such that $x \in \text{cell}_j$; see Figure 3.3 for a 1D example.

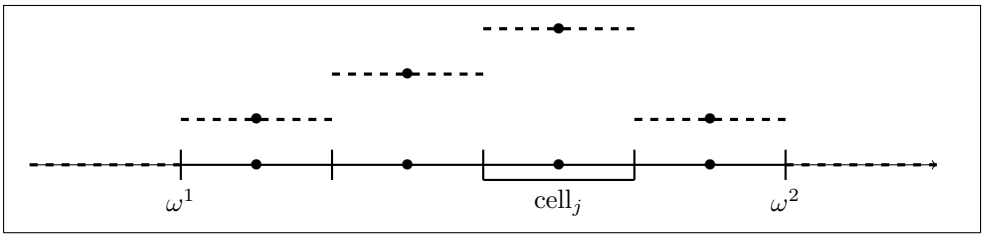


Figure 3.3: *Next neighbor interpolation in one dimension (dashed line).*

3.3 Linear Interpolation

The 1D case provides the necessary insight for the linear interpolation schemes. Kronecker product techniques are used to extend the scheme to higher dimensions.

3.3.1 Linear Interpolation for 1D Data

The basic idea is to assign a function value at a certain position as a weighted sum of the function values of the neighboring points. By using the simple linear map

$$x \mapsto x' = (x - \omega^1)/h + 0.5, \tag{3.3}$$

a domain $\Omega = (\omega^1, \omega^2)$ is mapped onto $\Omega' = (0.5, m + 0.5)$ and, in particular, $x_j = \omega^1 + (j - 0.5)h$ is mapped onto j. Thus, the neighbors and the weights for an arbitrarily chosen point x can be easily obtained by splitting x' into an integer part p and a remainder ξ, where

$$p = \lfloor x' \rfloor := \max\{j \in \mathbb{Z} \mid j \leq x'\} \quad \text{and} \quad \xi = x' - p, \quad 0 \leq \xi < 1. \tag{3.4}$$

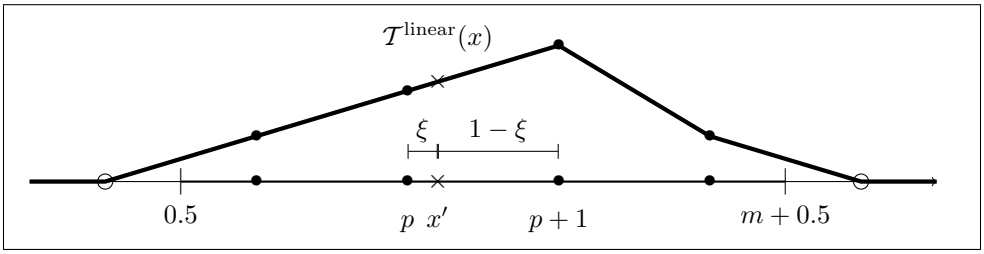


Figure 3.4: *Linear interpolation in one dimension.*

For $d = 1$ one could avoid the above mapping by starting with the primed quantities and, in particular $h' = 1$. For $d > 1$, however, the distance of data points can vary for different directions and the assumption $h = (1, \ldots, 1)$ can be too restrictive.

The basic formula for linear interpolation reads

$$\mathcal{T}^{\text{linear}}(x) := \texttt{dataT}(p) \cdot (1 - \xi) + \texttt{dataT}(p + 1) \cdot \xi; \tag{3.5}$$

see also Figure 3.4. Equation (3.5) is only valid for $x_1 \leq x < x_m$ or, equivalently, $1 \leq x' < m$. Adding the two additional artificial data points $(0, 0)$ and $(m + 1, 0)$ provides a simple trick to reduce the cases to be discussed in the implementation: For all $x \in \Omega$ and thus $x' \in [0.5, m+0.5]$ it then holds that $0 \leq p \leq m$ and $0 \leq \xi < 1$. Note, however, that $\mathcal{T}^{\text{linear}}$ might not be completely zero outside the domain $\Omega' = (0.5, m + 0.5)$; see also Figure 3.4. Fortunately, this is a minor problem: based on the assumption that $\mathcal{T}$ is compactly supported, one could always choose a bigger interval such that $\texttt{dataT}(1) = \texttt{dataT}(m) = 0$ (padding the data) which would resolve this problem immediately. Note, however, that in most applications the data is not compactly supported. See Chapter 10 for a discussion.

3.3.2 Linear Interpolation for Higher-Dimensional Data

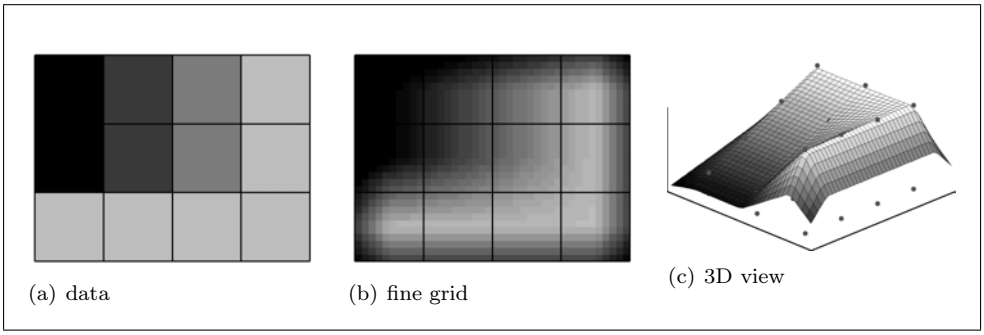
The extension to higher dimensions is based on a Kronecker product approach; cf. FAIR 2 (p.17). Here, the above concepts are applied to any coordinate, and $\mathcal{T}^{\text{linear}}$ is computed as a weighted sum of the data from neighboring cells,

$$\mathcal{T}^{\text{linear}}(x) = \sum_{k \in \{0,1\}^d} \texttt{dataT}(p + k) \prod_{i=1,\ldots,d} (\xi^i)^{k^i} (1 - \xi^i)^{(1-k^i)}. \tag{3.6}$$

For $d = 2$, (3.6) results in $p = (p^1, p^2)$, $\xi = (\xi^1, \xi^2)$, and

$$\begin{aligned}\mathcal{T}^{\text{linear}}(x) = {} & \texttt{dataT}(p^1, p^2)(1 - \xi^1)(1 - \xi^2) + \texttt{dataT}(p^1 + 1, p^2)\xi^1(1 - \xi^2) \\ & + \texttt{dataT}(p^1, p^2 + 1)(1 - \xi^1)\xi^2 + \texttt{dataT}(p^1 + 1, p^2 + 1)\xi^1\xi^2.\end{aligned}$$

For an implementation see also Example 3.7.



(a) data (b) fine grid (c) 3D view

Figure 3.5: *Linear interpolation of the data of Example 3.7 (a particular colormap has been used for* (a) *and* (b)*).*

3.3.3 Summarizing Linear Interpolation

Linear interpolation is a reasonable tool in image registration. The interpolant can be evaluated with low computational costs and has attractive features. For example, the values of the interpolant do not exceed the interval spanned by the data, and the interpolation has no spurious oscillations; see, e.g, Figure 3.5. However, although the interpolant is differentiable almost everywhere, it is not differentiable at the grid points. Linear interpolation is thus the interpolation method of choice when no derivatives are needed. In order to benefit from fast and efficient optimization schemes, smoother interpolants are needed. Among the many existing techniques, cubic spline interpolation is a reasonable compromise between differentiability and efficiency and is thus supported by FAIR.

Example 3.7 (Linear Interpolation in Two Dimensions)
A typical linear interpolation call; see `linearInter2D` for an implementation.

This file is `E3_linearInterpolation2D.m`

```
%%%
dataT = flipud([1,2,3,4;1,2,3,4;4,4,4,4])';
m     = size(dataT);
omega = [0,m(1),0,m(2)];
M     = {m,10*m};   % two resolutions, coarse and fine
xc = reshape(getCenteredGrid(omega,M{1}),[],2);     % coarse resolution
xf = reshape(getCenteredGrid(omega,M{2}),[M{2},2]); % fine   resolution
Tc = linearInter2D(dataT,omega,xc(:));
Tf = linearInter2D(dataT,omega,xf(:));
clf; ph = plot3(xc(:,1),xc(:,2),Tc(:),'ro'); hold on;
qh = surf(xf(:,:,1),xf(:,:,2),reshape(Tf,M{2}));
```

3.4 Spline Interpolation

While the next neighbor and linear interpolations are probably originated in graphics, the spline approach has its origin in shipbuilding. The objective is to find a function $\mathcal{T}^{\text{spline}}$ interpolating the data and minimizing its bending energy. Again, the 1D situation provides a perfect starting point for higher dimensions, where schemes are derived from a Kronecker product approach.

3.4.1 Spline Interpolation for 1D Data

The bending energy is approximated by an integration over the square of the second derivative,

$$\mathcal{S}[\mathcal{T}] = \int_{\Omega} (\mathcal{T}''(x))^2 \, dx. \tag{3.7}$$

The solution of the interpolation problem

$$\mathcal{S}[\mathcal{T}] \stackrel{!}{=} \min \quad \text{subject to} \quad \mathcal{T}(x_j) = \texttt{dataT}(j), \quad j = 1, \ldots, m,$$

is a cubic spline that can be expanded in terms of some coefficients c_j and basis functions b_j; cf., e.g., [162]. One of the many outstanding properties of a spline space is that it allows for an expansion in terms of a simple basis, where each basis function b_j is a translated version of a so-called "*mother*" spline b. For a more extended introduction to spline interpolation see [53, 169, 190] and the books listed in Section 1.4.

In order to achieve a convenient access to the indexing of the basis function, the map introduced in (3.3) is used. The mapped cell-centered grid points are $x_j = j$. Figure 3.6 shows the basis function $b = b^0$ and two arbitrarily chosen translates b^2 and b^7, where $b^j(x) = b(x - j)$ and

$$b(x) = \begin{cases} (x+2)^3, & -2 \le x < -1, \\ -x^3 - 2(x+1)^3 + 6(x+1), & -1 \le x < 0, \\ x^3 + 2(x-1)^3 - 6(x-1), & 0 \le x < 1, \\ (2-x)^3, & 1 \le x < 2, \\ 0, & \text{else.} \end{cases} \tag{3.8}$$

The goal is to expand the interpolant by

$$\mathcal{T}(x) = \mathcal{T}^{\text{spline}}(x) = \sum_{j=1}^{m} c_j b^j(x) \tag{3.9}$$

and to derive fast ways for evaluating (3.9) and for computing the coefficients $c = [c_1; \ldots; c_m]$. Expanding (3.9) at the cell centers $x_j = j$ gives the interpolation

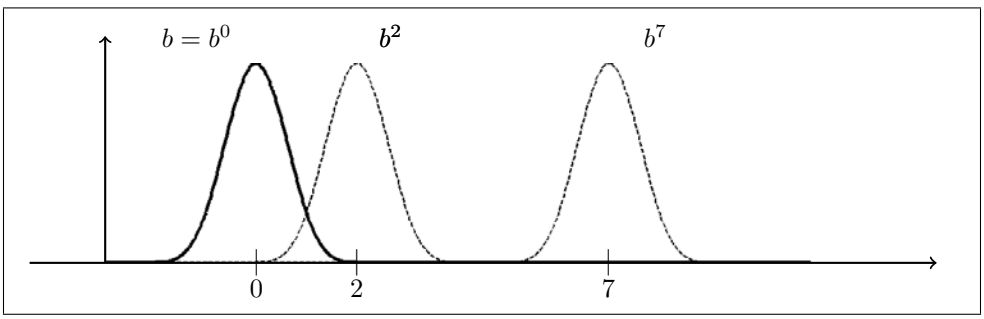


Figure 3.6: *"Mother" spline $b = b^0$ (solid) and basis functions b^2 and b^7.*

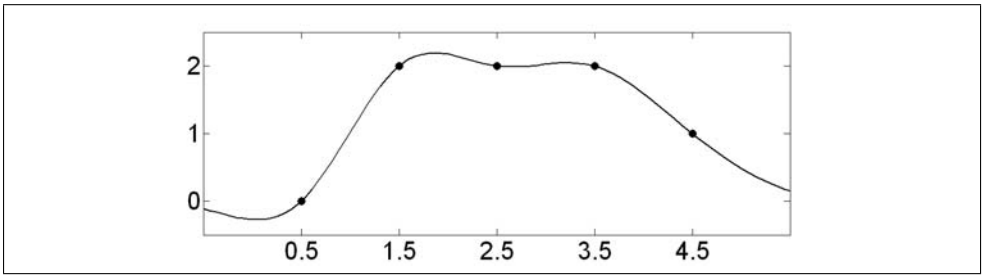


Figure 3.7: *Spline interpolation in one dimension; see Example 3.8.*

condition

$$\texttt{dataT}(j) = \mathcal{T}(x_j) = \sum_{k=1}^{m} c_k b^k(j) = [b^1(j), \ldots, b^m(j)]\ c, \quad j = 1, \ldots, m. \tag{3.10}$$

Gathering all function values in $T(\texttt{xc}) = [\mathcal{T}(x_1); \ldots; \mathcal{T}(x_m)]$ yields the equivalent formula

$$\texttt{dataT} = T(\texttt{xc}) = [b^1(\texttt{xc}), \ldots, b^m(\texttt{xc})]\ c = B_m\ c \tag{3.11}$$

with

$$B_m = [b^k(x_j)] = \begin{bmatrix} 4 & 1 & & 0 \\ 1 & \ddots & \ddots & \\ & \ddots & \ddots & 1 \\ 0 & & 1 & 4 \end{bmatrix} \in \mathbb{R}^{m,m} \tag{3.12}$$

and presents a convenient formula for computing the coefficients c; see also Exercise 3.6.

Since $b^0(x) = 0$ for $x \notin (-2, 2)$, for any point $x = p + \xi$ with integer part p and remainder ξ (cf. (3.4)), at most four basis functions are nonzero and thus

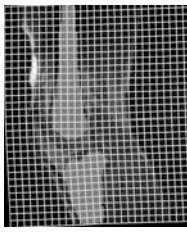
$$\mathcal{T}^{\text{spline}}(x) = c_{p-1}b(\xi + 1) + c_p b(\xi) + c_{p+1}b(\xi - 1) + c_{p+2}b(\xi - 2),$$

which provides an efficient way of evaluating the spline.

Example 3.8 (Spline Interpolation in One Dimension)
The following code illustrates how to use the spline interpolation in one dimension; see also Figure 3.7.

This file is E3_splineInterpolation1D.m

```
%%%
dataT = [0,2,2,2,1]; m = length(dataT); omega = [0,m];
xc = getCenteredGrid(omega,m);
B  = spdiags(ones(m,1)*[1,4,1],[-1:1],m,m);
T  = B\reshape(dataT,m,1);
xf = linspace(-1,6,101);
Tf = book_splineInter1D(T,omega,xf);
figure(1); clf; ph = plot(xc,dataT,'.',xf,Tf,'g-','markersize',30);
```



This file is book_splineInter1D.m

```
%%%function Tc = splineInterpolation1D(T,omega,x);

% get data size m, cell size h, dimension d, and number n of interpolation points
m  = length(T);
h  = (omega(2:2:end)-omega(1:2:end))./m;
d  = length(omega)/2;
n  = length(x)/d;
x  = reshape(x,n,d);
% map x from [h/2,omega-h/2] -> [1,m],
for i=1:d, x(:,i) = (x(:,i)-omega(2*i-1))/h(i) + 0.5; end;

Tc = zeros(n,1);                              % initialize output
Valid = @(j) (-1<x(:,j) & x(:,j)<m(j)+2); % determine indices of valid points
valid = find( Valid(1) );
if isempty(valid), return; end;
                                              % pad data to reduce cases
pad = 3; TP = zeros(m+2*pad,1); TP(pad+(1:m)) = reshape(T,m,1);
P = floor(x); x = x-P;                        % split x into integer/remainder
p = pad + P(valid); xi = x(valid);            % add the padding

b1 = @(xi) (1-xi).^3;                         % abbreviation for the spline segments
b2 = @(xi) -2*(1-xi).^3+xi.^3+6*(1-xi);
b3 = @(xi) (1-xi).^3-2*xi.^3+6*xi;
b4 = @(xi) xi.^3;
                                              % compute Tc as weighted sum
Tc(valid) = TP(p-1).*b1(xi) + TP(p).*b2(xi) + TP(p+1).*b3(xi) + TP(p+2).*b4(xi);
```

The above example points out one of the drawbacks of the spline approach; see also Exercise 3.8. Although the data is constant on the interval $[1.5, 3.5]$, the spline is not. The interpolant has an oscillatory behavior also known as ringing. Techniques for overcoming this drawback are discussed in Section 3.6.

3.4.2 Spline Interpolation for Higher-Dimensional Data

A Kronecker product approach is used for higher dimensions. Here, (3.9) is replaced by

$$\mathcal{T}(x) = \mathcal{T}^{\text{spline}}(x) = \sum_{j_d=1}^{m^d} \cdots \sum_{j^1=1}^{m^1} c_{j^1,\ldots,j^d} b^{j^1}(x^1) \cdots b^{j^d}(x^d). \tag{3.13}$$

Example 3.9 (Computing 2D Spline Coefficients)
For $d = 2$, the interpolation condition reads

$$\texttt{dataT}(j^1, j^2) = \mathcal{T}(x^c_{j^1,j^2}) = \sum_{k^2=1}^{m^2} \sum_{k^1=1}^{m^1} c_{k^1,k^2} b^{k^1}(\xi^1_{j^1}) b^{k^2}(\xi^2_{j^2}),$$

since $x^c_j = [\xi^1_{j^1}, \xi^2_{j^2}]$. With the matrices $B_{m^i} = [b^k(\xi^1_j)]^{m^i}_{k,j=1}$ as introduced in (3.11), this can be rewritten as

$$\texttt{dataT}(j^1, j^2) = \sum_{k^2=1}^{m^2} \sum_{k^1=1}^{m^1} c_{k^1,k^2} B_{m^1}(k^1, j^1) B_{m^2}(k^2, j^2).$$

Using the lexicographical ordering (cf. (3.6)) $j = j^1 + (j^2 - 1)m^1$, $k = k^1 + (k^2 - 1)m^1$, $j, k = 1, \ldots, n = m^1 m^2$, and the Kronecker product $B_m = B_{m^2} \otimes B_{m^1} \in \mathbb{R}^{n,n}$, i.e.,

$B_m(j,k) = B_{m^1}(j^1,k^1)B_{m^2}(j^2,k^2)$, the above interpolation condition reads

$$\texttt{dataT}(j) = \sum_{k=1}^{n} B_m(j,k)c_k, \quad j = 1,\ldots,n, \quad \text{or simply} \quad \texttt{dataT} = B_m\ c.$$

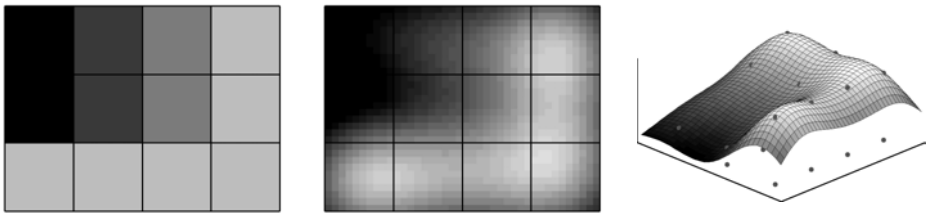
This formula provides a way of computing the spline coefficients $c = B_{m^1}^{-1}\ \texttt{dataT}\ B_{m^2}^{-1}$; see, e.g., [77].

Example 3.10 (Spline Interpolation in Two Dimensions)
This example continues Example 3.7. The following code illustrates how to use the spline interpolation in two dimensions; see also `splineInter2D` and Figure 3.8.

This file is `E3_splineInterpolation2D.m`

```
%%%
dataT = flipud([1,2,3,4;1,2,3,4;4,4,4,4])';
m     = size(dataT);
omega = [0,m(1),0,m(2)];
M     = {m,10*m};   % two resolutions
xc    = reshape(getCenteredGrid(omega,M{1}),[],2);
xf    = reshape(getCenteredGrid(omega,M{2}),[M{2},2]);

B  = @(i) spdiags(ones(m(i),1)*[1,4,1],[-1:1],m(i),m(i));
T  = B(1)\dataT/B(2);
Tc = book_splineInter2D(T,omega,xc(:));
Tf = book_splineInter2D(T,omega,xf(:));

clf;
ph  = plot3(xc(:,1),xc(:,2),Tc(:),'ro');    hold on;
qh = surf(xf(:,:,1),xf(:,:,2),reshape(Tf,M{2}));
```

The general case is a straightforward extension of the above example. Using a lexicographical ordering and the matrix $B_m = B_{m^d} \otimes \cdots \otimes B_{m^1}$, the interpolation condition yields $T(\texttt{xc}) = B_m(\texttt{xc})\ c$.

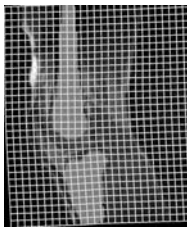


(a) on data grid (b) on finer grid (c) 3D view

Figure 3.8: *Spline interpolation of the data of Example 3.7 (a particular colormap has been used in* (a) *and* (b)*).*

3.5 Derivatives of Interpolation Schemes

Fast optimization schemes to be discussed later rely on derivatives. Since the transformed images are a central ingredient of the objective function, derivatives of the



interpolants are inevitable. This section focuses on spline interpolation and provides derivatives of single- and multivariate input. Moreover, a procedure for testing implementations of derivatives is supplied.

3.5.1 Derivatives of Interpolants

So far, an image is a function in $x \in \mathbb{R}^d$ defined by an interpolation scheme,

$$\mathcal{T}(x) = \texttt{inter(T,omega,xc)},$$

where `T` denotes the coefficients. All the interpolants used in FAIR are Kronecker products of 1D basis functions,

$$\mathcal{T}(x) = \sum_{j^d=1}^{m^d} \cdots \sum_{j^1=1}^{m^1} c_j \; b^{j^1}(x^1) \cdots \; b^{j^q}(x^q) \; \cdots b^{j^d}(x^d). \tag{3.14}$$

Therefore

$$\partial_q \mathcal{T}(x) = \sum_{j^d=1}^{m^d} \cdots \sum_{j^1=1}^{m^1} c_j \; b^{j^1}(x^1) \cdots \; (b^{j^q})'(x^q) \; \cdots b^{j^d}(x^d), \tag{3.15}$$

where $(b^q)'$ can be computed from (3.8.) For an example of spline interpolation with analytic derivatives, see `splineInter2D`.

3.5.2 Derivatives of Multivariate Interpolants

In the FAIR framework, the interpolation is typically evaluated for an ensemble of points $x_j = [x_j^1, \ldots, x_j^d]$, $j = 1, \ldots, n$. Gathering these points as

$$\texttt{xc} = [x_1^1; \ldots; x_n^1; \quad \ldots; \quad x_1^d; \ldots; x_n^d],$$

one is interested in $T : \mathbb{R}^{nd} \to \mathbb{R}^n$ with jth component $T_j(\texttt{xc}) = \mathcal{T}(x_j) = \mathcal{T}(x_j^1, \ldots, x_j^d)$ and the derivative is an n-by-nd matrix which is also known as the Jacobian of T, where

$$\mathrm{d}T(\texttt{xc}) = \left[\frac{\partial T_j(\texttt{xc})}{\partial \texttt{xc}(k)} \right]_{j=1,\ldots,n,\ k=1,\ldots,nd}.$$

Since the jth component of T depends only on $x_j = [x_j^1, \ldots, x_j^d]$, the Jacobian is a block matrix with diagonal blocks.

Example 3.11 (The Format of a 2D Interpolation Derivative)
The derivative of a 2D interpolation scheme evaluated for n points is

$$\mathrm{d}T(\texttt{xc}) = \begin{bmatrix} \partial_1 \mathcal{T}(x_1^1, x_1^2) & & & \partial_2 \mathcal{T}(x_1^1, x_1^2) & & \\ & \ddots & & & \ddots & \\ & & \partial_1 \mathcal{T}(x_n^1, x_n^2) & & & \partial_2 \mathcal{T}(x_n^1, x_n^2) \end{bmatrix}.$$

3.5.3 Testing Implementations of Derivatives

One of the many traps in optimization is working with an erroneous derivative. The following test provides a simple way of checking the implementation of a derivative. To this end, let f be a multivariate function $f : \mathbb{R}^n \to \mathbb{R}$ and let $v \in \mathbb{R}^n$ be an arbitrary vector in the Taylor expansion

$$f(x + hv) = f(x) + h\ df(x)\ v + \mathcal{O}(h^2).$$

A matrix A is the derivative of f if and only if the difference

$$\|f(x + hv) - f(x) - hAv\|$$

is essentially quadratic in h. The function `checkDerivative` computes this difference and visualizes the results; see Figure 3.9 for a typical result.

Example 3.12 (Testing a Derivative Implementation)
The following call checks the derivative of the spline interpolation scheme.

This file is `E3_checkDerivative.m`

```
%%%
E3_splineInterpolation2D;
fctn = @(x) splineInter2D(T,omega,x);
[fig,ph,th] = checkDerivative(fctn,xf(:));
```

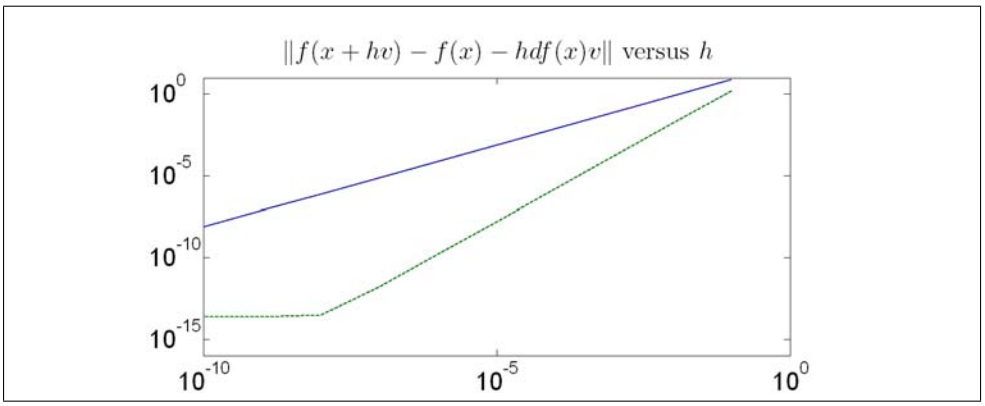
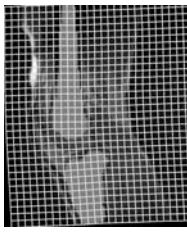


Figure 3.9: *A typical result of `checkDerivative`: linear decay (solid line) for $\|f(y + hv) - f(y)\|$ and quadratic decay (dashed line) for $\|f(y+hv) - f(y) - hdf(y)\|$ up to machine precision shown on a logarithmic scale.*

3.6 Multiscale Spline Interpolation

This section is intended for experienced readers and may be skipped in a first reading.



In practice, the data to be interpolated is contaminated by noise, and thus it is questionable whether to ask the interpolant for a perfect fit of the data. Instead of concentrating on a data fit, one can compromise between data fitting and smoothness of the interpolant. For example, the interpolant in Example 3.8 shows unreasonable oscillations and ringing artifacts. The idea is to replace the strong interpolation condition by a more relaxed one. A continuous parameter $\theta > 0$ controls the balance between data fitting and smoothness. This procedure is commonly known as Tychonoff regularization. Setting $\theta = 0$ yields the interpolation problem (3.17), whereas setting $\theta = \infty$ yields a very smooth solution. If the noise level of the data is known a priori, the weighting can be chosen accordingly; see [111, 203].

The basic idea is explained for 1D data and the linearized bending energy (3.7). Using Kronecker products, the extension to higher dimensions is straightforward.

3.6.1 Multiscale Interpolation in One Dimension

Given the measurements $\texttt{dataT}(j)$ at some spatial positions x_j, $j = 1, \ldots, m$, the interpolation problem may be phrased in the following fashion. Find a function $\mathcal{T} : \mathbb{R} \to \mathbb{R}$, such that $\mathcal{D}[\mathcal{T}] = 0$, where

$$\mathcal{D}[\mathcal{T}] := \|T(\texttt{xc}) - \texttt{dataT}\|_{\mathbb{R}^m}^2 = \sum_{j=1}^{m} (\mathcal{T}(x_j) - \texttt{dataT}(j))^2. \tag{3.16}$$

As already discussed, this problem has many solutions (piecewise constant, linear, spline, etc.) and thus additional conditions are necessary to specify a particular solution. For example, it can be meaningful to pick the solution with minimal bending energy. Using the approximation $\mathcal{S} = \int_\Omega (\mathcal{T}''(x))^2 \, dx$ (cf. (3.7)), the spline interpolation problem can also be phrased as follows. Find a function $\mathcal{T}$ such that

$$\mathcal{S}[\mathcal{T}] = \min \quad \text{subject to} \quad \mathcal{D}[\mathcal{T}] = 0, \tag{3.17}$$

and the solution is a spline, i.e., permits an expansion of the type (3.9). This approach has a strong focus on the data fitting term $\mathcal{D}$ which appears as a constraint in the optimization. A more relaxed version reads

$$\mathcal{D}[\mathcal{T}] + \theta \mathcal{S}[\mathcal{T}] = \min \quad \text{where} \quad \mathcal{T} \text{ is a spline.} \tag{3.18}$$

Since it is known that the solutions of (3.17) and (3.18) are splines anyway, the restriction to a spline space is not severe; see [97, 162] for a more extended discussion. The parameter $\theta > 0$ is a weighting factor balancing between data fitting and smoothness. Setting $\theta = 0$ yields the interpolation problem (3.17), whereas setting $\theta = \infty$ yields a very smooth solution. Moreover, if the noise level of the data is known, the weighting can be chosen accordingly; see [203].

Since the function $\mathcal{T}$ is parameterizable, the computations can be performed in parameter space. The expansion $\mathcal{T}(x) = \sum_{j=1}^{m} c_j b^j(x)$ yields

$$\mathcal{D}[\mathcal{T}] = \|Bc - \texttt{dataT}\|_{\mathbb{R}^m}^2 \tag{3.19}$$

(cf. (3.11)) and the regularizer $\mathcal{S}$ results in a weighted norm

$$\mathcal{S}[\mathcal{T}] = \|c\|_M^2 = c^\top M c, \quad \text{where} \quad M_{j,k} = \int_\Omega (b^j)''(b^k)'' \, dx. \tag{3.20}$$

An even more general approach is to replace M by an arbitrary symmetric, positive semidefinite weighting matrix W. The unique solution for both (3.17) and (3.18) is given by

$$(B^\top B + \theta W)c = B^\top \texttt{dataT}. \tag{3.21}$$

The particular choice $W = M$ is theoretically justified since the optimization is in the spline space (function minimizing the bending energy). However, one may argue about small coefficients ($W = I$, the so-called Tychonoff regularization) or small varying coefficients ($W = D^\top D$, Tychonoff–Phillips regularization), where D approximates a derivative,

$$D = \begin{bmatrix} -1 & 1 & & \\ & \ddots & \ddots & \\ & & -1 & 1 \end{bmatrix} \in \mathbb{R}^{m-1,m}.$$

Example 3.13 (Multiscale Spline Approximations in One Dimension) This example continues Example 3.8. Results for different choices of W and $\theta = 1, 10, 100$ are shown in Figure 3.10. A reference is the interpolant (coefficients $c = B^{-1}\texttt{dataT}$) with B from (3.11) and dataT denoting the data. For the computation of M see Exercise 3.7.

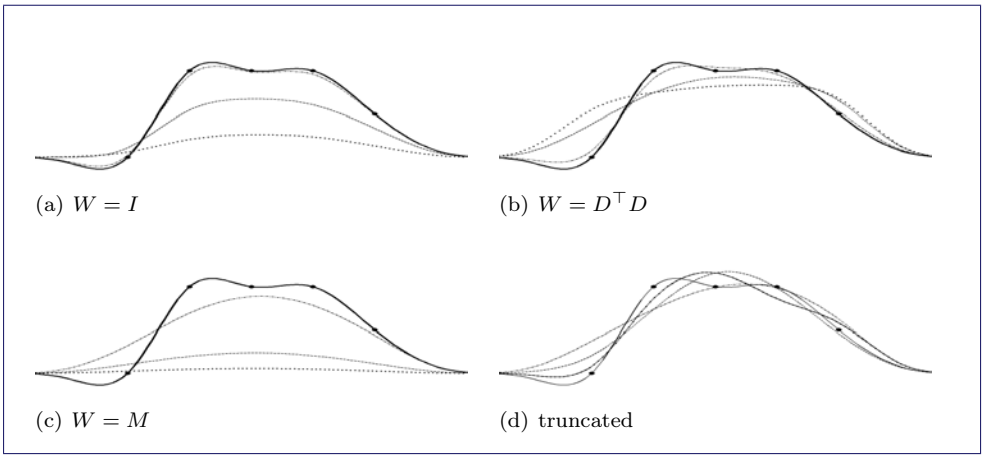
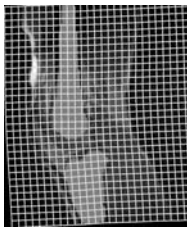


Figure 3.10: *Spline approximations with different regularization and varying θ's, $\theta = 0$ (thick solid line), $\theta = 1$ (dashed line), $\theta = 10$ (dashed-dotted line), and $\theta = 100$ (dotted line), see Example 3.13.* (d) *shows results for the truncated regularization; see Example 3.14.*



This file is E3_MS_splineInterpolation1D.m

```
%%%
function varargout = E3_MS_splineInterpolation1D(regularizer)
if nargin == 0, regularizer = 'I';  end;                  % default value
dataT  = [0,2,2,2,1]'; m = length(dataT); omega = [0,m];  % initialize some data
xc = linspace(omega(1)-1,omega(2)+1,101);                 % fine discretization
Tc = @(T) book_splineInter1D(T,omega,xc);                 % spline interpolant
B  = spdiags(ones(m,1)*[1,4,1],[-1:1],m,m);               % spline basis
D  = spdiags(ones(m,1)*[-1,1],[0,1],m-1,m);               % derivative operation
M  = toeplitz([96,-54,0,6,zeros(1,m-4)]);                 % second derivative
switch regularizer,                                  % initialize regularization
  case 'I', W  = speye(m,m);                         % identity,   Tychonoff
  case 'D', W  = D'*D;                               % derivative, Tychonoff-Phillips
  case 'M', W  = M;                                  % bending operator
end;
c  = @(theta) (B'*B+theta*W)\(B'*dataT);  % coefficients as function in  theta
ph(1:2) = plot(getCenteredGrid(omega,m),dataT,'.k',xc,Tc(B\dataT),'k-');  hold on;
ph(3:5) = plot(xc,Tc(c(1)),'k-.',xc,Tc(c(19)),'k--',xc,Tc(c(100)),'k-');  hold off;
if nargout ~= 0; varargout = {ph}; end;
```

From the registration perspective, the differences in the choices of W are minor, although choosing $W = I$ leads to a stronger reduction of the function values. Changing θ from low to high continuously degrades the function from the interpolant to a straight line.

3.6.2 Truncating High Frequencies

The approaches of the previous section allow a continuous smoothing of the image $\mathcal{T}$, where the smoothing is controlled by a parameter $\theta \in \mathbb{R}$. This section discusses a degradation by basically cutting off the highest k frequencies in the representation of $\mathcal{T}$. In contrast to smoothing with θ, a truncation of k frequencies is an intrinsically discrete process, i.e., $k \in \mathbb{N}$. Therefore, this is not a scale-space approach in the classical sense. However, this approach is presented in this section since it only modifies the coefficients in the representation of the function and does not change the data.

The matrix B in (3.11) is diagonalized by a discrete sine transform; see, e.g., [162]. With

$$V = \sqrt{\frac{2}{m+1}} \left[\sin \tfrac{ij\pi}{m+1} \right]_{i,j=1}^{m} \in \mathbb{R}^{m,m}, \tag{3.22}$$

$$D = 4 + 2\mathrm{diag}(\cos(i\pi/(m+1)), i = 1, \ldots, m) \in \mathbb{R}^{m,m}, \tag{3.23}$$

it holds that $B = VDV^\top$ and $V^\top V = I$ and $D\hat{c} = V^\top \texttt{dataT}$ with $c = V\hat{c}$. The coefficients c are thus a linear combination of the columns of V, which might be viewed as oscillations where the frequency increases with the column index; see Figure 3.11 for an example. A smooth coefficient vector and thus a smooth approximation can be obtained by replacing coefficients of $\hat{c} = D^{-1}V^\top \texttt{dataT}$ which belong to a higher frequency by zero; i.e.,

$$c = V\hat{c}^k, \qquad \hat{c}^k = \begin{cases} \hat{c}_j, & j \le k, \\ 0, & j > k. \end{cases}$$

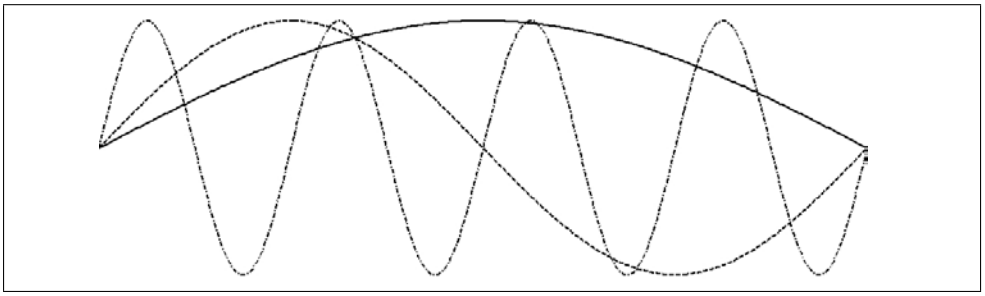


Figure 3.11: *Oscillations: jth column of V ($m = 8$), $j = 1$ (solid line), $j = 2$ (dashed line), and $j = 8$ (dashed-dotted line).*

Example 3.14 (Truncated Spline Approximations in One Dimension) This example continues Example 3.13. Figure 3.10 also shows results of the following program.

This file is `E3_truncatedSplineInterpolation1D.m`

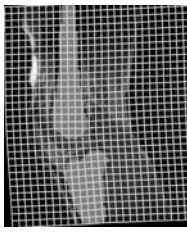
```
%%%
dataT  = [0,2,2,2,1]'; omega = [0,5]; m = length(dataT);  % initialize some data
xc = linspace(omega(1)-1,omega(2)+1,101);                 % fine discretization
Tc = @(T) book_splineInter1D(T,omega,xc);                 % spline interpolant
B  = spdiags(ones(m,1)*[1,4,1],[-1:1],m,m);               % spline basis
% D  = spdiags(ones(m,1)*[-1,1],[0,1],m-1,m);             % derivative operation
% M  = toeplitz([96,-54,0,6,zeros(1,m-4)]);               % second derivative

clf; ph = plot(getCenteredGrid(omega,m),dataT,'.k',xc,Tc(B\dataT),'k-'); hold on;
d = 1./(4+2*cos((1:m)'*pi/(m+1)));
V = sqrt(2/(m+1))*sin((1:m)'*(1:m)*pi/(m+1));
style = {'-','--','-.','.','-'}; col   = 'krgbmc';
for q=2:4,
  c= V'*diag([d(1:q);zeros(m-q,1)])*(V*dataT);
  ph(q+1) = plot(xc,Tc(c),style{q},'color',col(q),'linewidth',1.5);
end;
```

3.6.3 Multiscale Interpolation in Higher Dimensions

Fortunately, the general case is a straightforward extension of the 1D situation when using Kronecker products. Note that $B = B_d \otimes \cdots \otimes B_1$ and suppose $W = W_d \otimes \cdots \otimes W_1$ and θ are given. Setting

$$\begin{array}{lll} M_i = B_i & \text{and } P_i = I & \text{for interpolation or} \\ M_i = B_i^\top B_i + \theta W_i & \text{and } P_i = B_i & \text{for the multiscale approach,} \end{array}$$

the coefficients are basically given by $c = M_i^{-1}\, P_i$ `dataT`. However, in order to efficiently perform these computations in higher dimensions, some reorganization of the data is necessary. The steps for the multiscale approach for dimensions $1, 2$, and 3 are summarized as follows:



1D: $c = M_1^{-1}\ P_1\ \texttt{dataT} \in \mathbb{R}^{m^1}$.

2D: $c = M_1^{-1}\ P_1\ \texttt{dataT}\ P_2\ M_2^{-1} \in \mathbb{R}^{m^1,m^2}$, note that $(A \otimes B)\vec{X} = (AXB^\top)^{\rightarrow}$.

3D: Unfortunately, there is no matrix-times-vector analogue for dimensions $d > 2$ in MATLAB. This inconvenience is bypassed by a tiresome reordering of the data. Running over all three directions, the remaining two directions are condensed. After applying the operation to the 2D presentation, the data is retransformed. For example, let

$$\texttt{dataT}(:,:,1) = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix} \quad \text{and} \quad \texttt{dataT}(:,:,2) = \begin{bmatrix} 7 & 10 \\ 8 & 11 \\ 9 & 12 \end{bmatrix}.$$

The operation for the first direction can be mimicked by (a) reorganizing the data in a 3-by-4 array, (b) doing the computation as matrix-times-matrix, and (c) reorganizing the data back to 3-by-2-by-2:

$$\texttt{dataT} = \texttt{reshape(T,m(1),m(2)*m(3))} = \begin{bmatrix} 1 & 4 & 7 & 10 \\ 2 & 5 & 8 & 11 \\ 3 & 6 & 9 & 12 \end{bmatrix},$$

$$\texttt{dataT} = \texttt{reshape(}M_1^{-1}P_1\texttt{T,m)}.$$

The operation for the second direction can be mimicked by (a) making the second direction the leading direction, (b) reorganizing the data in a 2-by-6 array, (c) doing the computation as matrix-times-matrix, and (d) reorganizing the data back to 3-by-2-by-2:

$$\texttt{dataT} = \texttt{permute(T,[2,1,3])},$$
$$\texttt{dataT}(:,:,1) = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}, \qquad \texttt{dataT}(:,:,2) = \begin{bmatrix} 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix},$$
$$\texttt{dataT} = \texttt{reshape(permute(T,[2,1,3]),m(2),m(1)*m(3))} = \begin{bmatrix} 1 & 2 & 3 & 7 & 8 & 9 \\ 4 & 5 & 6 & 10 & 11 & 12 \end{bmatrix},$$
$$c = \texttt{permute(reshape(}M_2^{-1}P_2\ \texttt{T,m([2,1,3])),[2,1,3])}.$$

The operation for the third direction is along the same lines:

$$\texttt{dataT} = \texttt{permute(T,[3,1,2])},$$
$$\texttt{dataT}(:,:,1) = \begin{bmatrix} 1 & 2 & 3 \\ 7 & 8 & 9 \end{bmatrix}, \qquad \texttt{dataT}(:,:,2) = \begin{bmatrix} 4 & 5 & 6 \\ 10 & 11 & 12 \end{bmatrix},$$
$$\texttt{dataT} = \texttt{reshape(permute(T,[3,1,2]),m(3),m(1)*m(2))} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 7 & 8 & 9 & 10 & 11 & 12 \end{bmatrix},$$
$$c = \texttt{permute(reshape(}M_3^{-1}P_3\texttt{T,m([3,1,2])),[2,3,1])}.$$

Example 3.15 (Kronecker Product in MATLAB)
The following code provides a compact, though hard to digest, alternative to the above statements.

This file is book_kron3D.m

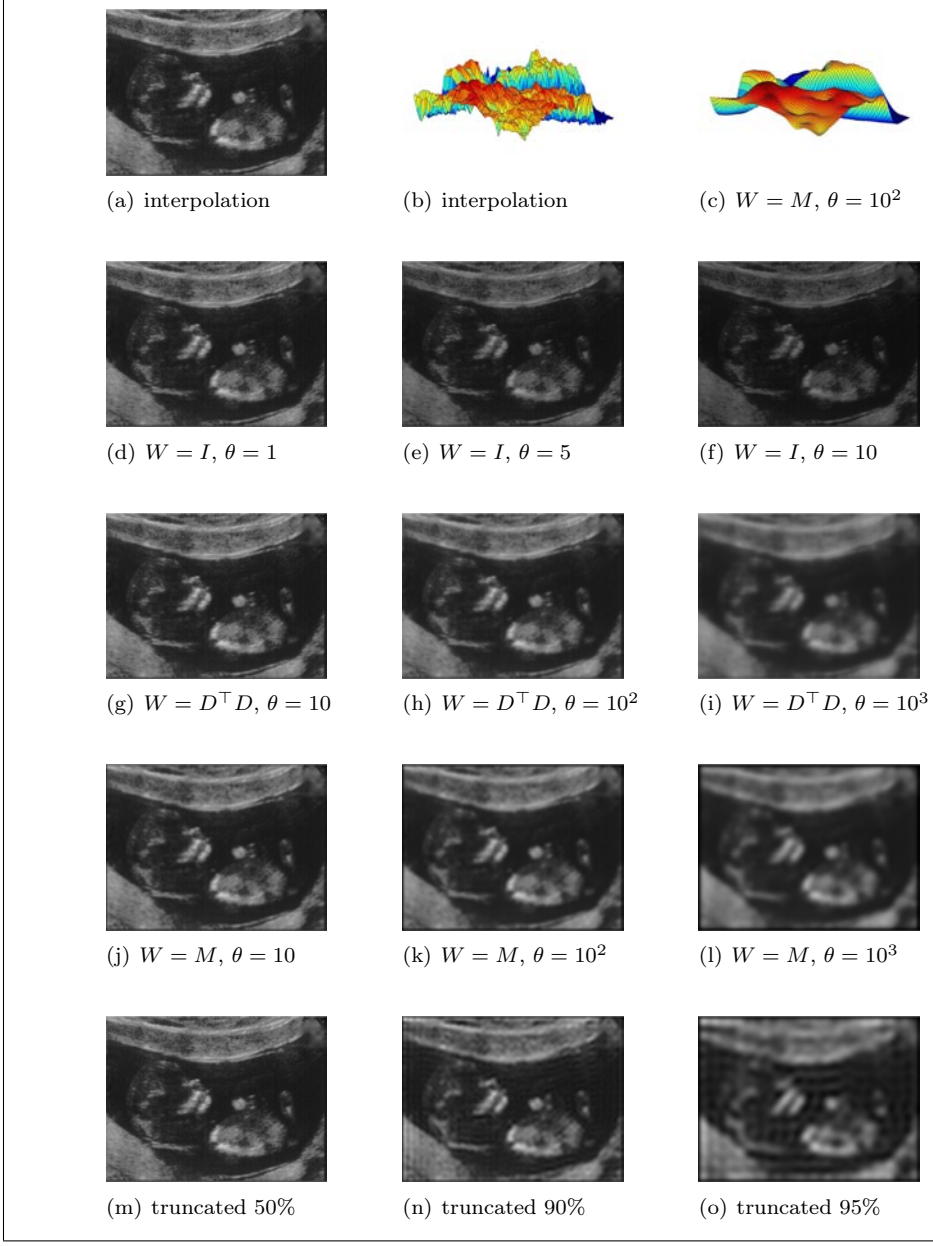
```
%%
% memory efficient implementation of
% reshape(Mx,m), Mx = kron(M{1},kron{M{2},M{3})) * x(:)

x = reshape(x,m); % make sure x is a 3d-array of size m

% given a permutation L of [1,2,3], the following function
% 1. permutes x, such that j=J(1) is the first dimension
% 2. reshapes the permuted x, such that it is m(j)-by-M(J(2))*m(J(3))
% 3. multiplies this by M which is assumed to be m(j)-by-m(j)
% 4. undoes the reshape to make the result m(j)-by-m(J(2))-ny-m(J(3))
% 5. undoes the permute

operate = @(M,x,L) ipermute( reshape( ...
  M*reshape( permute(z,L), m(L(1)),[]), m(L)), L);
% run over all directions
for ell=1:3,
  % make the ell-th component the first
  L = [ell,setdiff(1:3,ell)];
  % operate as indicated above
  x = operate(M{ell},x,L);
end;
```

Example 3.16 (Smoothing Spline Approximation in Two Dimensions)
The 2D US image from Example 2.6 serves as data for the following experiments. The following code shows how to use the spline approximation scheme. The main ingredients are getSplineCoefficients (works for $d = 1, 2, 3$) and splineInter2D; see also Figure 3.12.

This file is E3_MS_splineInterpolation2D.m

```
%%%
setupUSData; m = 128*[3,2]; xc = getCenteredGrid(omega,m);
T = getSplineCoefficients(dataT,'dim',2,'regularizer','gradient','theta',50);
figure(2); clf; viewImage2D(splineInter2D(T,omega,xc),omega,m);
colormap(gray(256));
```

The above example affirms the observation from the 1D example: The differences between the choices of W are minor; enlarging θ removes more and more details.



(a) interpolation (b) interpolation (c) $W = M$, $\theta = 10^2$

(d) $W = I$, $\theta = 1$ (e) $W = I$, $\theta = 5$ (f) $W = I$, $\theta = 10$

(g) $W = D^\top D$, $\theta = 10$ (h) $W = D^\top D$, $\theta = 10^2$ (i) $W = D^\top D$, $\theta = 10^3$

(j) $W = M$, $\theta = 10$ (k) $W = M$, $\theta = 10^2$ (l) $W = M$, $\theta = 10^3$

(m) truncated 50% (n) truncated 90% (o) truncated 95%

Figure 3.12: *Spline approximations in two dimensions with different regularization W and varying θ's; (b) and (c) show 3D views of a detail in the image; image from Example 2.6.*

3.7 Multilevel Representation of Data

A different approach to smoothing is obtained by a reinterpretation of the data acquisition. Suppose that the measurement $\texttt{dataT}(j)$ is the average light intensity on a cell c_j plus some noise. A smoothed measurement can then be obtained by averaging adjacent cells. This results in representing the displayed object on different levels.

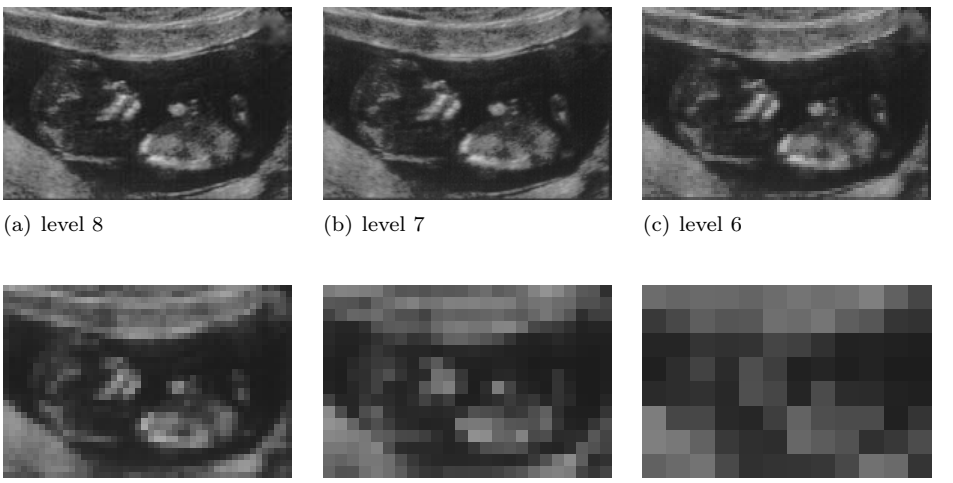
Example 3.17 (Multilevel Representation of 1D Data)
Suppose that $m = 2^L$, $L \in \mathbb{N}$, and $\texttt{dataT} \in \mathbb{R}^m$ is given. A multilevel representation of the data is $\{\texttt{T}^\ell, \ell = 0, \ldots, L\}$, where $\texttt{T}^L = \texttt{dataT}$ and for $\ell = L : -1 : 1$, $\texttt{T}^{\ell-1} = (\texttt{T}^\ell(1\colon 2\colon m-1) + \texttt{T}^\ell(2\colon 2\colon m))/2$.

Example 3.18 (Multilevel Representation of 2D Data)
Suppose that $m^1 = m^2 = 2^L$ with $L \in \mathbb{N}$ and $\texttt{dataT} \in \mathbb{R}^{m^1,m^2}$ is given. A multilevel representation of the data is $\{\texttt{T}^\ell, \ell = 0, \ldots, L\}$, where $\texttt{T}^L = \texttt{dataT}$ and for $\ell = L : -1 : 1$,

$$\begin{aligned}\texttt{T}^{\ell-1} = (&\texttt{T}^\ell(1\colon 2\colon m^1-1, 1\colon 2\colon m^2-1) + \texttt{T}^\ell(2\colon 2\colon m^1, 1\colon 2\colon m^2-1)\\ &+ \texttt{T}^\ell(1\colon 2\colon m^1-1, 2\colon 2\colon m^2) + \texttt{T}^\ell(2\colon 2\colon m^1, 2\colon 2\colon m^2))/4.\end{aligned}$$

A visualization of the multilevel representation for the US image (cf. Example 2.6) is given in Figure 3.13.

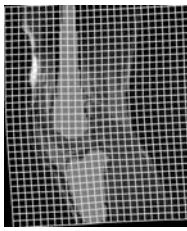
The objective of the multiscale and multilevel approaches is to derive a family of continuous models for the data. The multiscale approach adds a new dimension



(a) level 8 (b) level 7 (c) level 6

(d) level 5 (e) level 4 (f) level 3

Figure 3.13: *Multilevel representation of an ultrasound image.*



and considers the function $\mathcal{T}(x,\theta)$, where θ is a scale parameter. For $\theta = 0$, the data can be recovered, i.e., $\mathcal{T}(\mathtt{xc},0) = \mathtt{dataT}$, whereas choosing a large θ yields a smooth function. In contrast to the continuous multiscale approach, the multilevel approach is based on a finite number of interpolants $\mathcal{T}_\ell$ based on data $(\mathtt{xc}^\ell, T^\ell)$, where the discrete parameter ℓ ranges from a coarse to a fine level. FAIR provides `getMultilevel` to compute a multilevel representation; see FAIR 3 (this page).

FAIR 3: Multilevel Generation

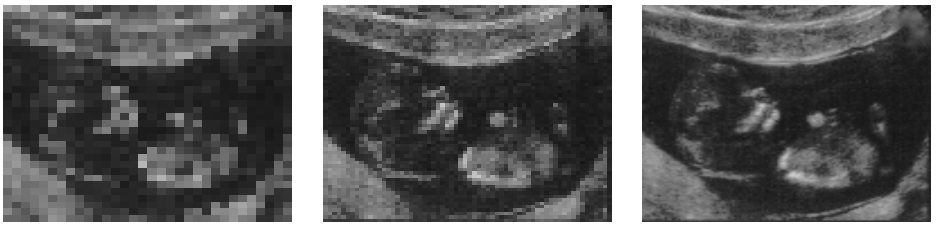
Given data `{T,R}` on a domain specified by `omega`, this function computes a multilevel representation `MLdata` of the data. The ℓth component of `MLdata` is a structure with fields names `{'m','omega','T','R'}`, where `m` is the size of the data on the ℓth level, `omega` is the domain specification, and `{T,R}` are the representations of the data on the ℓth level. The levels run from one to `maxLevel` $= \log_2(\max\{m\})$, with `MLdata{l}=[]` for `l` $<$ `minLevel`, the default for **`minLevel`** being three.

`[MLdata,minLevel,maxLevel,fig]= getMultilevel(IS,omega,m,varargin)`	
`IS`	list of input data, e.g., `{dataT,dataR}`,
`omega`	specifies the domain $\Omega = (\omega^1,\omega^2) \times \cdots \times (\omega^{2d-1},\omega^{2d})$
`m` $\in \mathbb{N}^d$	number of interpolation points on the finest level
`varargin`	optional parameters like `minLevel`, `maxLevel`; defaults: `minLevel` $= 3$, `maxLevel` $= \log_2(\max\{m\})$
`MLdata`	array of structs representing data on levels 1 : `maxLevel`
`minLevel`	first nonempty in `MLdata`, representing coarsest level
`maxLevel`	length of `MLdata`, representing finest level
`fig`	handle to the output figure

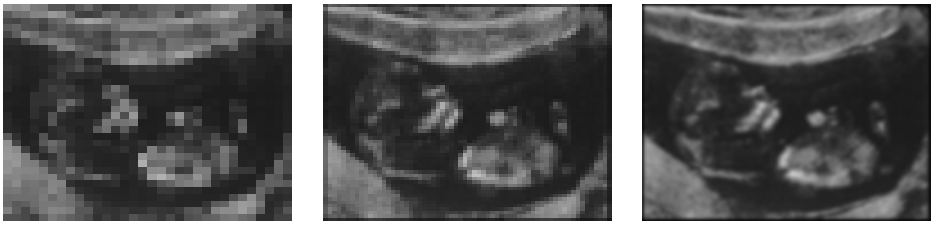
Example 3.19 (Creating a Multilevel Representation of the Data)
The following code illustrates how to create a multilevel representation of given data.

This file is `E3_US_getMultilevel`

```
% Tutorial for FAIR: creating a multilevel representation
% (C) 2008/05/01, Jan Modersitzki, see FAIR and FAIRcopyright.m.
% load USfair.jpg
% creating a multilevel representation using getMultilevel.m
clear, close all, help(mfilename)

% load some data, define a doman and an initial discretization
dataT  = double(imread('USfair.jpg'));
omega  = [0,size(dataT,1),0,size(dataT,2)];
m      = [128,128];

% set-up image viewer
viewImage('reset','viewImage','viewImage2D','colormap','gray(256)');

MLdata = getMultilevel(dataT,omega,m);
disp('MLdata{3}=')
disp(MLdata{3})
```



(a) linear, $m = [32, 32]$ (b) linear, $m = [64, 64]$ (c) linear, $m = [128, 128]$



(d) spline, $m = [32, 32]$ (e) spline, $m = [64, 64]$ (f) spline, $m = [128, 128]$

Figure 3.14: *Interpolation on duty: linear and spline interpolation on a cell-centered grid of dimension m; see Example 3.20.*

3.8 Summarizing the Interpolation Toolbox

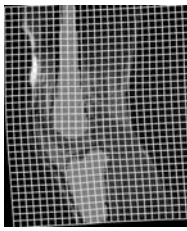
This chapter discusses the continuous viewpoint for images and explains how to derive continuous models $\mathcal{T}$ for an image based on given discrete data $[x_j, \texttt{dataT}(j)]$, $j = 1, \ldots, n$, and $x_j \in \mathbb{R}^d$. In particular, the important linear and spline interpolation schemes for dimensions $d = 1, 2, 3$ have been introduced and discussed.

The interpolation tools discussed in the chapter cover a multilevel approach that can be used to reduce the data size (see Section 3.7) as well as a spline-based multiscale approach that can be used to smooth the function $\mathcal{T}$.

An administrative function `inter` is supplied; see FAIR 4 (p.43). For general options see Section 2.3.7; additional options are as follows:

- `[T,R]= inter('coefficients',dataT,dataR,omega)`: computes the coefficients for the interpolation scheme to be used; for linear interpolation `T=dataT` and `R=dataR` ; for spline interpolation see `getSplineCoefficients`
- `Tc= inter(T,omega,yc)`: returns the function values
- `[Tc,dT]= inter(T,omega,yc)`: returns the function values and the derivative.

Example 3.20 (Linear Interpolation in Two Dimensions)
This example illustrates how the interpolation schemes can be facilitated in a uniform framework. The first loop calls `inter(T,omega,xc)`, where a linear interpolation scheme is used.



FAIR 4: Interpolation Toolbox

Given the data `dataT` on a cell-centered grid `xc` on a domain Ω, this function computes the value of the interpolant for any wanted point $y \in \mathbb{R}^d$. For a collection of n points $\texttt{yc} = [y_1^1; \ldots; y_n^1; \ \ldots; \ y_1^d; \ldots; y_n^d] \in \mathbb{R}^{nd}$, the result is a collection of corresponding function values $\texttt{Tc} = T(\texttt{yc}) = [\mathcal{T}(y_j^1, \ldots, y_j^d)]_{j=1}^n$.
The Jacobian is denoted by

$$\texttt{dT} = \mathrm{d}T(\texttt{yc}) = \left[\frac{\partial T_i(\texttt{yc})}{\partial \texttt{yc}(j)}\right]_{i=1,\ldots,n,\ j=1,\ldots,nd} \in \mathbb{R}^{n,nd}.$$

	`[Tc,dT]= inter(T,omega,yc)`
`T` $\in \mathbb{R}^{m_1,\ldots,m_d}$	the coefficients for a representation of $\mathcal{T}$
`omega`	specifies the domain $\Omega = (\omega^1, \omega^2) \times \cdots \times (\omega^{2d-1}, \omega^{2d})$
`yc` $\in \mathbb{R}^{nd}$	interpolation points
`Tc` $\in \mathbb{R}^{n}$	value of the interpolant at certain locations `yc` $\in \mathbb{R}^{nd}$
`dT` $\in \mathbb{R}^{n,nd}$	derivative of the interpolant

The second loop uses the same standardized call, but the method used is a regularized spline interpolation. The regularization (W = moment matrix and $\theta = 100$) is obtained by changing the coefficients appropriately; see also Figure 3.14. Since the data is not used any further, this memory may also be used to store the interpolation coefficients. Note that for linear interpolation, data and coefficients are the same anyway.

This file is `E3_interpolation2D.m`

```
%%%
setupUSData; close all; T = dataT; xc = @(m) getCenteredGrid(omega,m);
inter('set','inter','linearInter2D');

for p=5:7,
  m = 2^p*[1,1];
  Tc = inter(T,omega,xc(m));
  figure(p-4); viewImage2D(Tc,omega,m); colormap(gray(256));
end;

inter('set','inter','splineInter2D');
T = getSplineCoefficients(dataT,'regularizer','moments','theta',100);
for p=5:7,
  m = 2^p*[1,1];
  Tc = inter(T,omega,xc(m));
  figure(p-1); viewImage2D(Tc,omega,m); colormap(gray(256));
end;
```

3.9 FAIR Tutorials on Interpolation

FAIR contains the tutorial `BigTutorialInter` which summarizes a number of smaller tutorials that provide insight into the numerical realizations of the concepts discussed in this chapter.

BigTutorialInter	
E3_1D_basics	1D basic interpolation example
E3_1D_scale	1D multiscale example
E3_1D_derivatives	1D check the derivative example
E3_2D_basics	2D basic interpolation example
E3_Hands_ij2xy	2D data example, $(i,j) \leftrightarrow (x,y)$, multilevel
E3_viewImage	2D visualize data
E3_setupHandData	load data (with landmarks) and visualize
E3_2D_scale	2D multiscale example
E3_2D_generic	2D high-res and low-res representation
E4_US_trafo	2D US, rotate image
E3_2D_derivative	2D check the derivative example

3.10 Exercises

Exercise 3.1
Replacing $\xi_j^i = \omega^{2i-1} + (j-0.5)h^i$, $j = 1,\ldots,m^i$, *by* $\eta_j^i = \omega^{2i-1} + (j-1)h^i$, $j = 1,\ldots,m^i+1$, *one obtains the so-called* nodal grid. *Write a function* `getNodalGrid` *with input* `omega` *and* `m` *returning a nodal grid. Start with* $d = 1$. *Extend to* $d = 2, 3$.

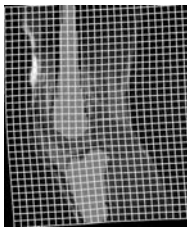
Exercise 3.2
Provide functions `plotCenteredGrid` *and* `plotNodalGrid` *for visualizing cell-centered and nodal grids, respectively.*

Exercise 3.3
Write a function visualizing linearized 3D data `T=dataT`; *use* `reshape` *and visualize*

$$\begin{bmatrix} \texttt{T(:,:,1)} & \texttt{T(:,:,2)} & \cdots & \texttt{T(:,:,p)} \\ \texttt{T(:,:,p+1)} & \texttt{T(:,:,p+2)} & \cdots & \texttt{T(:,:,2p)} \\ \vdots & \vdots & \ddots & \vdots \\ \texttt{T(:,:,k*p+1)} & \texttt{T(:,:,k*p+2)} & \cdots & \texttt{T(:,:,k*p+p)} \\ \texttt{T(:,:,(k+1)*p+1)} & \cdots & \texttt{T(:,:,q)} & 0 \end{bmatrix}.$$

Exercise 3.4
Write a function `Tc=matlabInterpolation(dataT,omega,xc)` *using the MATLAB built-in linear interpolation scheme. Use a finite difference scheme to approximate the derivatives. Check the approximation order of the derivative. Start with* $d = 1$, *extend to* $d = 2, 3$.

Exercise 3.5
Write your own function `Tc=linearInterpolation1D(dataT,omega,xc)`. *Test with scalar input for* `xc`. *Include the derivative as output. Where is the derivative undefined? If this works for* $d = 1$ *(use* `checkDerivative`*), extend to* $d = 2, 3$.



Exercise 3.6
Verify (3.11):

$$\texttt{dataT} = \mathcal{T}^{\text{spline}}(\texttt{xc}) = \sum_{j=1}^{m} c_j b^j(\texttt{xc}) = Bc \quad \textit{with} \quad B = [b^j(\texttt{xc}(i))]_{i,j} = \begin{bmatrix} 4 & 1 & & 0 \\ 1 & \ddots & \ddots & \\ & \ddots & \ddots & 1 \\ 0 & & 1 & 4 \end{bmatrix}.$$

What is to be expected for $d = 2$*?*

Exercise 3.7
Verify the matrix M *in Example* 3.13.

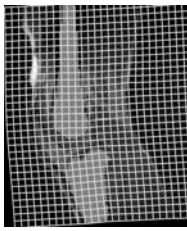
Exercise 3.8
Use the spline interpolation for the data given on a cell-centered grid, `omega` $= (0, 8)$, `m` $= 8$, `dataT=[0;1;1;1;1;1;1;0]`. *Observe that the interpolant* $\mathcal{T}$ *oscillates. Use the Tychonoff regularization to smooth* $\mathcal{T}$*, and use* $W = D^\top D$ *and various values of* θ.

Exercise 3.9
Compute the derivative of $T(\texttt{yc}) =$ `inter(T,omega,yc)` *for* $d = 3$*, i.e.,*

$$\Omega \subset \mathbb{R}^3, \quad \texttt{yc} = [y^1(:); \ldots; y^d(:)] \quad \textit{with} \quad y^i \in \mathbb{R}^{q^1,q^2,q^3}.$$

Exercise 3.10
Generate a movie of smoothed US images, where the time is the regularization parameter θ*; see* `setupUSdata`.

Exercise 3.11
Write a function `getMultilevel(dataT,dataR,omega,m,`$\ell^{\min}, \ell^{\max}$`)` *returning multilevel data* $\{\texttt{T}^\ell, \texttt{R}^\ell, m^\ell\}$, $\ell = \ell^{\min} : \ell^{\max}$, *where* $m^{\ell^{\max}} = m$ *and* $m^{\ell-1} = m^\ell/2$. *Start with* $d = 1$ *and an* m *having components that are multiples of powers of 2. Extend to arbitrary* m*. Extend to* $d = 2, 3$.



Chapter 4

Transforming Images by Parameterized Transformations

The objective of this chapter is to demonstrate the usage of the interpolation toolbox as well as to introduce parametric transformations. A parametric transformation is a function $y : \mathbb{R}^d \to \mathbb{R}^d$, where the components are linear combinations of certain basis functions q^ℓ and the coefficients are basically the parameters w_ℓ. For example, the linear function $y : \mathbb{R} \to \mathbb{R}$ with $y = w_1 x + w_2$ is parameterized by the parameters $w = [w_1; w_2]$ and the basis functions $q^1(x) = x$ and $q^2(x) = 1$. Setting $Q(x) = [q^1(x), q^2(x)]$ yields the compact description $y = Q(x)w$. Choosing a collection `xc` of points to be mapped, the transformed points are obtained by `yc` $= Q($`xc`$)w$ or, using FAIR notation, `yc=Q*wc`, where `Q` $= Q($`xc`$)$ and `wc` $= w$.

The concepts proposed in this chapter are very general and adapt to any spatial dimension d, while the particular implementation of course does depend on d. For ease of presentation, the implementations of different transformations are mainly discussed for $d = 2$, and the extensions to $d = 3$ are left as exercises. Using the interpolation framework presented in Chapter 3, the transformed image is straightforward to compute: `Tc= inter(T,omega,yc)`, where `T` denotes the coefficient for a linear interpolation method, and hence `T` $=$ `dataT`.

Some of the most important parametric transformations are discussed, and important implementation issues are described in the following sections. Among these transformations are the rigid, affine linear, and spline-based transformation, where the latter is also known as free-form deformation; see, e.g., [176]. For a general discussion of parametric transformations in medical image registration see, e.g., [200].

The examples presented in this chapter are for the 2D ultrasound (US) image introduced in Example 2.6; a cell-centered grid `xc` of size m is used and the visualization is performed using `viewImage2D`.

4.1 Translations

One of the simplest transformations is a translation of an image. Let $w = [w_1; w_2] \in \mathbb{R}^2$ denote a translation vector and let

$$y^1 = x^1 + w_1 \quad \text{and} \quad y^2 = x^2 + w_2.$$

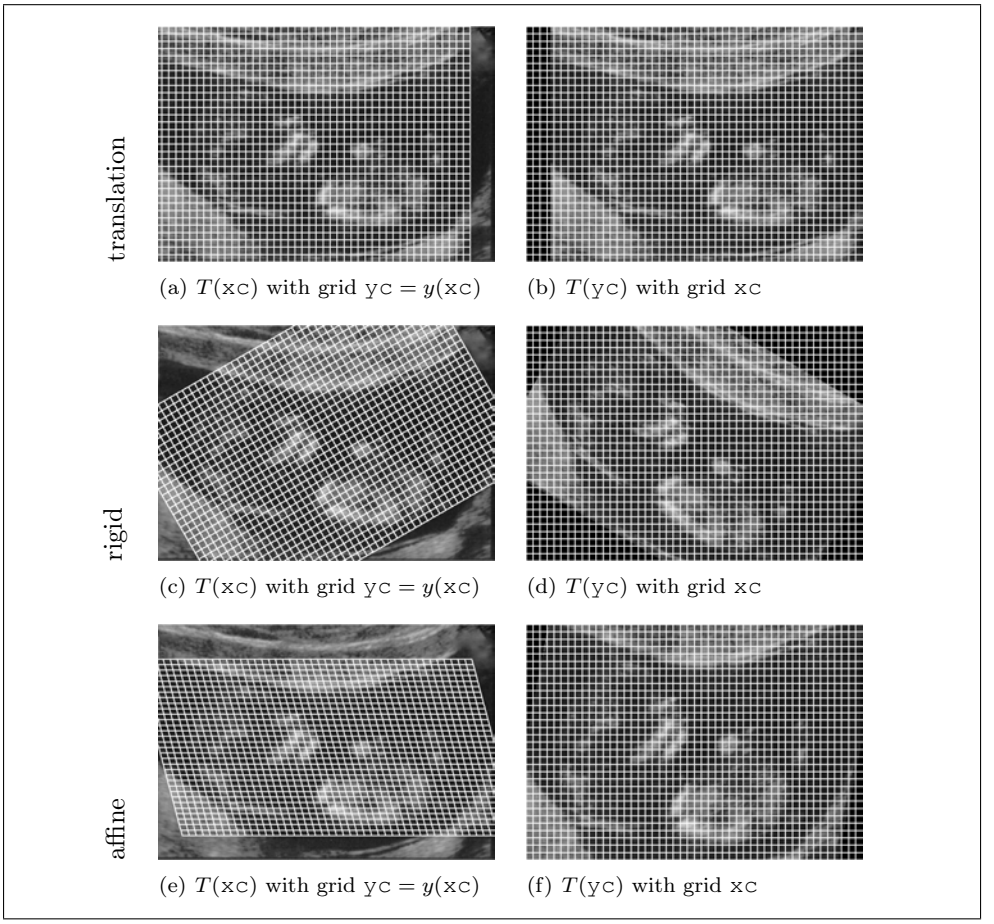


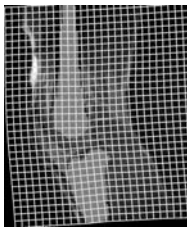
Figure 4.1: *Translation of a US image, rigid, and affine linear transformations.*

With

$$Q(x) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad \text{it holds that} \quad y = x + Q(x)w.$$

Figure 4.1 shows the original image $T(\mathtt{xc})$ overlaid by a part of the transformed grid $\mathtt{yc}$ as indicated by the white lines. The grid has been translated to the left. The figure also shows the transformed image $T(\mathtt{yc})$ overlaid with the original grid $\mathtt{xc}$. Note that since the translation maps $\mathtt{xc}$ to the left, the transformed image appears as being translated to the right.

Example 4.1 (Translations in x^1 Direction)
See also Figure 4.1.



This file is E4_Translation2Dplain.m

```
%%%
setupUSData;
wc = [-50;0];
xc = reshape(getCenteredGrid(omega,m),[],2);
yc = [(xc(:,1) + wc(1));(xc(:,2) + wc(2))];
Tc = linearInter2D(dataT,omega,yc);
figure(2); clf; viewImage2D(Tc,omega,m,'colormap','gray(256)');
```

4.2 Affine Linear Transformations

In addition to translation, an affine linear transformation allows for rotation, shearing, and, in particular, for individual scaling. The components of an affine linear transformation are

$$y^1 = w_1 x^1 + w_2 x^2 + w_3,$$
$$y^2 = w_4 x^1 + w_5 x^2 + w_6,$$

where $w = [w_1; \ldots; w_6] \in \mathbb{R}^6$ parameterizes the transformation. With

$$Q(x) = \begin{bmatrix} x^1 & x^2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x^1 & x^2 & 1 \end{bmatrix}, \quad \text{it holds that} \quad y = Q(x)w. \tag{4.1}$$

Note that for $d = 3$ the matrix $Q(\mathtt{xc})$ may require a nonnegligible amount of memory.

Example 4.2 (Affine Linear Transformations)
See also Figure 4.1.

This file is E4_affine2Dplain.m

```
%%%
setupUSData;
wc = [1 -0.2 50, 0, 0.75 50]';
xc = getCenteredGrid(omega,m);
yc = book_affine2D(wc,xc);
Tc = linearInter2D(dataT,omega,yc);
figure(2); viewImage2D(Tc,omega,m,'colormap','gray(256)');
```

This file is examples/book_affine2D.m

```
%%%function yc = affine2DBook(wc,xc)

xc = reshape(xc,[],2);
yc = [(wc(1)*xc(:,1) + wc(2)*xc(:,2) + wc(3))
      (wc(4)*xc(:,1) + wc(5)*xc(:,2) + wc(6))];
```

4.3 Rigid Transformations

A particular affine transformation is the so-called rigid transformation, which allows only for translations and rotations. The components of the rigid transformation are given by

$$y^1 = \cos(w_1)x^1 - \sin(w_1)x^2 + w_2,$$
$$y^2 = \sin(w_1)x^1 + \cos(w_1)x^2 + w_3,$$

where $w = [w_1; w_2; w_3] \in \mathbb{R}^3$ parameterizes the transformation. Although this function is nonlinear in w, it still allows an expansion $y(x) = Q(x)f(w)$ with Q from (4.1) and $f(w) = [\cos w_1; -\sin w_1; w_2; \sin w_1; \cos w_1; w_3]$. A particular example is given in the next section.

4.4 Rotations About the Domain Center

As an example for a transformation with only one parameter $w \in \mathbb{R}$, a rotation about the center of the domain $c = (\omega^2 - \omega^1, \omega^4 - \omega^3)/2$ is considered. A simple way to perform this transformation is to shift c to the origin, rotate about the origin, and shift back. With

$$R = \begin{bmatrix} \cos w & -\sin w \\ \sin w & \cos w \end{bmatrix},$$

it holds that $(y - c) = R(x - c)$, which results in $y = Rx + (I - R)c$ in the original domain.

Example 4.3 (Rotations About the Domain Center)
See also Figure 4.1.

This file is E4_rigid2Dplain.m

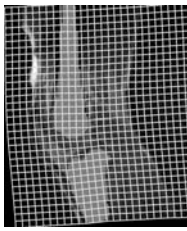
```
%%%
setupUSData; c = (omega(2:2:end)-omega(1:2:end))'/2; alpha = pi/6;
rot = [cos(alpha),-sin(alpha);sin(alpha),cos(alpha)];
wc  = [alpha;(eye(2)-rot)*c];
xc  = getCenteredGrid(omega,m);
yc  = book_rigid2D(wc,xc);
Tc  = linearInter2D(dataT,omega,yc);
figure(2); viewImage2D(Tc,omega,m,'colormap','gray(256)');
```

This file is book_rigid2D.m

```
%%%function yc = rigid2Dbook(wc,xc)

xc = reshape(xc,[],2);
yc = [(cos(wc(1))*xc(:,1) - sin(wc(1))*xc(:,2) + wc(2));
      (sin(wc(1))*xc(:,1) + cos(wc(1))*xc(:,2) + wc(3))];
```

4.5 Spline-Based Transformations

In the above examples, the degrees of freedom in the parameterized transformation are rather small: d parameters for a translation, $2d$ for a rigid, and $d(d+1)$ for an affine linear transformation. As a consequence, the set of transformations is rather small and restrictive. The example to be explained in this section allows for as many parameters as wanted. However, the transformation is still parameterized and not completely arbitrary. The components of the transformation are spanned by splines and the transformation is thus called a spline transformation or free-form



transformation,

$$y^1 = x^1 + \sum_{j^1=1}^{p^1} \sum_{j^2=1}^{p^2} w^1_{j^1,j^2} b^{j^1}(x^1) b^{j^2}(x^2) \text{ and}$$
$$y^2 = x^2 + \sum_{j^1=1}^{p^1} \sum_{j^2=1}^{p^2} w^2_{j^1,j^2} b^{j^1}(x^1) b^{j^2}(x^2),$$

where b^j denotes the splines (see Section 3.4) and w^1 and w^2 denote the coefficients for the first and second component of y, respectively. For ease of presentation, it is assumed that $w^1, w^2 \in \mathbb{R}^p$ with $p = [p^1, p^2]$ the number of coefficients in the spline expansion.

Using the vectorized w^i, $\iota = j^1 + p^1(j^2 - 1)$, and $q^\iota(x) = b^{j^1}(x^1) b^{j^2}(x^2)$, it holds that $y^i = Q(x)w^i = [q^1(x), \ldots, q^{p^1 p^2}(x)]w^i$. Moreover, setting

$$Q^1(x) = [b^1(x^1), \ldots, b^{p^1}(x^1)] \quad \text{and} \quad Q^2(x) = [b^1(x^2), \ldots, b^{p^2}(x^2)],$$

it holds that $Q(x) = Q^2(x^2) \otimes Q^1(x^1)$. This compact but inefficient form is used in the implementation shown below.

Example 4.4 (Spline-Based Transformations)
See also Figure 4.2 and `splineTransformation2D`.

This file is `E4_splineTransformation2D.m`

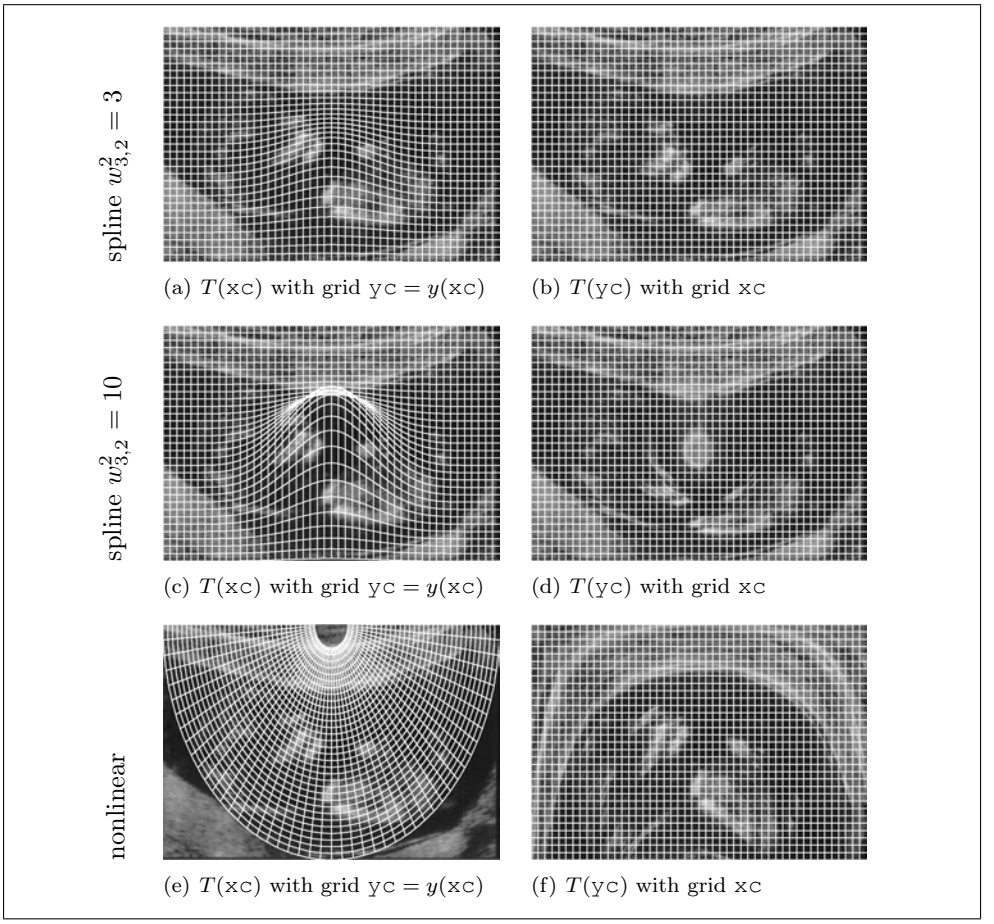
```
%%%
setupUSData; p = [5,4];
xc = getCenteredGrid(omega,m);
splineTransformation2D([],xc,'omega',omega,'m',m,'p',p);
w1 = zeros(p); w2 = zeros(p);  w2(3,2) = 3;
wc = [w1(:);w2(:)];
yc = splineTransformation2D(wc,xc);
Tc = linearInter2D(dataT,omega,yc);
figure(2); viewImage2D(Tc,omega,m,'colormap','gray(256)');
```

The above examples show that transformations with many degrees of freedom can be utilized in this framework. For the example shown, $p = [5, 4]$ and thus 40 parameters can be used to tune the transformation. However, not any choice automatically leads to reasonable transformations. For example, just changing parameter $w^2_{3,2}$ from 3 to 10 results in a transformation that folds the grid; see Figure 4.2. Avoiding unwanted solutions is one of the main problems in registration and is discussed in more depth in Chapter 8 on regularization.

4.6 More Bizarre Transformations

Just in order to show the generality of the concept, we consider $\Omega = (0, \omega_1) \times (0, \omega_2)$ and

$$y^1 = \omega_1((1 - 0.9x^2/\omega_2)\cos(\pi(1 - x^1/\omega_1))/2 + 0.5),$$
$$y^2 = \omega_2(1 - (1 - 0.9x^2/\omega_2)\sin(\pi(1 - x^1/\omega_1))).$$



(a) $T(\texttt{xc})$ with grid $\texttt{yc} = y(\texttt{xc})$ (b) $T(\texttt{yc})$ with grid $\texttt{xc}$

(c) $T(\texttt{xc})$ with grid $\texttt{yc} = y(\texttt{xc})$ (d) $T(\texttt{yc})$ with grid $\texttt{xc}$

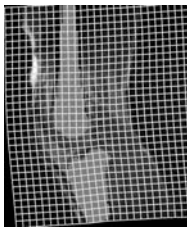
(e) $T(\texttt{xc})$ with grid $\texttt{yc} = y(\texttt{xc})$ (f) $T(\texttt{yc})$ with grid $\texttt{xc}$

Figure 4.2: *Spline-based and other nonlinear transformations of a US image.*

The transformation is visualized in Figure 4.2. The transformation is simultaneously very complex and very regular. For example, this transformation preserves angles.

4.7 Derivatives of Parameterized Transformations

For the optimization schemes to be used later, derivatives of the parametric transformations are required. Note that $y = y(w)$ is considered as a function in the parameters w. For cases where $y = Q(x)w$, this derivative is simply $Q(x)$. However, in an efficient code, the matrix Q should not be assembled every time the



transformation is called. As already shown in the spline example, a persistent variable can be used. The persistent variables are initialized by calling the function without an output request.

Rigid transformations depend nonlinearly on w, and the derivative is thus slightly more complex.

Example 4.5 (Derivative of a 2D Rigid Transformation)
Recall that a 2D rigid transformation is given by $y(w, x) = Q(x)f(w)$ with Q from (4.1) and

$$f(w) = [\cos w_1; -\sin w_1; w_2; \sin w_1; \cos w_1; w_3].$$

Therefore, $d_w y = Q(x)df$ with

$$df = \begin{bmatrix} -\sin w_1 & 0 & 0 \\ -\cos w_1 & 0 & 0 \\ 0 & 1 & 0 \\ \cos w_1 & 0 & 0 \\ -\sin w_1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix};$$

see also `rigid2D`.

Example 4.6 (Derivative of a 3D Rigid Transformation)
A tremendous simplification of the formulae for the derivatives of a 3D rigid transformation is achieved if the rotation matrix is considered as a product of three individual rotation matrices $R = R_3R_2R_1$, where each of the R_i describes a rotation in a 2D plane. Let $w \in \mathbb{R}^6$, $x = (x^1, x^2, x^3)$, $c = \cos$, $s = \sin$,

$$R_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c & -s \\ 0 & s & c \end{bmatrix}, \quad R_2 = \begin{bmatrix} c & 0 & s \\ 0 & 1 & 0 \\ -s & 0 & c \end{bmatrix}, \quad R_3 = \begin{bmatrix} c & -s & 0 \\ s & c & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

With $R(w) = R_3(w_3)R_2(w_2)R_1(w_1)$, it holds that $y = R(w)x + (w_4, w_5, w_6)^\top$, or $y = Q(x)f(w)$, where

$$f(w) = (R_{1,1}, R_{1,2}, R_{1,3}, w_4, R_{2,1}, R_{2,2}, R_{2,3}, w_5, R_{3,1}, R_{3,2}, R_{3,3}, w_6)^\top \quad \text{and}$$

$$Q(x) = \begin{bmatrix} x^1 & x^2 & x^3 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & x^1 & x^2 & x^3 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & x^1 & x^2 & x^3 & 1 \end{bmatrix}.$$

The partial derivatives of f with respect to w_4, w_5, and w_6 are straightforward; differentiating with respect to w_1, w_2, and w_3 yields

$$\partial_{w_1} R(w) = R_3(w_3)R_2(w_2)dR_1(w_1), \quad dR_1 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -s & -c \\ 0 & c & -s \end{bmatrix},$$

$$\partial_{w_2} R(w) = R_3(w_3)dR_2(w_2)R_1(w_1), \quad dR_2 = \begin{bmatrix} -s & 0 & c \\ 0 & 0 & 0 \\ -c & 0 & -s \end{bmatrix},$$

$$\partial_{w_3} R(w) = dR_3(w_3)R_2(w_2)R_1(w_1), \quad dR_3 = \begin{bmatrix} -s & -c & 0 \\ c & -s & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

See also `rigid3D`.

4.8 Summarizing the Parameterized Transformations

This chapter introduces a unified way of coding parameterized transformations `yc=trafo(wc,xc)` and computing their derivatives; see FAIR 5 (this page). Particular transformations discussed in this section are translations and rigid, affine linear, or spline-based transformations. Using the interpolation schemes introduced in Chapter 3, the transformed image can be computed conveniently. These techniques also enable a solution of the so-called forward problem; i.e., given a parameter vector w compute the transformation $y = Qw$ and the transformed image $\mathcal{T}[y]$.

FAIR 5: Parametric Transformations

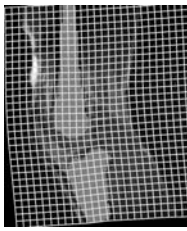
Given points `xc` in a domain Ω and parameters `wc`, this function computes the location of the transformed points `yc`, i.e., `yc=Q*f(wc)`, where `Q` and `f` depend on the specific transformation.

`[y,dy]=trafo(wc,xc)`	
`wc` $\in \mathbb{R}^p$	parameter of the transformation
`xc` $\in \mathbb{R}^{nd}$	grid points
`yc` $\in \mathbb{R}^p$	transformed grid points
`dy` $\in \mathbb{R}^{nd,p}$	derivative of transformation with respect to `wc`

An administrative function `trafo` is supplied. For general options see Section 2.3.7; additional options are as follows:

- `w0=trafo('w0')`: returns parameters of the identity, i.e., `xc==trafo(w0,xc)`
- `[yc,dy]=trafo(wc,xc)`: returns the transformed points and the derivative with respect to `wc`, i.e., `dy=Q`.

Example 4.7 (Configuring the Transformation Model)
Assuming `omega,m,p` are defined, the following code returns the transformed points.



```
trafo('reset','trafo','splineTransformation2D','omega',omega,'m',m,'p',p);
w0 = trafo('w0'); trafo('disp'); yc = trafo(w0,xc);
```

A more efficient implementation which avoids the storage of $I_d \otimes Q(\mathtt{xc})$ is shown below.

Example 4.8 (3D Affine Linear Transformations, Final Version)
This example presents a more efficient implementation which avoids the storage and repetitive computations of $I_d \otimes Q(\mathtt{xc})$; see Table 4.1. For a convenient usage, the function returns its name and location as well as a parameter vector w which results in the identity transformation $y(w, x) = x$.

Table 4.1 A more efficient implementation of an affine linear transformation using a persistent variable Q.

This file is affine3Dsparse.m

```
% function [y,dy] = affine3Dsparse(w,x,varargin)
% (c) Jan Modersitzki 2009/03/24, see FAIR.2 and FAIRcopyright.m.
% computes y = Qfull*w and the derivative wrt. w.
% x = reshape(x,[],3);
% Q = [x(:,1),x(:,2),x(:,3),1], Qfull = kron(I_2,Q) (not used), dy = Q
% if no arguments are given, the parameters for the identity map are returned.

 function [y,dy] = affine3Dsparse(w,x,varargin)

persistent Q
y  = []; dy = [];
if nargin == 0 || ischar(w), % return name and parameters of identity
  y = mfilename('fullfile');
  dy = [1;0;0;0;0;1;0;0;0;0;1;0];
  if nargin>0 && ischar(w), Q = []; end;
  return;
end;
if isempty(w) || (size(Q,1) ~= numel(x)), % rebuild Q
  n = length(x)/3;
  Q = {[reshape(x,[],3),ones(n,1)]};
  if nargout == 0, return; end;
end;
w  = reshape(w,4,3);
% mimicing Qfull*w as [Q*w(:,1),Q*w(:,2),Q*w(:,3)]
y  = [Q{1}*w(:,1);Q{1}*w(:,2);Q{1}*w(:,3)];
dy = Q;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

4.9 FAIR Tutorials on Transformations

FAIR contains the tutorial BigTutorialTrafo which explains how to use the transformation models.

BigTutorialTrafo

E4_US_trafo	plain rotation of 2D US image
E4_US_rotation	rotate 2D US image and grid
E4_US	various transformations for 2D US image

4.10 Exercises

Exercise 4.1
Implement a 2D translation using a persistent Q.

Exercise 4.2
Implement the affine linear transformations `[y,dy]=affine2D(wc,xc)`.

Exercise 4.3
Implement a quadratic transformation `yc=quadratic2D(wc,xc)` *(no derivative required!), where*

$$y^i = w_1^i + w_2^i x^1 + w_3^i x^2 + w_4^i (x^1)^2 + w_5^i (x^2)^2 + w_6^i x^1 x^2, \quad i = 1, 2.$$

Exercise 4.4
Implement the function `yc=rigid2D(wc,xc)`.

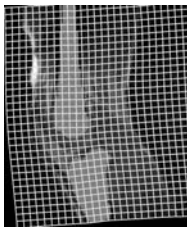
Exercise 4.5
Implement a rotation about the center of the domain, `yc=rotation2D(wc,xc)`.

Exercise 4.6
Improve the implementation of the spline transformation. For $d = 1$, the jth column of $Q^1(X)$ can be computed using

$$Q^1(:, j) = \texttt{splineInter1D}(e^j, \texttt{omega}, X),$$

where $e^j \in \mathbb{R}^p$ with $e_j^j = 1$ and $e_i^j = 0$ for $i \neq j$. Verify that Q^1 is a sparse and structured matrix. Show that for $d = 2$, $Q(X) = Q^2(X^2) \otimes Q^1(X^1)$. Provide a 2D implementation based on a persistent Q.

Exercise 4.7
Extend your implementations to $d = 3$.



Chapter 5

Landmark-Based Registration

In the previous chapter we discussed how to deal with the forward problem; i.e., compute the transformed image $\mathcal{T}[y]$ given a transformation y. Now it is time to measure the quality of the transformation by comparing $\mathcal{T}[y]$ and $\mathcal{R}$. Several measures are known in the literature and can be classified as feature- or intensity-based. In this chapter, the so-called landmark-based registration schemes are discussed. This important class can be viewed as a representative for general feature-based schemes. Intensity-based distance measures are discussed in Chapter 7.

The concept of landmark-based registration is explained for $d = 2$ but can be extended to any spatial dimension. Two X-ray images of human hands serve as the example; see Example 2.7. Outstanding points—the so-called *landmarks*—have been marked manually in the reference image and corresponding points are manually identified in the template image. The idea is to transform the template image such that the distance between the corresponding landmarks becomes as small as possible. Figure 5.1 also illustrates the error (red dashed line) to be minimized.

Various attempts to automatically detect landmarks have been discussed in the literature. However, for many medical applications this is still challenging, and often semiautomatic or even manual detection is performed. Techniques for automatization are not discussed in this chapter; see, e.g., [175] for details.

Let $t_j = [t_j^1, t_j^2]$ denote the position of the jth landmark in the template image and $r_j = [r_j^1, r_j^2]$ the position of the corresponding landmark in the reference image, $j = 1, \ldots, n$, where n is the number of the given landmarks. The goal is to find a transformation $y : \mathbb{R}^2 \to \mathbb{R}^2$, such that

$$y(r_j) = t_j \quad \text{for all} \quad j = 1, \ldots, n.$$

A slightly more general approach is to replace these interpolation conditions by an approximation condition; see also Section 3.6.

The parameterized transformations can be computed analytically: $y(x) = y(\mathtt{wc}, x)$ is known for all $x \in \Omega$. However, a discretization of the domain is used for practical reasons and visualization purposes. This discretization $\mathtt{xc}$ corresponds to the location of the given data, and $T(\mathtt{yc})$ and $R(\mathtt{xc})$ denote the sampled template and the reference image, respectively.

(a) R with LM (b) T with corresponding LM (c) T and difference in LM

Figure 5.1: *Reference and template with corresponding landmarks $r_j = [r_j^1, r_j^2]$ (green $\times$) and $t_j = [t_j^1, t_j^2]$ (red $+$), and landmark errors (red dashes).*

FAIR 6: Landmark-Based Registration

Find a reasonable $y : \mathbb{R}^d \to \mathbb{R}^d$, such that

$$\mathcal{D}[y] = \mathcal{D}^{\mathrm{LM}}[y] = \sum_{j=1}^{n} \|y(r_j) - t_j\|^2 \stackrel{!}{=} \min . \tag{5.1}$$

5.1 Affine Linear Landmark-Based Registration

In this section, it is assumed that the transformation is affine linear; cf. Section 4.2. Therefore, y is given by

$$\begin{bmatrix} y^1 \\ y^2 \end{bmatrix} = \begin{bmatrix} w_1 & w_2 \\ w_4 & w_5 \end{bmatrix} \begin{bmatrix} x^1 \\ x^2 \end{bmatrix} + \begin{bmatrix} w_3 \\ w_6 \end{bmatrix} . \tag{5.2}$$

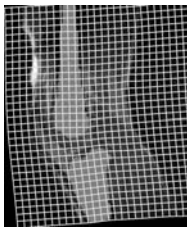
From

$$\mathcal{D}^{\mathrm{LM}}[y] = \sum_{j=1}^{n} \|y(r_j) - t_j\|^2 = \sum_{j=1}^{n} (y^1(r_j) - t_j^1)^2 + \sum_{j=1}^{n} (y^2(r_j) - t_j^2)^2,$$

it follows that the optimization problem (FAIR 6 (this page)) is decoupled. For $i = 1, 2$ let

$$\begin{aligned} w^1 &= [w_1; w_2; w_3], & w^2 &= [w_4; w_5; w_6], & \mathtt{e} &= [1; \ldots; 1] \in \mathbb{R}^n, \\ \mathtt{t} &= [t_j^i] \in \mathbb{R}^{n,d}, & \mathtt{r} &= [r_j^i] \in \mathbb{R}^{n,d}, & Q &= [\mathtt{r}, \mathtt{e}] \in \mathbb{R}^{n,3}. \end{aligned}$$

The problem can then be phrased as $\|Qw^1 - \mathtt{t}(:,1)\|^2 + \|Qw^2 - \mathtt{t}(:,2)\|^2 \stackrel{!}{=} \min$ and the coefficients specifying the minimizer can be computed from these least squares problems: $w^i = (Q^\top Q)^{-1}(Q^\top \mathtt{t}(:,2))$. Implicitly, it is assumed that $Q^\top Q$ has full rank, i.e., $n \geq 3$ and not all reference landmarks are on a straight line; see [162] for theoretical issues.



Example 5.1 (Linear Landmark-Based Registration)
The results for the data displayed in Figure 5.1 are shown in Figure 5.2. The left figure shows the initial hand $T(\texttt{xc})$ with landmarks t_j (red $+$) as well as the mapped landmarks $y(r_j)$ (green $\circ$). The right figure shows the transformed image $T(\texttt{yc})$ with the landmarks r_j (green $\times$) and—assuming y is invertible—the preimage of t_j, $y^{-1}(t_j)$ (red $\circ$).

This file is `E5_linear.m`

```
%%%
setupHandData; xc = reshape(getCenteredGrid(omega,m),[],2);
Q  = [LM(:,3:4),ones(size(LM,1),1)];
wc = (Q'*Q)\(Q'*LM(:,1:2));
yc  = [(wc(1,1)*xc(:,1) + wc(2,1)*xc(:,2) + wc(3,1)),...
       (wc(1,2)*xc(:,1) + wc(2,2)*xc(:,2) + wc(3,2)) ];
LM(:,[5,6]) = ...
     [(wc(1,1)*LM(:,3) + wc(2,1)*LM(:,4) + wc(3,1)),...
      (wc(1,2)*LM(:,3) + wc(2,2)*LM(:,4) + wc(3,2)) ];

P5_LM; % for nice plots
```

5.2 Quadratic Landmark-Based Registration

In the linear case, the match is not perfect. This is to be expected, since the landmark correspondence put nd conditions for only $3d$ parameters. The intuitive idea is to enlarge the transformation space, for example by choosing quadratic transformations. Thus,

$$y^i = w_1^i + w_2^i x^1 + w_3^i x^2 + w_4^i (x^1)^2 + w_5^i (x^2)^2 + w_6^i x^1 x^2, \quad i = 1, \ldots, d.$$

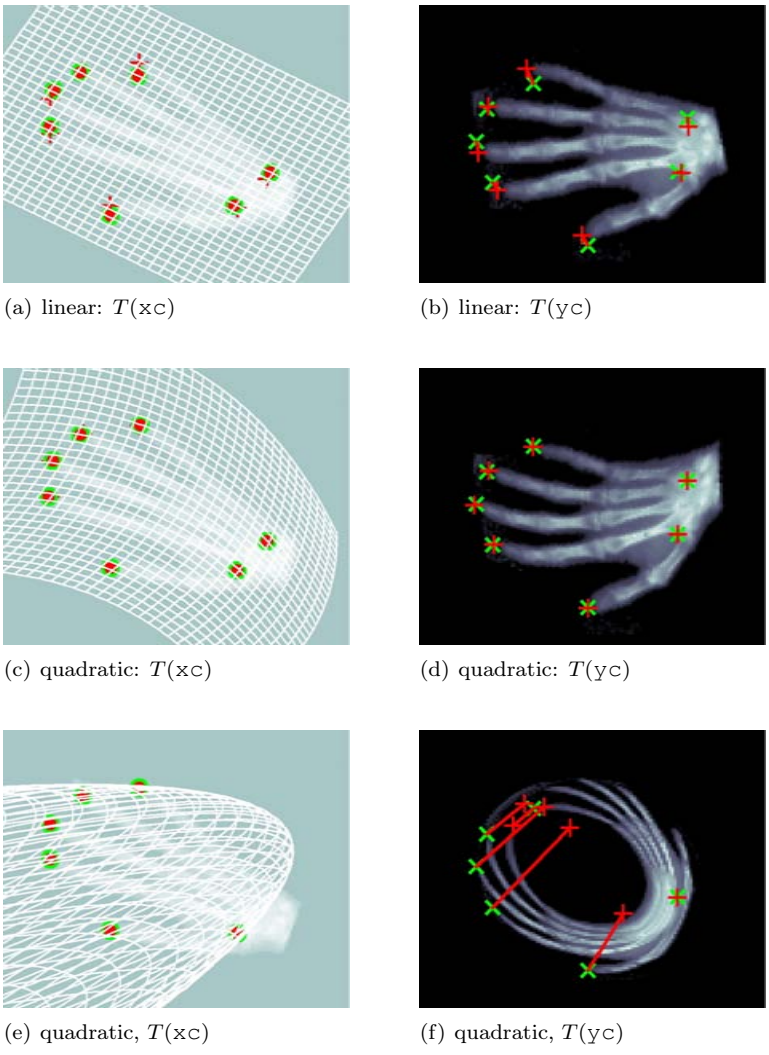
Updating $Q = [1, \texttt{r(:,1)}, \texttt{r(:,2)}, \texttt{r(:,1).^2}, \texttt{r(:,2).^2}, \texttt{r(:,1).*r(:,2)}]$, the coefficients can again be obtained from $w^i = (Q^\top Q)^{-1}(Q^\top \texttt{t(:,i)})$.

Example 5.2 (Quadratic Landmark-Based Registration)
Example 5.1 is continued; results are shown in Figure 5.2. Note that the landmark match is perfect, while the overall appearance of the transformed template is not great as there is unnatural bending of fingers.

This file is `E5_quadratic.m`

```
%%%
setupHandData; xc = reshape(getCenteredGrid(omega,m),[],2); %LM(7,:) = [];
Q = [ones(size(LM,1),1),LM(:,[3:4]),LM(:,3).^2,LM(:,4).^2,LM(:,3).*LM(:,4)];
wc = (Q'*Q)\(Q'*LM(:,1:2));

quad = @(w,x1,x2) (w(1)+w(2)*x1+w(3)*x2+w(4)*x1.^2+w(5)*x2.^2+w(6)*x1.*x2);
yc = [quad(wc(:,1),xc(:,1),xc(:,2));quad(wc(:,2),xc(:,1),xc(:,2))];
LM(:,[5,6]) = [quad(wc(:,1),LM(:,3),LM(:,4)),quad(wc(:,2),LM(:,3),LM(:,4))];

P5_LM; % for nice plots
```

Example 5.3 (Quadratic Landmark-Based Interpolation)
Just removing the last landmark and taking the first 6 landmarks as input yields 12 equations for 12 unknowns, and one ends up with an interpolation problem. The results are shown in Figure 5.2. Although a perfect match of the mapped landmarks



(a) linear: $T(\mathtt{xc})$ (b) linear: $T(\mathtt{yc})$

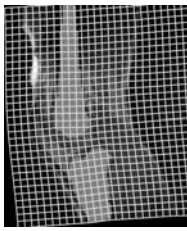
(c) quadratic: $T(\mathtt{xc})$ (d) quadratic: $T(\mathtt{yc})$

(e) quadratic, $T(\mathtt{xc})$ (f) quadratic, $T(\mathtt{yc})$

Figure 5.2: *Template and transformed template with landmarks for linear and quadratic landmark-based registrations and a quadratic landmark interpolation.*

(Figure 5.2(e)) is obtained, the transformation is invertible and thus meaningless for registration purposes.

The previous examples indicate that not every transformation that can be computed is reasonable in the registration context. For many applications the transformation must be one-to-one. All rigid, almost all affine linear, and almost none of the quadratic transformations are one-to-one. The more degrees of freedom one has, the more likely one ends up with an unreasonable transformation. The next section provides some ideas for bypassing this dilemma.



5.3 Thin-Plate-Spline Registration

This is again a section for the more experienced reader. Details of the concepts briefly discussed here can be found in [97, 73, 175, 162]. The underlying ideas have already been discussed in Chapter 3 and can now be applied to the landmark matching as well. A particularly parameterized transformation, such as linear or quadratic, is replaced by the smoothed transformation fulfilling the interpolation condition. The basic ingredient for the smoothness measure is

$$\mathcal{S}^{\mathrm{TPS}}[\eta] = \int_\Omega \langle \nabla^2\eta, \nabla^2\eta \rangle \; dx, \tag{5.3}$$

where the Hessian of a function $\eta : \mathbb{R}^d \to \mathbb{R}$ is denoted by $\nabla^2\eta = [\partial_{i,j}\eta]_{i,j=1}^d$ and the inner product of two matrices A and B is given by $\langle A, B\rangle = \sum_{i,j} a_{i,j}b_{i,j}$. This energy is a linearized version of the bending energy of a thin plate; see, e.g., [97, 73].

5.3.1 Thin-Plate-Spline Interpolation

Using the smoothness criterion (5.3) for any component of the transformation y, the registration problem can be phrased as follows.

FAIR 7: Thin-Plate-Spline Interpolation

Find a reasonable $y : \mathbb{R}^d \to \mathbb{R}^d$, such that

$$\mathcal{S}[y] = \sum_{i=1}^d \mathcal{S}^{\mathrm{TPS}}[y^i] \stackrel{!}{=} \min \quad \text{subject to} \quad \mathcal{D}^{\mathrm{LM}}[y] = \sum_{j=1}^n \|y(r_j) - t_j\|^2 = 0.$$

Note that there is no restriction on y to belong to a certain class of transformations. Fortunately, it has been shown that the components of the solution of FAIR 7 (this page) belong to a certain space that is spanned by shifts of an a priori known radial basis function ρ and a polynomial correction term. To be precise,

$$y^i(x) = \sum_{j=1}^n c_j^i \rho(\|x - r_j\|) + w_0^i + w_1^i x^1 + \cdots + w_d^i x^d, \quad i = 1, \ldots, d, \tag{5.4}$$

where

$$\rho(r) = \begin{cases} r^2 \log r, & d = 2, \\ r, & d = 3. \end{cases}$$

With $\rho_{j,k} = \rho(\|r_k - r_j\|)$, $A = [\rho_{j,k}] \in \mathbb{R}^{n,n}$, and $B = [\mathtt{e}, \mathtt{r}] \in \mathbb{R}^{n,d+1}$, the interpolation condition yields

$$t_j^i = y^i(r_j^1, \ldots, r_j^d) = \sum_{k=1}^n c_k^i \rho_{j,k} + B(j,:)w^i$$

or `t(:,i)` $= Ac^i + Bw^i$. This, together with the necessary condition $B^\top c^i = 0$ for y^i to belong to the spline space, results in a system of linear equations for the coefficients in the expansion (5.4):

$$\begin{bmatrix} A & B \\ B^\top & 0 \end{bmatrix} \begin{bmatrix} c^i \\ w^i \end{bmatrix} = \begin{bmatrix} \texttt{t(:,i)} \\ 0 \end{bmatrix}.$$

The components of the transformation are hence thin-plate splines; which motivates the name thin-plate-spline transformation.

5.3.2 Thin-Plate-Spline Approximation

As for the interpolation schemes discussed in Chapter 3, the interpolation condition $\mathcal{D}^{\mathrm{LM}}[y] = 0$ can be relaxed by replacing interpolation by approximation.

FAIR 8: Thin-Plate-Spline Approximation

Find a reasonable $y : \mathbb{R}^d \to \mathbb{R}^d$ from a thin-plate-spline space, such that

$$\mathcal{D}^{\mathrm{LM}}[y] + \theta \mathcal{S}[y] \stackrel{!}{=} \min,$$

where $\theta \geq 0$ is a parameter balancing mismatch of the landmark and smoothness of the transformation.

It can be shown that the expansion (5.4) holds for the solution of FAIR 8 (this page), where the coefficients are given by

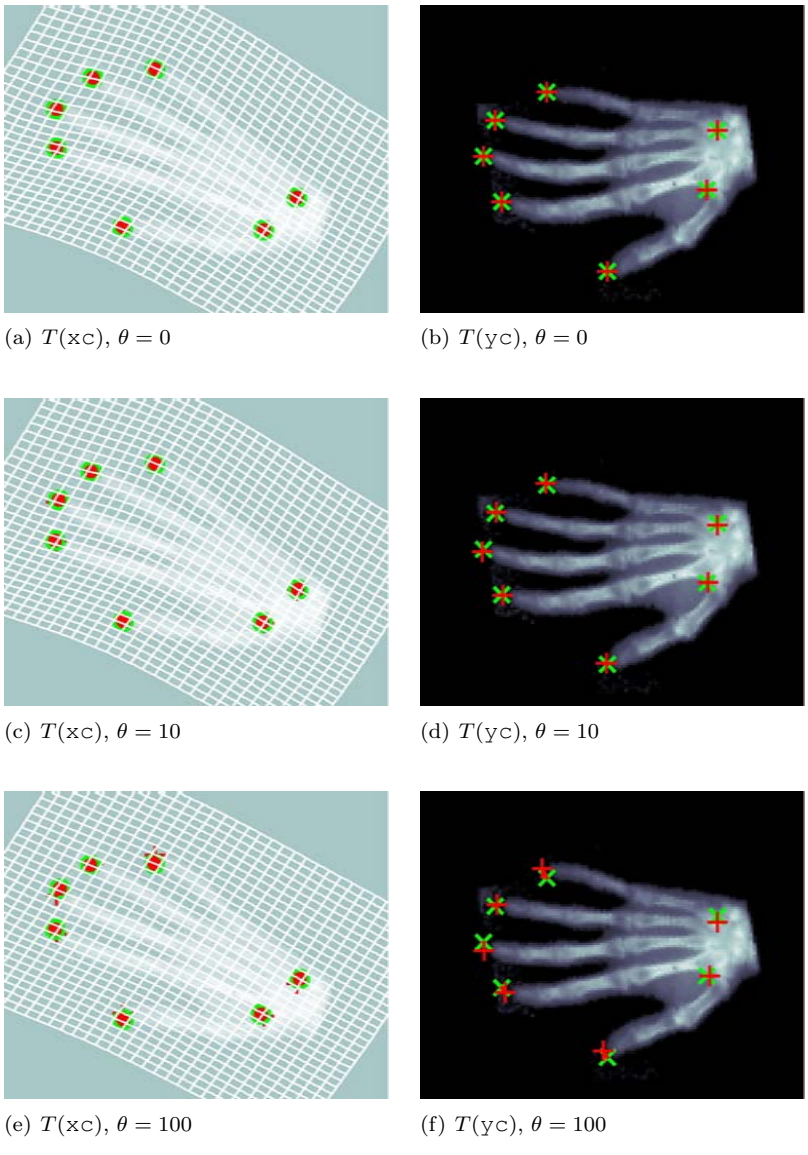
$$\begin{bmatrix} A + \theta I & B \\ B^\top & 0 \end{bmatrix} \begin{bmatrix} c^i \\ w^i \end{bmatrix} = \begin{bmatrix} \texttt{t(:,i)} \\ 0 \end{bmatrix};$$

see, e.g., [162, §4.3.4].

Example 5.4 (Thin-Plate-Spline Registration)
Example 5.1 is continued; results are shown in Figure 5.3. A perfect match is obtained for $\theta = 0$; although the transformation is one-to-one, an unnatural bending of the fingers can be observed. Increasing θ moves the transformation more towards an affine linear map, where the landmark misfit increases but the bending energy decreases. See also `getTPScoefficients` and `evalTPS`.

This file is `E5_TPS.m`

```
%%%
setupHandData; xc = getCenteredGrid(omega,m);
c = getTPScoefficients(LM(:,1:4),'theta',0);
[yc,yLM] = evalTPS(LM,c,xc);
LM(:,[5,6]) = reshape(yLM,[],2);

P5_LM; % for nice plots
```



(a) $T(\mathrm{xc})$, $\theta = 0$

(b) $T(\mathrm{yc})$, $\theta = 0$

(c) $T(\mathrm{xc})$, $\theta = 10$

(d) $T(\mathrm{yc})$, $\theta = 10$

(e) $T(\mathrm{xc})$, $\theta = 100$

(f) $T(\mathrm{yc})$, $\theta = 100$

Figure 5.3: *Thin-plate-spline registration with various θ's.*

5.4 Summarizing Landmark-Based Registration

Landmark-based registration techniques provide fast and efficient tools for image registration. Typically, the components of the transformation can be expanded in terms of some basis functions,

$$y^i(x) = Q^i(x)w^i = \sum_{k=1}^{n} w_k^i q_k(x),$$

where the coefficients are deduced from the interpolation condition

$$t_j^i = y^i(r_j) = \sum_{k=1}^{n} w_k^i q_k(r_j), \quad j = 1, \ldots, n, \qquad \text{or} \quad \texttt{t(:,i)} = Qw^i.$$

For the thin-plate-spline approach, additional conditions arise. The interpolation condition can easily be replaced by an approximation condition. The transformation to be computed is parametric and can be computed using techniques similar to the ones presented in Chapter 4.

Drawbacks are that the transformation is completely determined by the choice of landmarks and that the dense intensity information is used only for landmark detection. If the transformation is not pleasing, an improvement can be achieved only by adding more landmarks—or removing some of the landmarks. However, it is not clear how to detect additional corresponding landmarks nor that the result is more pleasing. In addition, the transformation is not physical. Even for the thin-plate-spline approach, the transformation is a combination of more or less smooth components. There is no interaction between these components and no guarantee that the overall transformation is one-to-one.

A remedy is to use geodesic splines which are designed to guarantee a one-to-one transformation, but this topic goes beyond the scope of this book. The interested reader is referred to [196, 197].

5.5 FAIR Tutorials on Landmark-Based Registration

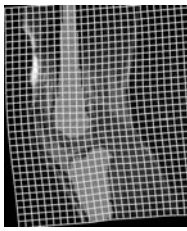
FAIR contains the tutorial `BigTutorialLandmarks` which explains how to use the approaches discussed in this chapter.

```
BigTutorialLandmarks
  E5_2D_affine       use affine transformation
  E5_2D_quadratic    use quadratic transformation
  E5_2D_TPS          use thin-plate-spline transformation
```

5.6 Exercises

Exercise 5.1
Provide a registration scheme based on landmarks and rigid transformations.

Exercise 5.2
Using `ginput`*, write a tool for setting landmarks.*

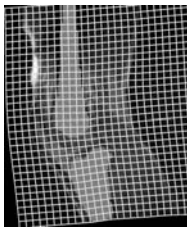


Exercise 5.3
Experiment with other landmarks. Try to pick $2, 10,$ *and* 100 *landmarks.*

Exercise 5.4
Think about a 3D landmark setting tool for hundreds of landmarks.

Exercise 5.5
Experiment with the thin-plate-spline approach and play with θ*.*

Exercise 5.6
Try to choose landmarks such that the transformation cannot be one-to-one.



Chapter 6
Parametric Image Registration

In the previous chapter we discussed how to derive registrations using a distance measure based on image features, or, more precisely, on landmarks. Supported by a strong theory, the transformation is basically a linear combination of a priori chosen or implicitly defined basis functions. The coefficients of an optimal expansion can be computed by solving least squares problems.

In this chapter, parametric image registration (PIR) for intensity-based distance measures is discussed. The L_2-norm of the intensity difference serves as a distance measure; see Section 6.2. However, the concepts are very general and will be used for the various distance measures discussed in Chapter 7. All distance measures to be considered can also be phrased as

$$\mathcal{D}[\mathcal{T}, \mathcal{R}] = \int_\Omega \phi(\mathcal{T}, \mathcal{R})\, dx. \tag{6.1}$$

The first complication arises from the fact that the integral cannot be computed analytically but only numerically. The midpoint quadrature rule is used here; see Section 6.1. This method provides approximations as accurate as needed. Exploiting numerical integration, a discretized analogue of the L_2-norm is derived.

Finally, a joint objective function J is composed of a parametric transformation $y(w) = y(w, x)$, the transformed template $\mathcal{T}[y]$, and the distance measure $\mathcal{D}[\mathcal{T}[y], \mathcal{R}]$,

$$J(w) = \mathcal{D}[\mathcal{T}(y(w, x), \mathcal{R})] + S(w).$$

The explanation of the additional regularization S is given in Section 6.5. To begin with $S(w) = 0$ in Sections 6.1–6.4.

The remainder of this chapter is devoted to a Gauss–Newton-type optimization technique, computing a minimizer of the distance measure numerically; see Section 6.3. For a more general approach to numerical optimization, see Section 1.4 and in particular [165]. Various examples of PIR are presented in Section 6.4. In Section 6.5, this approach is extended to high-dimensional transformation spaces, where an additional regularization is required. Section 6.5 may be skipped in a first reading.

In Section 6.6, emphasis is given to a multilevel strategy. Running from coarse to fine discretization, a sequence of problems is solved, where the solution of a coarser problem serves as a starting guess for a finer problem. There are basically two big advantages of the multilevel strategy. The first advantage is a numerical one. The bulk of computations is performed on the cheaper coarse levels, while on the finer, more expensive, levels, typically only a few correction steps are needed. Based on the outstanding starting point and fast optimization schemes, the correction is expected to be very efficient. The second advantage is based on the smoothing strategy, which can also be seen as a kind of convexification. On a coarse level, the objective function is replaced by a smoothed version. Therefore, the optimization is easier and, most importantly, the chance of ending at a local minimum is reduced. However, there is no guarantee of avoiding local minima; see also Exercise 6.5.

As always, the concepts are valid for any dimension. However, the description, and in particular the visualization, is for $d = 2$. Two slices from a histological serial sectioning serve as test data; see Example 2.8, Figure 2.4, and `setupHNSPdata`.

6.1 Numerical Integration—Discretizing Integrals

There exist many excellent approaches to numerical integration; see the literature provided in Section 1.4.3. A discussion of those is far beyond the scope of this book. Instead, insight is given into the simple and robust midpoint quadrature which performs sufficiently for image registration purposes. To begin with, the ideas are explained for an arbitrary integrable function $\psi : [0, \omega] \to \mathbb{R}$.

Suppose a function $\psi : \Omega \to \mathbb{R}$ is to be integrated; see Figure 6.1. Dividing the interval $\Omega = (\omega^1, \omega^2)$ into m cells of width h and cell centers x_j, the idea is to evaluate the function at the cell centers, multiply these function values by the cell width, and sum, i.e.,

$$\int_\Omega \psi(x)\, dx = h \sum_{j=1}^{m} \psi(x_j) + \mathcal{O}(h^2). \tag{6.2}$$

Note that the meaning of m, and thus h, is not to be confused with the data size and data grid discussed in Chapter 3. There is no relation between the

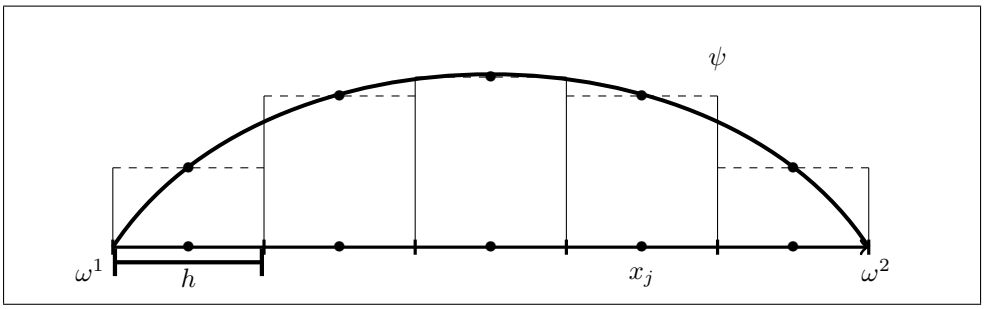
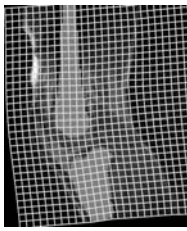


Figure 6.1: *Midpoint quadrature rule in one dimension.*



quadrature rule for the continuous function ψ and the data size. For the quadrature rule, m can be chosen arbitrarily. Since the number of cells m is an integer, it is more convenient to present this number rather than presenting the cell width h, which is the numerically important quantity. The error in the above formula is of order h^2. This implies that any error tolerance can be satisfied, provided h is chosen small enough.

Example 6.1 (Numerical Quadrature of a 1D "Mother" Spline)
The first example is to numerically compute the integral of the function $\psi(x) = b^3(x) = b(x-3)$ on $\Omega = (0,6)$ from (3.8), where $I := \int_\Omega \psi(x)\,dx = 6$. Figure 6.2 shows $Q(h) = h\sum_{j=k}^{n} \psi(x_k^h)$ and the error $|I - Q(h)|$ versus h, where $h(j) = 6/m(j)$, $m(j) = 2^j$, $j = 1:10$, and x^h denotes a cell-centered grid of width h. Observe the quadratic behavior of the error.

This file is E6_quadrature_Spline1D.m

```
%%%
omega = [0,6];   I = 1;  psi = @(x) spline1D(3,x); h = []; Q = [];
for j=1:10,
  m    = 2^j;
  h(j) = diff(omega)/m;
  xc   = getCenteredGrid(omega,m);
  Q(j) = h(j)*sum(psi(xc));
end;
figure(1); clf; p1=semilogx(h/h(1),Q+eps,'kx',h/h(1),Q,'k-');
figure(2); clf; p2=loglog(h/h(1),abs(I-Q)+eps,'kx',h/h(1),abs(I-Q),'k-');
```

The extension to higher dimensions is straightforward. In fact formula (6.2) applies if h is replaced by $\mathtt{hd} = h^1 \cdots h^d$ and $\mathcal{O}(h)$ by $\mathcal{O}(\|\mathtt{hd}\|)$.

Example 6.2 (Numerical Quadrature of a 2D Spline)
The previous example is continued, where now $d = 2$, $\Omega = (0,6)\times(0,8)$, and $\psi(x) = b^3(x^1)b^4(x^2)$. Note that $I = \int_\Omega \psi\,dx = 36$. Figure 6.2 shows $Q(h) = h^1h^2\sum_{k=1}^{n}\psi(x_k^c)$ and the error $|I - Q(h)|$ versus $h = h^1 = h^2$. Observe again the quadratic behavior of the error.

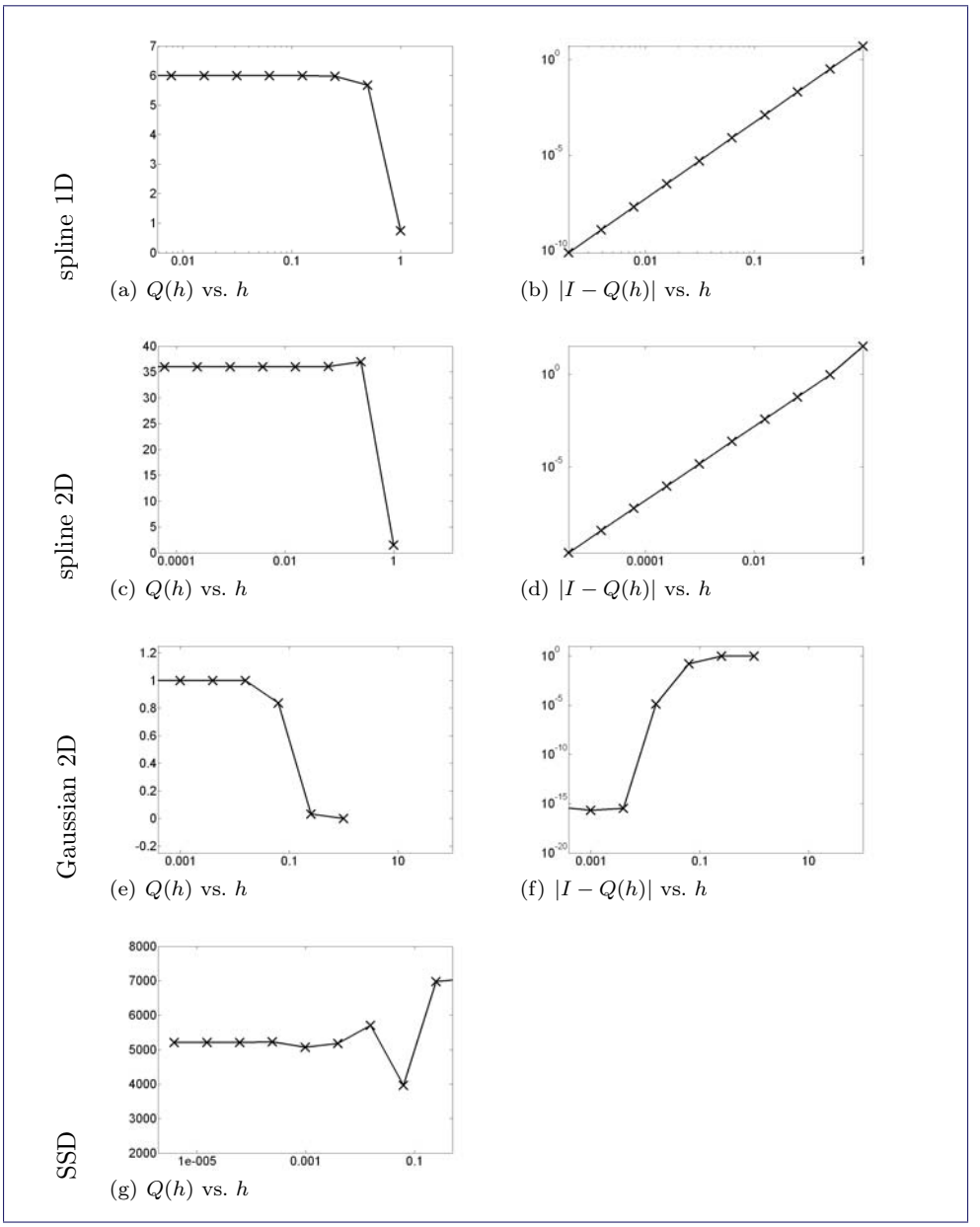
This file is E6_quadrature_Spline2D.m

```
%%%
omega = [0,6,0,8]; I = 36; T = zeros(6,8); T(3,4) = 1; h = []; Q = [];
psi = @(xc) splineInter2D(T,omega,xc);
for j=1:10,
  m    = 2^j*[1,1];
  h(j) = prod((omega(2:2:end)-omega(1:2:end))./m);
  xc   = getCenteredGrid(omega,m);
  Q(j) = h(j)*sum(psi(xc));
end;
figure(1); clf; p1=semilogx(h/h(1),Q+eps,'kx',h/h(1),Q,'k-');
figure(2); clf; p2=loglog(h/h(1),abs(I-Q)+eps,'kx',h/h(1),abs(I-Q),'k-');
```

Example 6.3 (Numerical Quadrature of a 2D Gaussian)
In this example $d = 2$, $\Omega = (0,20)\times(0,20)$, and ψ is a translated Gaussian, i.e.,

$$\psi(x) = 0.5/\pi \cdot \exp\left(-0.5\|x-c\|^2\right), \quad c = [10,10].$$

Note that if $\Omega = \mathbb{R}^2$, the integral would be one, i.e., $I = 1$. However, the Gaussian is not compactly supported, and integration over Ω is more involved than in the



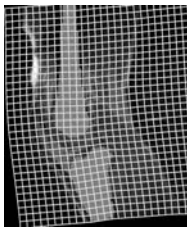
(a) $Q(h)$ vs. h (b) $|I - Q(h)|$ vs. h

(c) $Q(h)$ vs. h (d) $|I - Q(h)|$ vs. h

(e) $Q(h)$ vs. h (f) $|I - Q(h)|$ vs. h

(g) $Q(h)$ vs. h

Figure 6.2: *Quadrature on duty for various functions ψ.*



previous examples. Figure 6.2 shows $Q(h) = h^1 h^2 \sum_{k=1}^{n} \psi(x_k^h)$ and the "error" $|I - Q(h)|$ versus $h = h^1 = h^2$. Note that this "error" should go to a constant, as 1 is not the value of the integral. This expectation is confirmed by this experiment.

This file is E6_quadrature_Gaussian2D.m

```
%%%
omega = [0,20,0,20]; h = zeros(10,1); Q = zeros(size(h));
I   = 1; % Note that I=1 is just an approximation to the integral!
psi = @(x1,x2) (1/(2*pi))*exp(-0.5*((x1-10).^2+(x2-10).^2));
for j=1:length(h),
  m    = 2^j*[1,1];
  h(j) = prod((omega(2:2:end)-omega(1:2:end))./m);
  xc   = reshape(getCenteredGrid(omega,m),[m,2]);
  Q(j) = h(j)*sum(sum(psi(xc(:,:,1),xc(:,:,2))));
end;
figure(1); clf; p1=semilogx(h/h(1),Q+eps,'kx',h/h(1),Q,'k-');
figure(2); clf; p2=loglog(h/h(1),abs(I-Q)+eps,'kx',h/h(1),abs(I-Q)+eps,'k-');
```

6.2 Sum of Squared Differences

In this section, the L_2-*norm* of the difference image or *sum of squared differences (SSD)* is introduced as a prototype of a distance measure. The focus here is on making this function accessible so that the optimization scheme to be discussed in the next section. A discussion from an imaging point of view is presented in Chapter 7; see also [162].

The first subsection presents the continuous formulation, while the second addresses its discretization based on the midpoint quadrature rule. The main point here is that instead of one particular discretization, a sequence of discretizations ranging from coarse to fine is obtained, which approximate the continuous SSD up to any desired accuracy. It is explained how this implementation can be combined with the parametric transformations as introduced in Chapter 4.

6.2.1 Continuous SSD

The distance to be discussed basically measures the energy contained in the difference image $\mathcal{T}[y] - \mathcal{R}$. For this to be meaningful, it has to be assumed that the intensities of the two images are comparable; i.e., the gray value of a particle is more or less the same in the reference and template images. The SSD measure is defined as follows.

FAIR 9: Sum of Squared Differences (SSD) Distance Measure

Given $\mathcal{T}$ and $\mathcal{R}$, the SSD measure is

$$\mathcal{D}^{\mathrm{SSD}}[\mathcal{T}, \mathcal{R}] = \frac{1}{2} \int_{\Omega} (\mathcal{T}(x) - \mathcal{R}(x))^2 \, dx.$$

Although a continuous setting is used, the integral cannot be computed analytically. Therefore, numerical integration or quadrature is required.

6.2.2 Discretized SSD

A discrete analogue of the SSD is given by a numerical integration of the function $\psi(x) = \frac{1}{2}(\mathcal{T}(x) - \mathcal{R}(x))^2$, where $\mathcal{T}$ and $\mathcal{R}$ are the interpolants of the template and reference, respectively. For a particular h, let $\mathtt{xc}$ denote the corresponding cell-centered grid of width h, $T^h = T(\mathtt{xc})$, and $R^h = R(\mathtt{xc})$, respectively. A discretized version of the SSD is defined in FAIR 10 (this page).

FAIR 10: Discretized SSD

The discretized SSD is based on a midpoint quadrature rule with an a priori chosen cell-centered grid of width h and reads

$$D^{\mathrm{SSD},h}(T^h, R^h) = \frac{1}{2} \cdot \mathtt{hd} \cdot \|T^h - R^h\|^2, \quad \text{where} \quad \mathtt{hd} = h_1 \cdots h_d.$$

Example 6.4 (Computing the SSD)
The examples of the previous section are continued. For the SSD, the integral I, and hence the error, is unknown. However, the midpoint quadrature rule guarantees a quadratic convergence to the integral. Figure 6.2 confirms this expectation.

This file is E6_quadrature_SSD2D.m

```
%%%
setupHNSPData; clf; h = []; Q = [];
inter('reset','inter','splineInter2D');
[T,R] = inter('coefficients',dataT,dataR,omega,'out',0);
for j=1:10,
  m    = 2^j*[1,1];
  h(j) = prod((omega(2:2:end)-omega(1:2:end))./m);
  xc   = getCenteredGrid(omega,m);
  res  = inter(T,omega,xc) - inter(R,omega,xc);
  psi  = 0.5*h(j)*res'*res;
  Q(j) = psi;
end;
figure(1); clf; p1=semilogx(h/h(1),Q+eps,'kx',h/h(1),Q,'k-');
```

The bottom line of the above example is that there is absolutely no reason to worry about the lack of an analytic expression for the integral. Numerical integration enables the computation of the integral up to any desired precision. However, asking for high accuracy (wanted) results in small h and requires a longer computing time (not wanted). This dilemma will be addressed in Section 6.6.

In the above code, the distance measure is coded in a two-layer fashion, i.e., using a residual r and an outer function ψ. Although this is certainly overkill for the SSD, the concept serves as a template for the distance measures to be discussed in Chapter 7.

6.2.3 SSD and Parametric Transformations

For ease of presentation, dependencies on h are neglected in this section. For example, the objective function reads $J(\mathtt{wc}) = D^{\mathrm{SSD}}(T(y(\mathtt{wc}, \mathtt{xc}), R)$. In this section, the combination of parametric transformations and the SSD distance is explained. In particular, rotations about the center of the domain and translations are considered.

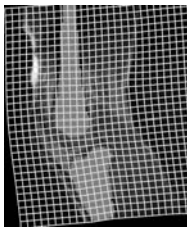


Table 6.1 Implementations of the rotation and translation examples.

This file is E6_HNSP_SSD_rotation2D_level4.m

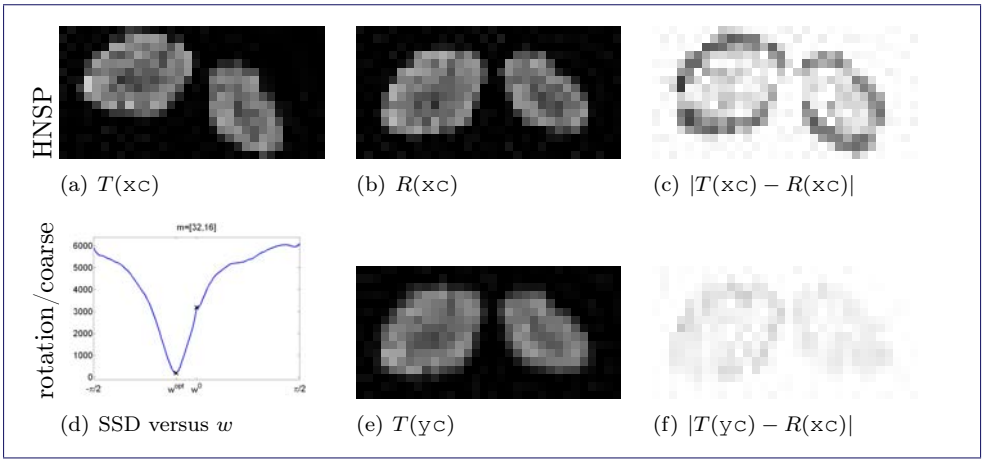
```
%%%
setupHNSPData;
inter('set','inter','linearInter2D');
level = 4; omega = MLdata{level}.omega; m = MLdata{level}.m;
[T,R] = inter('coefficients',MLdata{level}.T,MLdata{level}.R,omega);
xc = getCenteredGrid(omega,m);
Rc = inter(R,omega,xc);

center = (omega(2:2:end)-omega(1:2:end))'/2;
trafo('set','trafo','rotation2D','c',center);

wc = pi/2*linspace(-1,1,101);  dc = zeros(size(wc));
figure(1); clf;
for j=1:length(wc),
  yc = trafo(wc(j),xc);
  Tc = inter(T,omega,yc);
  dc(j) = SSD(Tc,Rc,omega,m);
  viewImage(255-abs(Tc-Rc),omega,m); drawnow; pause(1/60)
end;
figure(2); clf; p1 = plot(wc,dc);
```

This file is E6_HNSP_SSD_translation2D_level4.m

```
%%%
setupHNSPData;
level = 4; m = MLdata{level}.m;
inter('set','inter','linearInter2D');
[T,R] = inter('coefficients',MLdata{level}.T,MLdata{level}.R,omega);
xc = getCenteredGrid(omega,m); Rc = inter(R,omega,xc);

trafo('set','trafo','translation2D');
figure(1); clf;
[w1,w2] = ndgrid(0.2*linspace(-1,1,21),0.2*linspace(-1,1,21));
dc  = zeros(size(w1));
for j=1:numel(dc),
  yc = trafo([w1(j);w2(j)],xc);
  Tc = inter(T,omega,yc);
  dc(j) = SSD(Tc,Rc,omega,m);
  viewImage(Tc,omega,m); pause(1/100)
end;
figure(1); clf; surf(w1,w2,dc); hold on; grid off; contour(w1,w2,dc)
title(sprintf('translation, m=[%d,%d]',m)); view(-135,33);
```

Example 6.5 (SSD and Rotations)
The rotation is implemented in `rotation2D` and Table 6.1. The SSD is considered as a function in w; see Figure 6.3. The SSD function indicates where a reasonable match occurs; the optimal `wc` is a minimizer of the SSD. Comparing the results for a coarse discretization (level 4, $m = [32, 16]$, Figure 6.3) with the ones for a fine discretization (level 8, $m = [512, 256]$, Figure 6.4), we see that only minor differences. This motivates a strategy where a minimizer which is efficiently computed on a coarse grid serves as a starting point for the finer grid. This multilevel strategy is discussed in Section 6.6.

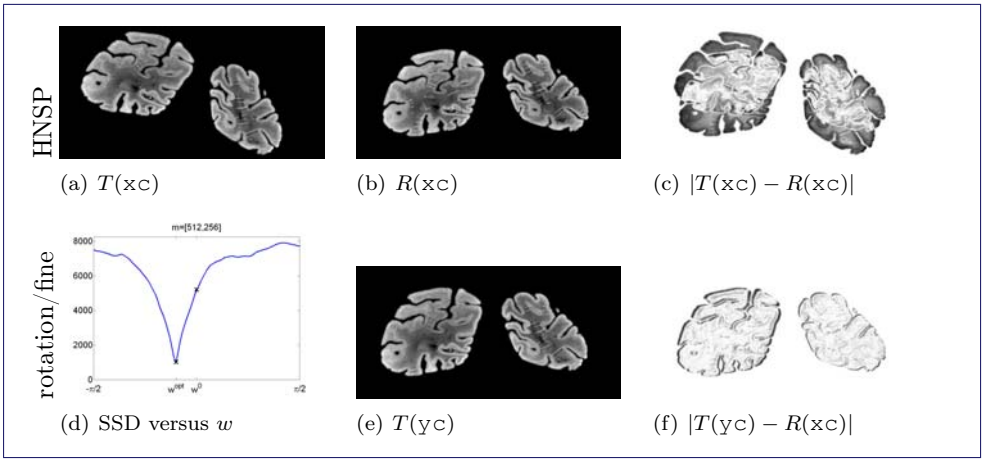
Example 6.6 (SSD and Translations)
The translation is implemented in `translation2D`; see Table 6.1. Figure 6.5 shows the SSD as a function in $w \in \mathbb{R}^2$, where two different representations of the im-



(a) $T(\mathtt{xc})$ (b) $R(\mathtt{xc})$ (c) $|T(\mathtt{xc}) - R(\mathtt{xc})|$

(d) SSD versus w (e) $T(\mathtt{yc})$ (f) $|T(\mathtt{yc}) - R(\mathtt{xc})|$

Figure 6.3: *SSD versus rotations for* $m = [32, 16]$; $\mathtt{yc} = y(\mathtt{wc}, \mathtt{xc})$.



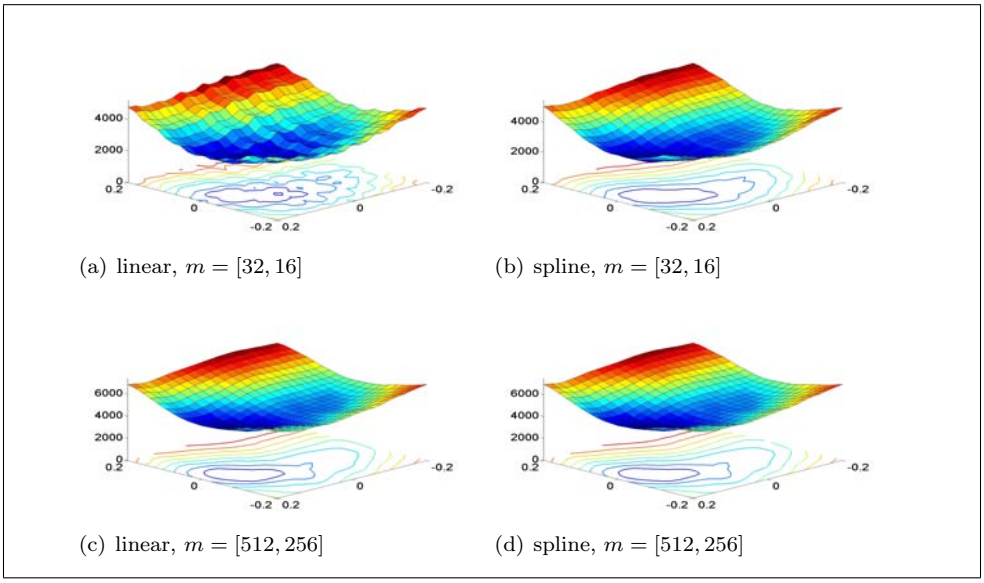
(a) $T(\mathtt{xc})$ (b) $R(\mathtt{xc})$ (c) $|T(\mathtt{xc}) - R(\mathtt{xc})|$

(d) SSD versus w (e) $T(\mathtt{yc})$ (f) $|T(\mathtt{yc}) - R(\mathtt{xc})|$

Figure 6.4: *SSD versus rotations for* $m = [512, 256]$; $\mathtt{yc} = y(\mathtt{wc}, \mathtt{xc})$.

ages ($m = [32, 16]$ and $m = [512, 256]$) are shown and two different interpolation schemes are used: linear interpolation (cf. Section 3.3) and spline interpolation (cf. Section 3.4).

At first glance, the objective functions look pretty much the same and this supports the strategy to compute a minimizer on a coarse level with low computational costs. However, a qualitative difference can be observed with respect to the smoothness of the SSD. The linear interpolation scheme results in a nonsmooth objective function. Thus, optimization schemes are expected to perform poorly using this approach. In contrast, the spline interpolation scheme yields an objective function that is sufficiently smooth for fast optimization schemes.



(a) linear, $m = [32, 16]$ (b) spline, $m = [32, 16]$

(c) linear, $m = [512, 256]$ (d) spline, $m = [512, 256]$

Figure 6.5: *SSD versus translations for $m = [32, 16]$ and $m = [512, 256]$ using linear and spline interpolations, respectively.*

6.3 Numerical Optimization of Parametric Image Registration

The goal of this section is to provide a numerical optimization scheme enabling an automatic identification of optimal parameters. The intention is to provide insight into a basic optimization tool rather than discussing more sophisticated techniques. For an overview on numerical optimization techniques, see the literature in Sections 1.4.4 and 9.6. In this section, the Gauss–Newton scheme is discussed, and the algorithm is summarized in `GaussNewtonArmijo`.

The implementation needs to be flexible enough to handle different spatial dimensions, various parametric transformations, and various distance measures. Analogously to the transformation function `trafo` in FAIR 5 (p.54) a general purpose distance function `distance` is used. At present, this would be the SSD distance of FAIR 10 (p.72); other options are discussed in the next chapter. Finally, a meaningful stopping rule has to be supplied which is compatible with a multilevel strategy. For ease of presentation, visualization is completely neglected. However, from an application point of view, this is probably one of the more important issues.

6.3.1 PIR Objective Function

Since the outline is already complex, an objective function J is provided. Based on a certain discretization `xc` and the current parameters `wc`, this function computes the transformation $\texttt{yc} = y(\texttt{wc}, \texttt{xc})$, the transformed image $T(\texttt{yc})$, and the distance $D(T(\texttt{yc}), R)$,

$$J(\texttt{wc}) = D(T(y(\texttt{wc}, \texttt{xc})), R(\texttt{xc})) + S(\texttt{wc}). \tag{6.3}$$

The objective function also enables an additional regularization S which is explained and explored in Section 6.5 but disabled here by setting $S(w) = 0$. The objective function is outlined as follows.

objective function

```
function [Jc,dJ,H] = PIRobjFctn(T,Rc,omega,m,xc,beta,wc)
[yc,dy] = trafo(wc,xc);                          % compute transformation
[Tc,dT] = inter(T,omega,yc);                     % compute transformed image
[Jc,rc,dD,dr,d2psi] = distance(Tc,Rc,omega,m);   % compute distance
dJ = dD*dT*dy; dr = dr*dT*dy;                    % multiply outer and inner derivatives
H  = dr'*d2psi*dr + beta*speye(length(wc));      % compute approximation to Hessian
```

A note on the approximation of the Hessian of J is in place. It is assumed that the objective function J can be written as $J(\mathtt{wc}) = \psi(r(\mathtt{wc}))$. For example, $r = T(y(w, X)) - R(X)$ and $\psi = \frac{1}{2}\, r^\top r$ for the SSD. The central idea is to replace J by a quadratic $\hat{J}$ obtained from a Taylor expansion,

$$J(\mathtt{wc} + \mathtt{dw}) \approx \hat{J}(\mathtt{wc} + \mathtt{dw}) = J + dJ\ \mathtt{dw} + \frac{1}{2}\ \mathtt{dw}^\top H\ \mathtt{dw},$$

where the approximation of the Hessian is $H = dr^\top d^2\psi\ dr$ and thus positive semidefinite for a convex ψ. Hence, $\hat{J}$ is convex, and a minimizer is characterized by the following linear system:

$$H\ \mathtt{dw} = -dJ. \tag{6.4}$$

The key point of the Gauss–Newton idea is to use the second derivative of the outer function $d^2\psi$, but to neglect higher-order derivatives of the inner functions. This is a perfect option for image registration, since the outer function is smooth while the residual depends on the noisy data. Particularly for the SSD, (6.4) results in

$$dr^\top\ dr\ \mathtt{dw} = dr^\top r \iff dr^\top (dr\ \mathtt{dw} - r) = 0,$$

which is also known as the normal equations for the linearized least squares problem $\|r(\mathtt{wc} + \mathtt{dw})\| \xrightarrow{\mathtt{dw}} \min$; see, e.g., [112].

In order to avoid a singular approximation of the Hessian, it might be worthwhile to add a small part of the identity: $H \leftarrow H + \beta I$. However, in most cases H is symmetric positive definite and this additional regularization might be skipped.

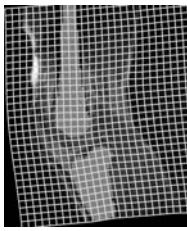
6.3.2 Practical Issues in Coding the PIR Objective Function

Although the above implementation does work in principle, a more sophisticated one is used; see `PIRobjFctn` and FAIR 11 (p.77). This function has an addition output `para` which basically stores intermediates used for visualization; see Section 6.3.4. It also has two additional input arguments `M` and `wRef` which are used for the optional Tychonoff regularization in (6.3),

$$S(\mathtt{wc}) = \frac{1}{2}\ (\mathtt{wc} - \mathtt{wRef})^\top \mathtt{M}\ (\mathtt{wc} - \mathtt{wRef}).$$

Setting `M=0` or `M=[]` disables this option.

The implementation of this function has two parts. The first part assumes that `wc` is not an input argument, and it is only used to force a status report on



the current setting, i.e., to show information about `trafo`, `inter`, `distance`, and so forth. The second part performs the actual computation. In contrast to the above code, a flag `doDerivative` based on the number of input arguments is used to avoid unnecessary computation of derivatives.

The implementation also enables a sparse coding of `dy` $= I_d \otimes Q$. Note that it is more efficient to store just the matrix Q rather than the potentially big Kronecker product. In case of a dimension mismatch of `dD` and `dY`, the implementation thus assumes that `dY={Q}` and uses an appropriate matrix-vector multiplication. In fact, in most cases $Q = Q_d \otimes \cdots \otimes Q_1$ and setting `dY={Q_1,...,Q_d}` provides an even bigger option for saving memory.

FAIR 11: Objective Function for Parametric Image Registration (PIR)

The discretized objective function for PIR is

$$J^h(\mathtt{wc}) = D^h(T(\mathtt{yc}), R(\mathtt{xc})) + S(\mathtt{wc}),$$

where `yc` $= y(\mathtt{wc}, \mathtt{xc})$, D^h is a distance measure, and S is a regularizer of the coefficients. The specific transformation, interpolation, and distance measure are supplied by `trafo`, `inter`, and `distance`. Note that `H` could be a matrix or a function handle if a matrix-free code is used; see Section 8.5 for details.

`[Jc,para,dJ,H]= PIRobjFctn(T,Rc,omega,m,beta,M,wRef,xc,wc)`	
`T,Rc`	template and sampled reference `Rc` $= R($`xc`$)$
`omega, m`	specifying discretization of Ω
`beta` ≥ 0,	parameter for regularizing Hessian
`M, wRef`	optional regularization (default: `M=[]; wRef=[]`)
`xc` $\in \mathbb{R}^{dn}$	underlying grid points
`wc` $\in \mathbb{R}^{p}$	current parameters
`Jc` $\in \mathbb{R}$	current objective function value based on `wc`
`para` (structure)	collects intermediates for visualization `para={Tc,Rc,omega,m,yc,Jc}`, where `yc` $= y($`wc`$,$ `xc`$)$ and `Tc` $= T($`yc`$)$
`dJ` $\in \mathbb{R}^{1,p}$	derivative of J w.r.t. `wc`
`H` $\in \mathbb{R}^{p,p}$	approximation to Hessian, `H` $\approx d^2D + d^2S + \beta I$

6.3.3 Gauss–Newton Scheme

A plain vanilla Gauss–Newton scheme is implemented; see [165] for an extended description. The code is generic and used for other optimization tasks as well. Thus internally, the variable is called `yc`. In the context of parametric registration, `wc` would have been a better name. However, this is a generic optimization code and will be used in a different context later. Thus, a typical call of the Gauss–Newton scheme reads as follows.

```
fctn  = @(wc) PIRobjFctn(T,Rc,omega,m,beta,M,wRef,xc,wc); % handle to objective function
w0    = trafo('w0');                                      % initial guess
[wc,His] = GaussNewtonArmijo(fctn,w0);                    % call the optimizer
```

Starting with an initial guess wc, the idea is to improve this guess by an update dw which minimizes the quadratic model and is thus characterized by $H\texttt{dw} = -dJ$. Note that since the approximation H to the Hessian is symmetric positive definite, dw is a descent direction: $\texttt{dJ}^\top\texttt{dw} < 0$. This procedure is iterated until the stopping criteria are satisfied.

A description of the stopping criteria, the line search, and a sketch of the implementation is given below; details are given in GaussNewtonArmijo and FAIR 12 (p.79).

Gauss-Newton Scheme

```
function [wc,His] = PIRGaussNewtonArmijo(T,R,omega,m,yc)
% -- start initial phase -------------------------------------------------
[Jc,dJ,H] = fctn(yc);                              % compute current values
% -- start iteration phase -----------------------------------------------
while 1,
  iter    = iter + 1;                              % update iteration count
  checkStoppingRules;                              % check the stopping rules
  dy   = -H\dJ;                                    % solve Quasi-Newton's system
  [t,yt,LSiter] = Armijo(fctn,yc,dy,Jc,dJ);        % perform Armijo line-search
  if LSiter<0, break; end;                         % break if line-search fails
  yc = yt;   [Jc,dJ,H] =fctn(yc);                  % update current values
end;%while
```

Stopping

The following common stopping criteria for optimization are used for this iteration; see, e.g., [109].

stopping criteria

```
STOP(1)  = abs(Jold-Jc)   <= tolJ*(1+abs(JRef));
STOP(2)  = norm(yc-yold)  <= tolW*(1+norm(yc));
STOP(3)  = hd*norm(dJ)    <= tolG*(1+abs(JRef));
STOP(4)  = norm(dJ)       <= eps;
STOP(5)  = (iter > maxIter);
STOP     = all(STOP(1:3)) | any(STOP(4:5));
```

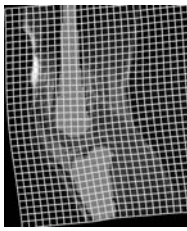
The first criterion measures the relative variation in the objective function, the second criterion measures the relative variation in the parameters, and the third measures the norm of the gradient. If all these numbers are small, the iteration is terminated. As safeguards serve $\|\texttt{dy}\| \leq \text{eps}$ and iter>maxIter, where eps denotes the machine precision and maxIter is an a priori chosen number. See, e.g., [109] for a detailed discussion.

Solving the Quasi-Newton System

Using the MATLAB backslash operator works fine for a moderate number of parameters. For higher-dimensional problems, iterative schemes can be used with advantage. For example, a conjugate-gradient-type scheme can exploit the sparsity structure of the building blocks and does not require one to explicitly form the Hessian [137, 69]. The FAIR implementation calls a function solveGN which can be parameterized to use different solvers. However, the default for parametric registration is the MATLAB backslash operation.

Armijo's Line Search

In order to ensure a sufficient decrease in the objective function, a line search procedure is required; see, e.g., [165]. FAIR uses a standard Armijo line search; see



Armijo. Starting with $t = 1$, the candidate $y^t = y + t \cdot dy$ is tested. If the reduction of the objective function is not sufficient, the procedure is iterated, replacing t by $\frac{1}{2}t$. The standard sufficiency criterion $J(y^t) < J(y) + \texttt{tol}\ t\ (dJ^\top y)$ with standard choice $\texttt{tol} = 10^{-4}$ is used. As a safeguard, the line search as well as the iteration are terminated if a step becomes too small. In this situation, the optimization algorithm is declared to fail to converge; see Armijo.

FAIR 12: Gauss–Newton Optimization with Armijo's Line Search

This is a Quasi-Newton-type optimization technique using Armijo's line search for numerically minimizing an objective function `[J,para,dJ,H]=fctn(yc)`, where `yc` denotes the current iterate, `J` and `dJ` denote the function value and the derivative, and `H` is an approximation to the Hessian. Note that $\texttt{dJ} = \nabla J(\texttt{yc})^\top$. The structure `para` can be used for an efficient visualization of intermediate results; see Section 6.3.4.

function [yc,his]= GaussNewtonArmijo(fctn,yc,varargin)	
fctn	handle to the objective function
yc	current iterate
varargin	optional parameters controlling the stopping rule, etc.
yc	numerical optimizer
his (cell)	iteration history

6.3.4 Brief Comments on a Visualization

Although the optimization is generic and completely decoupled from the registration problem, it is nice to visualize some intermediate results. Unfortunately, this is not so easy since, in contrast to the generic optimization algorithms, the visualization very much depends on the data. FAIR provides the function FAIRplots which compromises between convenience and simplicity. A discussion of the functionality is beyond the scope of this book; the interested reader is referred to the source code. Instead, the basic modes are explained. In order to produce skim calls, the variables are collected in a structure

```
para = struct(...
'Tc',Tc,'Rc',Rc,'omega',omega,'m',m,'yc',yc,'Jc',Jc),
```

where `yc= trafo(wc,xc)`, `Tc= inter(T,omega,yc)`, `Rc= inter(R,omega,xc)`, and `Jc` $= J($`wc`$)$.

Setup A typical setup call reads

```
FAIRplots('set','mode',figureName,'fig',figureNumber)
```

and produces no graphical output but initializes various persistent variables such as, for example, the figure number, the figure name, and function handles to the

data viewers, etc. The basic layout is a 2-by-3 array of subplots which generically reads

$R(\texttt{xc})$	$T(\texttt{y0})$	$T(\texttt{yc})$
$T(\texttt{xc})$&grid	$T(\texttt{y0}) - R(\texttt{xc})$	$T(\texttt{yc}) - R(\texttt{xc})$

Initialization The initialization call reads `FAIRplots('init',para)`, assuming `yc` = `xc`. This call visualizes $R(\texttt{xc})$, $T(\texttt{xc})$, and $T(\texttt{xc}) - R(\texttt{xc})$ in the subplots $1, 2, 4, 5$.

Stopping An import item in particular in a multilevel framework is stopping; see Section 6.6. It is assumed that `wc=wStop`, `yc=trafo(wc,xc)`, `Tc=inter(T,omega,yc)` and that a call `FAIRplots('stop',para)` displays `Tc` and `Tc` − `Rc` in subplots 3 and 6 and visualizes the grid `yc` in subplot 4. A persistent variable `Jstop` = `Jc` = $J(\texttt{wStop})$ is initialized for use in later calls.

Starting It is assumed that `wc=w0`, `yc=trafo(wc,xc)`, `Tc=inter(T,omega,yc)` and that a call `FAIRplots('start',para)` displays `Tc` and `Tc` − `Rc` in subplots 2 and 5 and visualizes the grid `yc` in subplot 4. The ratio `Jc/JStop` is displayed as a percentage.

Iteration It is assumed that `yc=trafo(wc,xc)`, `Tc=inter(T,omega,yc)` and that a call `FAIRplots(iter,para)` displays `Tc` and `Tc` − `Rc` in subplots 3 and 6 and visualizes the grid `yc` in subplot 4. The ratio `Jc/JStop` is displayed as a percentage. The value of `iter` indicates the iteration.

6.3.5 PIR Examples

Two basic examples are presented, and more examples are provided in the next section.

Example 6.7 (PIR: Plain and Simple)
This example illustrates how to use `GaussNewtonArmijo` for a particular application; see also Table 6.2. The first lines in the following script file initializes the data, image viewer (based on the information from the data setup), interpolator (spline interpolation), transformation (affine), starting guess (`w0=[1;0;0;0;1;0]`), and distance measure (SSD). The graphical output is prepared. A handle to the objective function is created. Note that the interpolated reference image is used to avoid an interpolation in every call of the objective function and that the optional regularization has been disabled by setting `M=[]` and `wRef=[]`. Finally, the Gauss–Newton scheme is used to compute a numerical minimizer of the objective function. Results are shown in Table 6.3.

The above program also creates two figures, shown in Figure 6.6. One figure shows the reference $R(\texttt{xc})$, template $T(\texttt{xc})$, transformed template $T(\texttt{yc})$, the template with the transformed grid `yc`, the initial difference $|T(\texttt{xc}) - R(\texttt{xc})|$, and the final difference $|T(\texttt{yc}) - R(\texttt{xc})|$ and reports some numerical values. The second figure visualizes the iteration history.

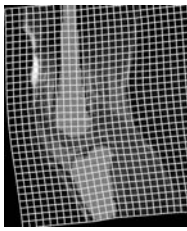
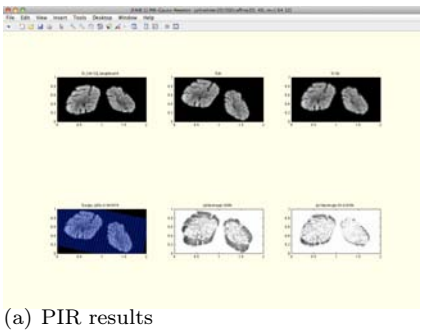


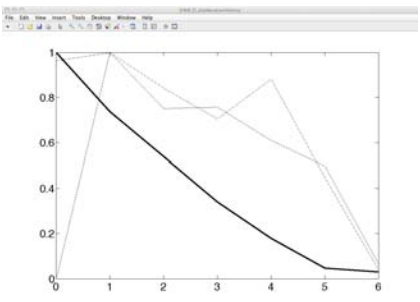
Table 6.2 Parametric image registration on a fixed level

This file is E6_HNSP_PIR_GN.m

```
%%%%
% set up data (MultiLevel based) and initialize image viewer
setupHNSPData;

% initialize the interpolation scheme and coefficients
inter('set','inter','splineInter2D');
level = 4; omega = MLdata{level}.omega; m = MLdata{level}.m;
[T,R] = inter('coefficients',MLdata{level}.T,MLdata{level}.R,omega);
xc = getCenteredGrid(omega,m);
Rc = inter(R,omega,xc);

% initialize distance measure
distance('set','distance','SSD');

% initialize the transformation and a starting guess
trafo('reset','trafo','affine2D');
w0 = trafo('w0');

% set up plots and initialize
FAIRplots('set','mode','PIR-Gauss-Newton','omega',omega,'m',m,'fig',1,'plots',1);
FAIRplots('init',struct('Tc',T,'Rc',R,'omega',omega,'m',m));

% build objective function
% note: T  is template image
%       Rc is sampled reference
%       optional Tychonoff-regularization is disabled by setting m = [], wRef = []
%       beta = 0 disables regularization of Hessian approximation
beta = 0; M = []; wRef = [];
fctn = @(wc) PIRobjFctn(T,Rc,omega,m,beta,M,wRef,xc,wc);
fctn([]);   % report status

% -- solve the optimization problem ---------------------------------------
[wc,his] = GaussNewtonArmijo(fctn,w0,'Plots',@FAIRplots,'solver',[],'maxIter',100);
plotIterationHistory(his,'J',[1,2,5],'fig',20+level);
```



(a) PIR results



(b) PIR iteration history

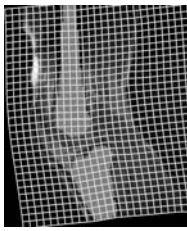
Figure 6.6: *Plots from PIR.*

Table 6.3 Parametric image registration results.

The first four lines report the data source, the initialization of the interpolation scheme and the image viewer, and the computation of the spline coefficients. Calling the objective function with an empty argument `fctn([])` reports the current configuration. The optimizer reports parameter choices and the iteration history showing six columns with the iteration number, the current improvement (difference between old and current function values), the norm of the gradient, the norm of the update, and the number of line search (LS) iterations. The first row (`iter=-1`) reports on the stopping value (which can be different from the initial value; see Section 6.6). The last five lines report on the stopping criteria. The first character (`0/1`) is a Boolean indicating the value of the expression embraced by square brackets. In this example, the function terminates because the maximum number of iterations has been performed.

PIRR output

```
% ----------  [ setupHNSPData ]  ---------------------------------------------
load(/Users/Jan/JM/SVN/FAIR_m/mfiles/examples/setupHNSPData.mat)
  initialize inter and viewImage
compute 2D spline coefficients, m=[ 64 32], regularizer=[none], theta=[0]
compute 2D spline coefficients, m=[ 64 32], regularizer=[none], theta=[0]
Parametric Image Registration:    J(w)=D(T(y(w)),R) + (w-wRef)'*M*(w-wRef) != min
                    m : [ 64 32]
                omega : [ 2 1]
          INTERPOLATION : splineInter2D
               DISTANCE : SSD
                  TRAFO : affine2D
             length(wc) : 6
                   beta : 0
% ----------  [ GaussNewtonArmijo(JM 2008/08/05) ]  ----------------------------
[ maxIter=10 / tolJ=0.001 / tolY=0.01 / tolG=0.01 / length(Yc)=6 ]
iter J            Jold-J       |\nabla J|   |dY|          LS
-------------------------------------------------------------------
  -1 4.0560e+03   0.000e+00    0.000e+00    0.000e+00       0
   0 4.0560e+03   0.000e+00    1.307e+04    0.000e+00       0
   1 3.7859e+03   2.702e+02    1.597e+04    4.063e-02       1
   2 3.5372e+03   2.486e+02    1.254e+04    4.419e-02       1
   3 3.3229e+03   2.144e+02    1.233e+04    3.541e-02       1
   4 3.0346e+03   2.883e+02    1.233e+04    4.106e-02       1
   5 2.7783e+03   2.563e+02    1.358e+04    3.982e-02       1
   6 2.5201e+03   2.582e+02    1.346e+04    4.219e-02       1
   7 2.3077e+03   2.124e+02    1.284e+04    3.447e-02       1
   8 2.0913e+03   2.164e+02    1.133e+04    3.491e-02       1
   9 1.8491e+03   2.422e+02    1.203e+04    3.701e-02       1
STOPPING:
0[ (Jold-Jc) =  2.82180873e+02 <= tolJ*(1+|Jstop|)         =  4.05704499e+00]
0[ |Yc-Yold| =  4.39777040e-02 <= tolY*(1+norm(Yc))        =  2.47181777e-02]
0[ |dJ|      =  1.04498380e+04 <= tolG*(1+abs(Jstop)       =  4.05704499e+01]
0[ norm(dJ)  =  1.04498380e+04 <= eps                      =  2.22044605e-13]
1[ iter      =  10             >= maxIter                  =  10            ]
% ----------  [ GaussNewtonArmijo : done ! ]  ----------------------------------
```



Example 6.8 (PIR: Using a Scale-Space)
This example presents a modification of the previous example that takes advantage of the multiscale idea: starting with a very smooth representation of the data, the optimizer serves as a starting point for a less smoothed presentation.

This file is E6_HNSP_PIR_scale.m

```
%%%%
setupHNSPData;
inter('set','inter','splineInter2D');
level = 5; omega = MLdata{level}.omega; m = MLdata{level}.m;
[T,R] = inter('coefficients',MLdata{level}.T,MLdata{level}.R,omega);
xc = getCenteredGrid(omega,m);
Rc = inter(R,omega,xc);

distance('set','distance','SSD');       % initialize distance measure

trafo('reset','trafo','affine2D');
w0 = trafo('w0');

FAIRplots('set','mode','PIR-affine','omega',omega,'m',m,'fig',1,'plots',1);
FAIRplots('init',struct('Tc',T,'Rc',R,'omega',omega,'m',m));

% -- solve the optimization problem on different scales ----------------------

wStop = w0;             % use one GLOBAL stopping criterion
theta = [1e2,1e1,1,0];  % the different scales: smooth to detailed
for j=1:length(theta),
  % initilaize the data for scale theta(j)
  inter('reset','inter','splineInter2D','regularizer','moments','theta',theta(j));
  [T,R] = inter('coefficients',MLdata{level}.T,MLdata{level}.R,omega);

  % set-up plots and initialize it
  FAIRplots('set','mode','PIR-Gscale','fig',j,'plots',1);
  FAIRplots('init',struct('Tc',T,'Rc',R,'omega',omega,'m',m));

  % initialize optimizer
  Rc   = inter(R,omega,xc); % note Rc depends on the scale as well
  fctn = @(wc) PIRobjFctn(T,Rc,omega,m,0,[],[],xc,wc);
  if j ==1, fctn([]); end;  % report status

  % solve problem for this scale
  [wc,his] = GaussNewtonArmijo(fctn,w0,'yStop',wStop,'Plots',@FAIRplots);
  % and use solutions as starting guess for next scale
  w0 = wc;
end;
```

6.4 PIR Experiments on Fixed Levels

The data used in this section is provided by setupHNSPdata, and the multilevel representation is generated using getMultilevel. Here, $\Omega = (0,2) \times (0,1)$, a spline-based interpolation approach, the SSD distance measure, and no additional regularization of the parameter are used. The reduction of the distance is measured by

$$\text{reduction} = J(\texttt{wc})/J(\texttt{w0}), \tag{6.5}$$

where w0 is the starting guess and wc is the numerical optimizer.

The experiments are performed for different parametric transformations and are based either on coarse level or fine level representation of the data:

	$\ell = \texttt{level}$	m_ℓ
coarse	4	$[32, 16]$
fine	7	$[256, 128]$

For ease of presentation, the dependence of xc, yc, T, and R on ℓ is not indicated.

The results obtained for the coarse and fine levels are very close, although the optimization on the fine level is much more expensive: more iterations are needed and the computation consumes much more time. This motivates a multilevel strategy, as discussed in the following section.

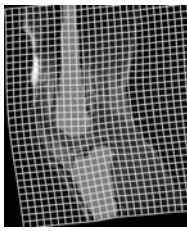
Example 6.9 (PIR: Rotations About the Center of the Domain)
The transformation is a rotation about the center of the domain and is parameterized by $w \in \mathbb{R}$. This first experiment is on the coarse level; results are shown in Figure 6.7. The numerical minimizer is obtained after 8 iterations, and the initial reduction is about 8%. The second experiment is on the finer level; results are shown in Figure 6.8. The numerical minimizer is obtained after 81 iterations, and the initial reduction is about 18%. The convergence history is presented in Figure 6.9.

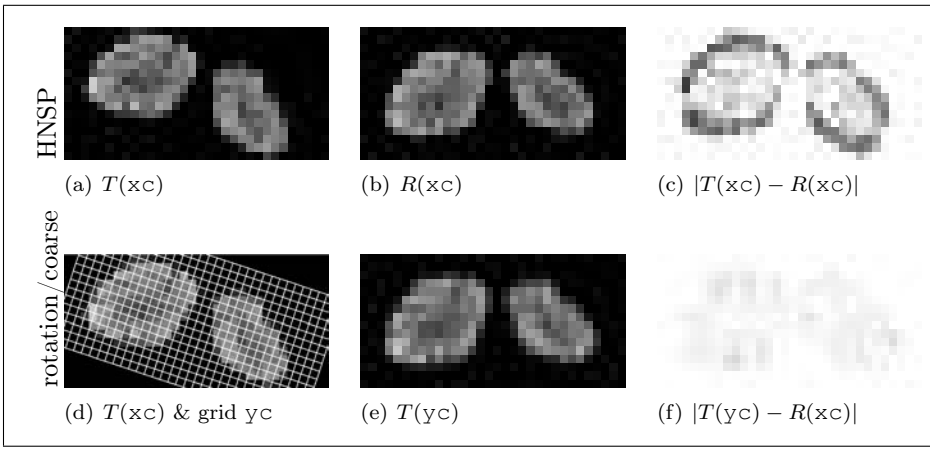
This file is E6_HNSP_PIR_SSD_rotation2D_level4.m

```
%%%%
setupHNSPData;
inter('set','inter','splineInter2D');
level = 4; omega = MLdata{level}.omega; m = MLdata{level}.m;
[T,R] = inter('coefficients',MLdata{level}.T,MLdata{level}.R,omega);
xc = getCenteredGrid(omega,m);
Rc = inter(R,omega,xc);

distance('set','distance','SSD');       % initialize distance measure

center = (omega(2:2:end)-omega(1:2:end))'/2;
trafo('reset','trafo','rotation2D','c',center);
w0 = trafo('w0');

FAIRplots('set','mode','PIR-rotation','omega',omega,'m',m,'fig',1,'plots',1);
FAIRplots('init',struct('Tc','T,'Rc',R,'omega',omega,'m',m));

% ----- call Gauss-Newton ----------------------------------
GNoptn = {'maxIter',500,'Plots',@FAIRplots};
fctn  = @(wc) PIRobjFctn(T,Rc,omega,m,0,[],[],xc,wc);
[wc,his] = GaussNewtonArmijo(fctn,w0,GNoptn{:});

figure(1); clf
viewImage(inter(T,omega,xc),omega,m,'axis','off'); hold on
ph = plotGrid(trafo(wc,xc),omega,m,'spacing',1,'linewidth',1,'color','w');

% plot iteration history
his.str{1} = sprintf('iteration history PIR: distance=%s, y=%s',distance,trafo);
[ph,th] = plotIterationHistory(his,'J',1:4,'fig',2);
```

Example 6.10 (PIR: Rigid Transformations)
A rigid transformation is used in these examples, parameterized by $w \in \mathbb{R}^3$. Results for the coarse level and the fine level are shown in Figures 6.10 and 6.11, respectively. The convergence history is presented in Figure 6.12.

On the coarse level, the numerical minimizer is obtained after 7 iterations and the initial reduction is about 2% ($\texttt{wc} \approx [-0.3119, -0.1233, 0.3534]$) whereas for the fine level, the numerical minimizer is obtained after 72 iterations and the initial reduction is about 4% ($\texttt{wc} \approx [-0.3130, -0.1156, 0.3444]$).

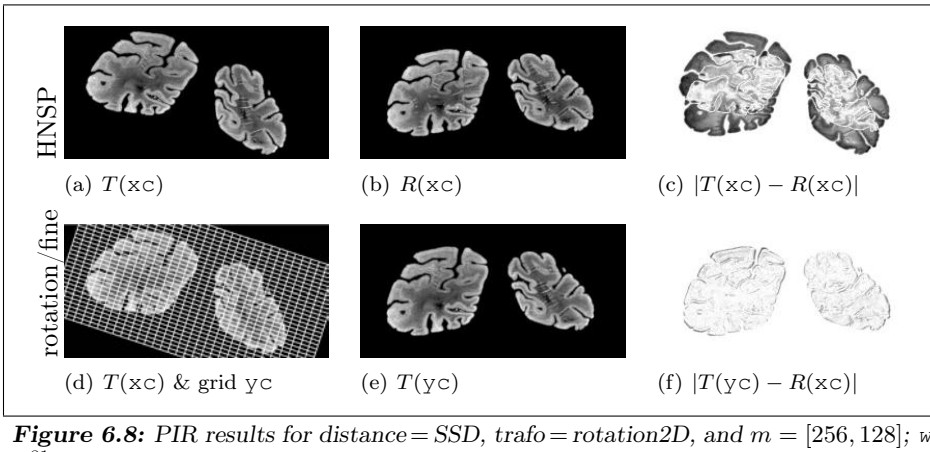




(a) $T(\mathtt{xc})$ (b) $R(\mathtt{xc})$ (c) $|T(\mathtt{xc}) - R(\mathtt{xc})|$

(d) $T(\mathtt{xc})$ & grid $\mathtt{yc}$ (e) $T(\mathtt{yc})$ (f) $|T(\mathtt{yc}) - R(\mathtt{xc})|$

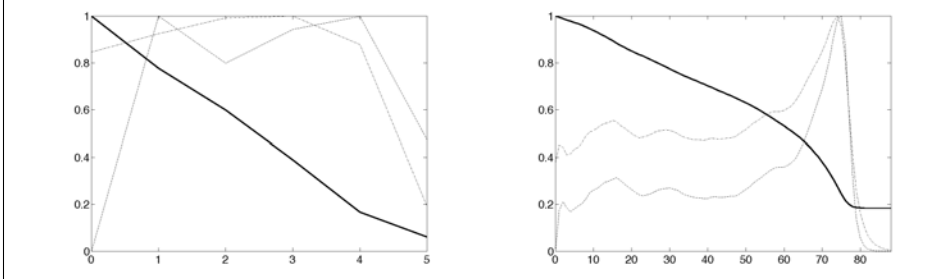
Figure 6.7: *PIR results for distance = SSD, trafo = rotation2D, and* $m = [32, 16]$*;* $\mathtt{yc} = y(\mathtt{wc}, \mathtt{xc})$*,* $\mathtt{wc} = w^7 \approx -0.3229$*, reduction* $\approx 8.4\%$*.*



(a) $T(\mathtt{xc})$ (b) $R(\mathtt{xc})$ (c) $|T(\mathtt{xc}) - R(\mathtt{xc})|$

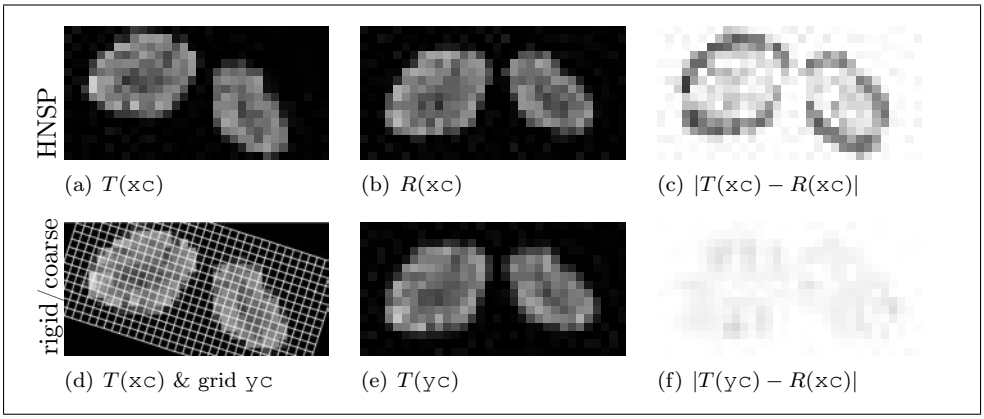
(d) $T(\mathtt{xc})$ & grid $\mathtt{yc}$ (e) $T(\mathtt{yc})$ (f) $|T(\mathtt{yc}) - R(\mathtt{xc})|$

Figure 6.8: *PIR results for distance = SSD, trafo = rotation2D, and* $m = [256, 128]$*;* $\mathtt{wc} = w^{81} \approx -0.3202$*, reduction* $\approx 18.5\%$*.*



(a) iteration history for $[32, 16]$ (b) iteration history for $[256, 128]$

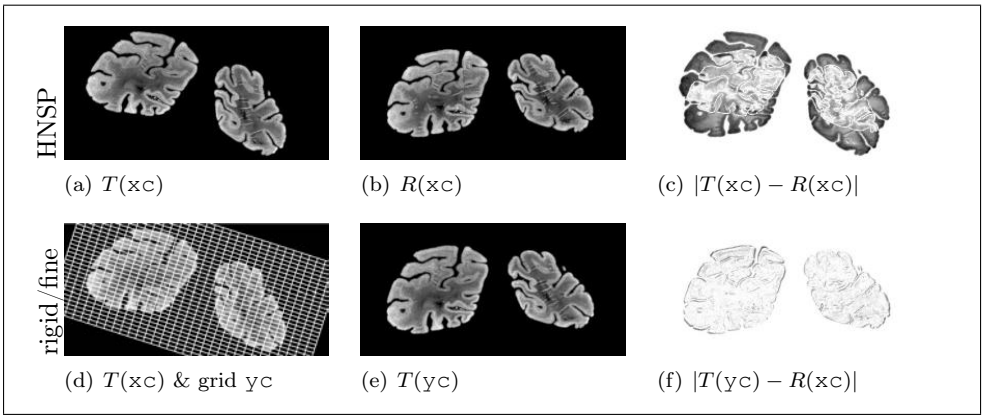
Figure 6.9: *Iteration history for PIR with SSD and rotation2D: coarse (left) and fine (right) discretization;* $J(w^k)$ *(solid line),* $\|w^k - w^{k-1}\|$ *(dashed line), and* $\|d_w J(w^k)\|$ *(dashed dotted line); all numbers are relative.*



(a) $T(\mathtt{xc})$ (b) $R(\mathtt{xc})$ (c) $|T(\mathtt{xc}) - R(\mathtt{xc})|$

(d) $T(\mathtt{xc})$ & grid $\mathtt{yc}$ (e) $T(\mathtt{yc})$ (f) $|T(\mathtt{yc}) - R(\mathtt{xc})|$

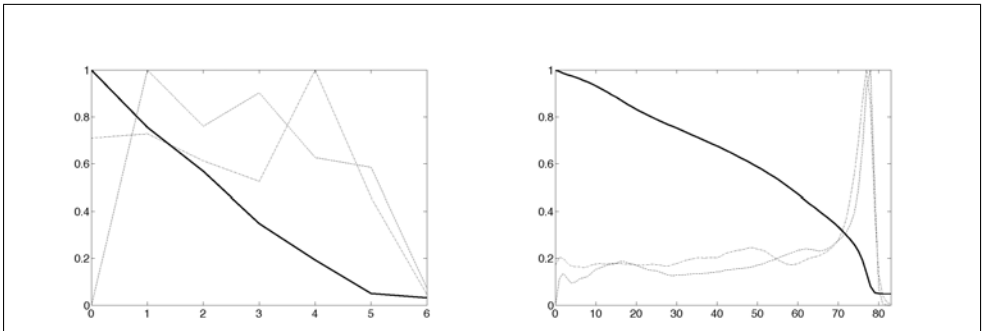
Figure 6.10: *PIR results for distance = SSD, trafo = rigid2D, and* $m = [32, 16]$*;* $\mathtt{wc} = w^6 \approx [-0.3119, -0.1233, 0.3534]$, reduction $\approx 2\%$.



(a) $T(\mathtt{xc})$ (b) $R(\mathtt{xc})$ (c) $|T(\mathtt{xc}) - R(\mathtt{xc})|$

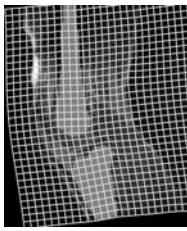
(d) $T(\mathtt{xc})$ & grid $\mathtt{yc}$ (e) $T(\mathtt{yc})$ (f) $|T(\mathtt{yc}) - R(\mathtt{xc})|$

Figure 6.11: *PIR results for distance = SSD, trafo = rigid2D, and* $m = [256, 128]$*;* $\mathtt{wc} = w^{72} \approx [-0.3130, -0.1156, 0.3444]$, reduction $\approx 4.4\%$.



(a) iteration history for $[32, 16]$ (b) iteration history for $[256, 128]$

Figure 6.12: *Iteration history for PIR with SSD and rigid2D: coarse (left) and fine (right) discretization;* $J(w^k)$ *(solid line),* $\|w^k - w^{k-1}\|$ *(dashed line), and* $\|d_w J(w^k)\|$ *(dashed dotted line); all numbers are relative.*



6.5 Regularized Parametric Image Registration

This section explains the needs for the additional regularization but may be skipped in a first reading.

Additional regularization can be an interesting option for low-dimensional transformation spaces. A regularizer can be used to create a bias towards a particular solution or prohibit an unwanted solution. However, regularization becomes inevitable if the transformation space is high dimensional, as is indicated by the following example.

Example 6.11 (PIR: Nonregularized Spline Transformations)
The script file listed in Table 6.4 runs a parametric spline registration; the result is shown in Figure 6.13. Although the image distance has been reduced considerably,

Table 6.4 Parametric spline registration.

This file is E6_HNSP_PIR_spline2D_level5.m

```
%%%%
setupHNSPData;
inter('set','inter','splineInter2D');
level = 5; omega = MLdata{level}.omega; m = MLdata{level}.m;
[T,R] = inter('coefficients',MLdata{level}.T,MLdata{level}.R,omega);
xc = getCenteredGrid(omega,m);
Rc = inter(R,omega,xc);

distance('set','distance','SSD');       % initialize distance measure

center = (omega(2:2:end)-omega(1:2:end))'/2;
trafo('reset','trafo','rotation2D','c',center);

% initialize transformation and starting guess;
% here: spline with 2 times [4,5] coefficients
trafo('reset','trafo','splineTransformation2D','omega',omega,'m',m,'p',[4,5]);
w0 = trafo('w0');

FAIRplots('set','mode','PIR-spline','omega',omega,'m',m,'fig',1,'plots',1);
FAIRplots('init',struct('Tc',T,'Rc',R,'omega',omega,'m',m));

% ----- call Gauss-Newton ----------------------------------
GNoptn = {'maxIter',50,'Plots',@FAIRplots};
fctn  = @(wc) PIRobjFctn(T,Rc,omega,m,0,[],[],xc,wc);
[wc,his] = GaussNewtonArmijo(fctn,w0,GNoptn{:});

figure(1); clf
viewImage(inter(T,omega,xc),omega,m,'axis','off'); hold on
ph = plotGrid(trafo(wc,xc),omega,m,'spacing',1,'linewidth',1,'color','w');
```

(a) $T(\mathtt{xc})$ & grid $\mathtt{yc}$ (b) $|T(\mathtt{xc}) - R(\mathtt{xc})|$ (c) $|T(\mathtt{yc}) - R(\mathtt{xc})|$

Figure 6.13: *PIR results for SSD and spline transformation, $m = [32, 16]$ and $p = [4, 5]$; $yc = y(wc, xc)$.*

the transformation does not look appealing since it is not one-to-one.

To circumvent an unwanted solution, a penalty or regularizer S is added to the objective function. The theoretical necessity for an additional penalty is discussed in Chapter 8 in much more detail. In this section, simple ideas which have already been discussed in the interpolation chapter, Chapter 3, serve as a starting point; cf. Section 3.4. To keep the discussion focused, a spline transformation is considered,

$$y^i(x) = x^i + Q^i(x)w^i, \quad i = 1, \dots, d,$$

where the matrices Q^i are generated from a spline basis as discussed in Section 4.5; see also `splineTransformation2D`. Moreover, a norm of the coefficients is used for regularization:

$$S(w) = \frac{1}{2}\, w^\top M w$$

with a symmetric positive definite matrix M. The particular choice $M = \alpha I$ results in penalizing the displacement $u^i = y^i(x) - x^i$. The regularization parameter $\alpha > 0$ is integrated in M.

Remark 6.1
The above regularization is a particular specification of the more general approach

$$\mathcal{S}[u^i] = \int_\Omega \left\langle \mathcal{B}u^i, \mathcal{B}u^i \right\rangle \, dx,$$

where $\mathcal{B}$ is some differential operator. Substituting the parametric form of u^i,

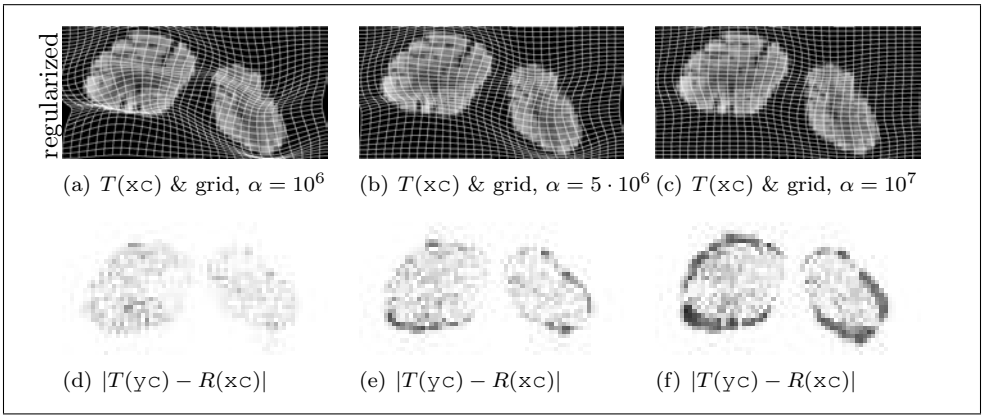
$$\begin{aligned} S(w^i) = \mathcal{S}[Q^i w^i] &= \int_\Omega \left\langle \textstyle\sum_j \mathcal{B}q^j w_j^i, \sum_k \mathcal{B}q^k w_k^i \right\rangle \, dx \\ &= \sum_{j,k} w_j^i w_k^i \int_\Omega \left\langle \mathcal{B}q^j, \mathcal{B}q^k \right\rangle \, dx \\ &= (w^i)^\top M w^i, \quad \text{where} \quad M_{j,k} = \int_\Omega \left\langle \mathcal{B}q^j, \mathcal{B}q^k \right\rangle \, dx. \end{aligned}$$

This approach is closely related to the discussion on multiscale interpolation in Section 3.6. The regularizer used in (3.20) is based on the second derivative: $\mathcal{B}q^k = (q^k)''$.

With a regularization parameter $\alpha > 0$, compromising between similarity and reasonability, the joint objective function is

$$J(w) = \mathcal{D}[\mathcal{T}[y(w)], \mathcal{R}] + \alpha S(w). \tag{6.6}$$

Results for the regularized approach for three different values of α are shown in Figure 6.14. Note that increasing α pushes the transformation towards the identity and increases the distance.



(a) $T(\mathtt{xc})$ & grid, $\alpha = 10^6$ (b) $T(\mathtt{xc})$ & grid, $\alpha = 5 \cdot 10^6$ (c) $T(\mathtt{xc})$ & grid, $\alpha = 10^7$

(d) $|T(\mathtt{yc}) - R(\mathtt{xc})|$ (e) $|T(\mathtt{yc}) - R(\mathtt{xc})|$ (f) $|T(\mathtt{yc}) - R(\mathtt{xc})|$

Figure 6.14: *Regularized PIR for SSD and spline transformations;* $m = [64, 32]$, $p = [8, 8]$, $M = \alpha I$, $\alpha = k \cdot 10^6$, $k = 1, 5, 10$, $yc = y(wc, xc)$.

6.6 Multilevel Parametric Image Registration

The experiments of the previous sections strongly suggest using multilevel parametric image registration (MLPIR). Starting on a coarse level where computations are cheap, a starting guess for a finer level is computed. On the fine level, only a very few correction steps are expected. The tools needed are provided by `getMultilevel` and `MLPIR`.

The basic idea is to obtain smoother representations of the images. Therefore, the data on a coarser level is replaced by a mean value or an average over adjacent cells. Figures 6.15(a)–(c) shows interpolations of the data $\mathtt{T}^\ell$ for level $\ell = 3, 5, 7$; see also Section 3.7. Obviously, the functions $\mathcal{T}^\ell(x) = \mathtt{inter}(\mathtt{T}^\ell, \mathtt{omega}, x)$ are smoother on coarser levels. Using a multilevel approach offers three major advantages. First, the optimization problems are easier to solve on the coarser levels, i.e., less iterations are needed to compute a minimizer. Second, details are diminished on coarser levels and the optimization is thus more robust, and the risk of being tracked in local minima is reduced. Third, the quadrature rule for the smoother functions $\mathcal{T}^\ell$ can be based on coarse discretizations, which reduces even further the computation costs needed for one iteration on the coarse level; see Figures 6.15(d)–(f).

The stopping criteria are not yet compatible with a multilevel strategy. Supplying the solution of a coarse level as a starting guess for a finer level, the initial guesses become better and better. As a consequence, the first stopping criterion becomes unacceptably hard. The remedy is to replace this criterion by

```
STOP1=(Jold-Jc)<=tolJ*(1+abs(JStop)),
```

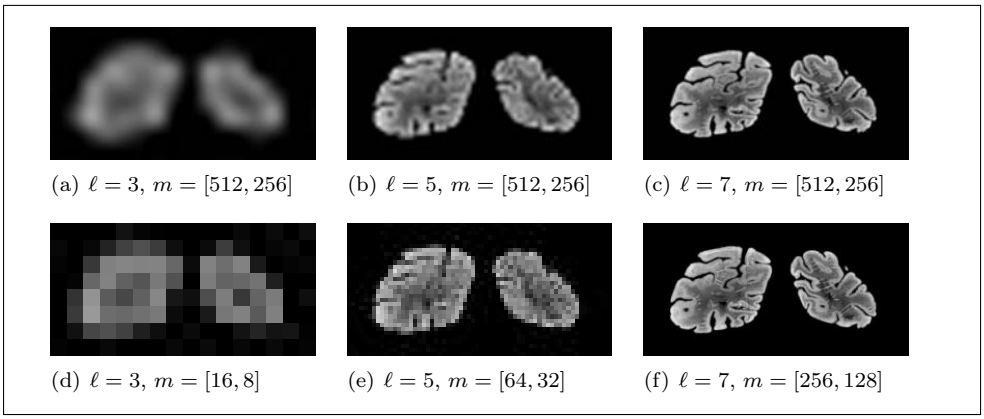
where `JStop` is obtained from a global starting guess `wStop`, which stays constant with respect to levels.

Example 6.12 (MLPIR: SSD and Rigid Transformations)
This example continues the experiments of Section 6.10. Starting on level $\ell = 3$, a numerical minimizer is computed in 4 iterations of PIR and serves as a starting

guess for level $\ell = 4$. The iteration history is summarized in Figure 6.16. As it turns out, the starting guess on the finest level $\ell = 8$ already satisfies the stopping criteria. Results are shown in Figure 6.17.

This file is E6_HNSP_MLPIR_SSD_rigid2D.m

```
%%%%
setupHNSPData;                              % set up data
distance('reset','distance','SSD');         % specify distance measure
inter('reset','inter','splineInter2D');     % specify interpolator
trafo('reset','trafo','rigid2D');           % specify transformation
[wc,his] = MLPIR(MLdata,'minLevel',3,'plotMLiter',1);
```



(a) $\ell = 3$, $m = [512, 256]$ (b) $\ell = 5$, $m = [512, 256]$ (c) $\ell = 7$, $m = [512, 256]$

(d) $\ell = 3$, $m = [16, 8]$ (e) $\ell = 5$, $m = [64, 32]$ (f) $\ell = 7$, $m = [256, 128]$

Figure 6.15: *Multilevel representation of the data $\mathcal{T}^\ell$ (a–c) and the functions $\mathcal{T}^\ell(xc^\ell)$, $\ell = 3, 5, 7$ (d–f).*

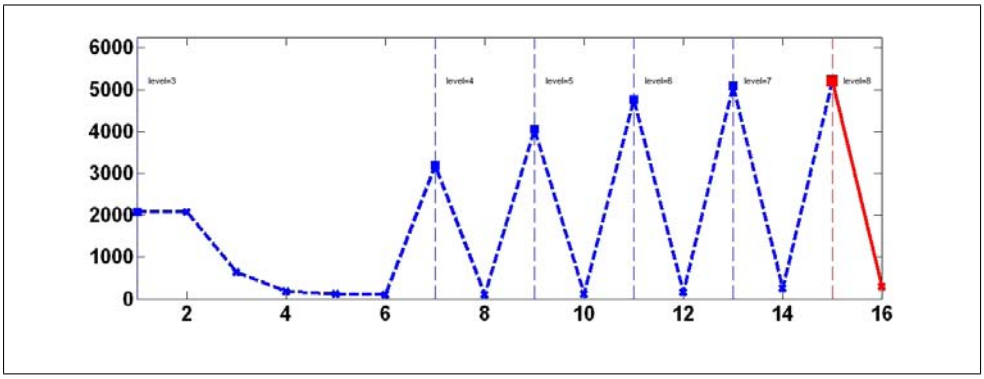


Figure 6.16: *Iteration history for MLPIR with SSD and rigid2D, $D(y(w^k))$ versus k: vertical dashed lines separate different levels; reference values on the ℓth level are shown as squares; iterations as crosses; on finest level ($\ell = 8$, solid line) only a comparison step is needed.*

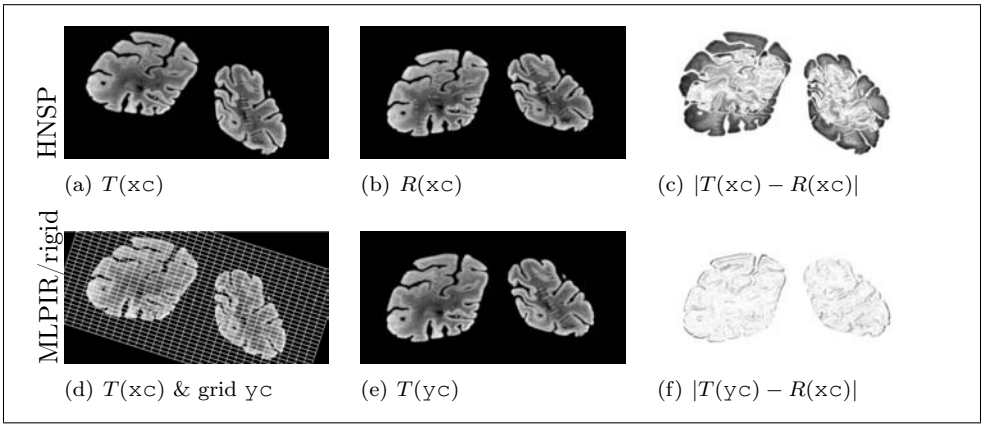


Figure 6.17: *MLPIR results for distance = SSD, trafo = rigid2D, and* $m = [512, 256]$*;* `yc = trafo(wc,xc)`, `wc` $\approx [-0.3128, -0.1153, 0.3440]$, reduction $\approx 5.8\%$.

Example 6.13 (MLPIR: SSD and Affine Linear Transformations)
In this example, an affine linear transformation model is considered; see `affine2D`. As in the previous example, only a comparison step is required on the finest level $\ell = 8$. Figure 6.18 shows the iteration history and Figure 6.19 shows the results.

This file is `E6_HNSP_MLPIR_SSD_affine2D.m`

```
%%%%
setupHNSPData;                              % set up data
distance('reset','distance','SSD');         % specify distance measure
inter('reset','inter','splineInter2D');     % specify interpolator
trafo('reset','trafo','affine2D');          % specify transformation
[wc,his] = MLPIR(MLdata,'minLevel',3,'plotMLiter',1);
```

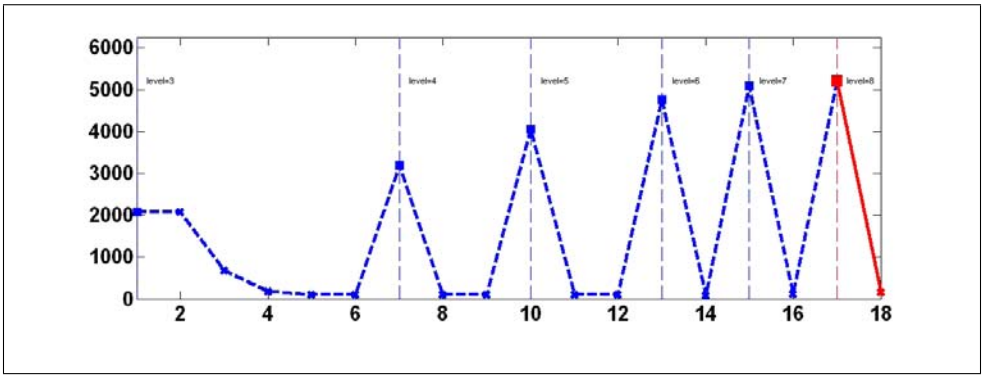
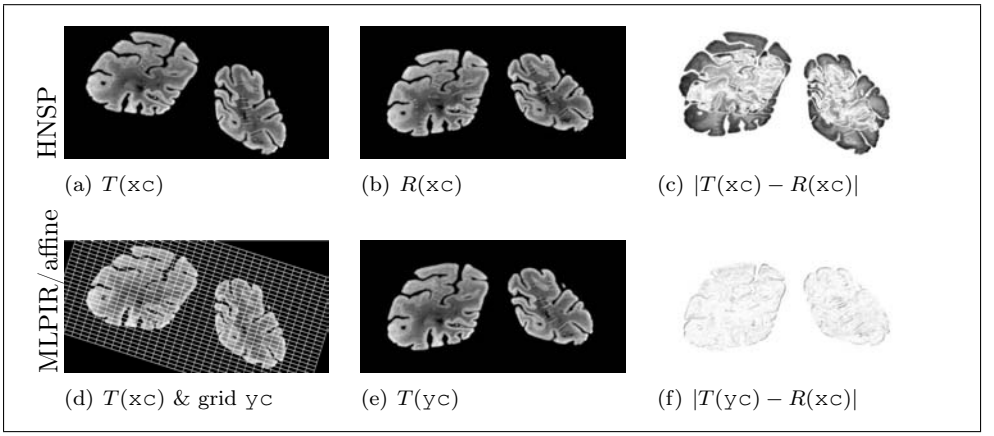


Figure 6.18: *Iteration history for MLPIR with SSD and affine2D,* $J(w^k)$ *versus* k*: vertical dashed lines separate different levels; reference values on the* ℓ*th level are shown as squares; iterations as crosses; on finest level* ($\ell = 8$) *only a comparison step is needed.*



(a) $T(\mathtt{xc})$ (b) $R(\mathtt{xc})$ (c) $|T(\mathtt{xc}) - R(\mathtt{xc})|$

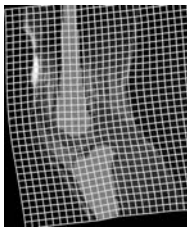
(d) $T(\mathtt{xc})$ & grid $\mathtt{yc}$ (e) $T(\mathtt{yc})$ (f) $|T(\mathtt{yc}) - R(\mathtt{xc})|$

Figure 6.19: *MLPIR results for distance = SSD, trafo = affine2D, and* $m = [512, 256]$*;* `yc = trafo(wc,xc)`, `wc` $\approx [0.959, 0.305, -0.122, -0.310, 0.966, 0.339]$, reduction ≈ 0.0314.

6.7 Summarizing Parametric Image Registration Topics

In this chapter the workhorse for the minimization of an integral-based distance measure has been discussed. Using a midpoint quadrature rule, it has been shown how to discretize integrals. Based on the SSD, i.e., the energy of the difference image between reference and transformed template, it has been shown exemplarily how to derive a discretized distance measure. The discretized measure has been combined with parametric transformations, and the distance can be considered as a function in the parameters. A minimizer of this function yields optimal parameters. The optimal parameters are computed using a Gauss–Newton-type optimization scheme. For high-dimensional transformations, an additional regularization has been introduced.

Emphasis has been given to the multilevel strategy. Instead of solving the optimization problem for an a priori fixed discretization width h, a sequence of nested problems ranging from coarse to fine is solved. The key point is to get a smooth representation of the images on a coarse grid. This yields a smooth objective function which is easy to optimize and, in addition, requires only a coarse discretization. The solution of the coarse problem serves as a perfect starting point for the fine problem. Since the starting guess is expected to be close to the optimizer on the fine level, the danger of being trapped by a local minima is reduced. Moreover, by exploiting a fast optimization scheme, only a very few correction steps are expected to be necessary. The procedure is stopped if the variation in the transformations is below a prescribed tolerance, or if the discretization width is below the discretization width of the data.



6.8 FAIR Tutorials on Parametric Image Registration

FAIR contains the tutorial `BigTutorialPIR` which provides a number of smaller tutorials explaining how to use the PIR.

BigTutorialPIR	
`E7_Hands_SSDvsRotation`	SSD versus rotation, hands
`E6_Hands_PIR_SD`	PIR, hands, rotations, SSD, steepest descent
`E6_Hands_PIR_GN`	PIR, hands, rotations, SSD, Gauss–Newton
`E6_Hands_MLPIR_pause`	MLPIR, hands, affine, SSD, including pauses
`E6_Hands_MLPIR`	MLPIR, hands, affine, SSD
`E6_HNSP_MLPIR_reg`	MLPIR, HNSP, splines, regularized
`E6_PETCT_MLPIR`	MLPIR, PETCT, affine, SSD

6.9 Exercises

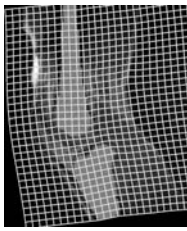
Exercise 6.1
Use the framework of `setupHNSPdata` *to set up your own example.*

Exercise 6.2
Use the framework of Section 6.3.5 to register your data with various transformations including rotation, rigid, affine, and spline transformations.

Exercise 6.3
Use the framework of Chapter 4 to create a quadratic transformation. Use PIR to compute optimal parameters.

Exercise 6.4
Write a steepest descent algorithm to minimize `PIRobjFctn`*.*

Exercise 6.5
Consider a registration of the images shown in Figure 8.1 using SSD and rigid transformation. Discuss the solutions and regularization.



Chapter 7
Distance Measures

Chapter 5 discussed feature-based distance measure and in Chapter 6 the sum of squared differences (SSD) as a prototype for an intensity-based measure was introduced. A disadvantage of the latter measure is that it assumes a correspondence of gray values of corresponding points. In this chapter, more powerful intensity-based distance measures are explored. All distance measures are considered as functionals in $\mathcal{T}$ and $\mathcal{R}$ and are phrased as

$$\mathcal{D}[\mathcal{T}, \mathcal{R}] = \int_\Omega \phi(\mathcal{T}(x), \mathcal{R}(x))\ dx. \tag{7.1}$$

Considering $\mathcal{D}[y] := \mathcal{D}[\mathcal{T}[y], \mathcal{R}]$ enables a unified treatment in numerical optimization.

In the first section the SSD is revisited from a more general point of view. In addition to the practical issues discussed in Section 6.2, the derivative is interpreted as a force field pushing the geometry. In the following sections, the normalized cross-correlation, mutual information, and normalized gradient field are discussed from a distance measure perspective. From a theoretical point of view, mutual information is probably the most general distance measure. As a result of this generality, mutual information can indicate a variety of possible matches, which might be undetectable for other measures. Therefore, one may argue about the necessity of less general measures. However, from a more global perspective on registration, ill-posedness is the ultimate challenge. Therefore an ideal distance measure should indicate only the desired match and should thus be as specific as possible. In this respect, mutual information is somehow the worst distance measure. The truth is probably in between. A coarse to fine technique concerning the choice of a distance measure can provide an interesting option.

7.1 Sum of Squared Differences

In Section 6.2 (FAIR 9 (p.71)), SSD has been introduced as

$$\mathcal{D}^{\mathrm{SSD}}[\mathcal{T}, \mathcal{R}] = \frac{1}{2} \int_\Omega (\mathcal{T}(x) - \mathcal{R}(x))^2\ dx.$$

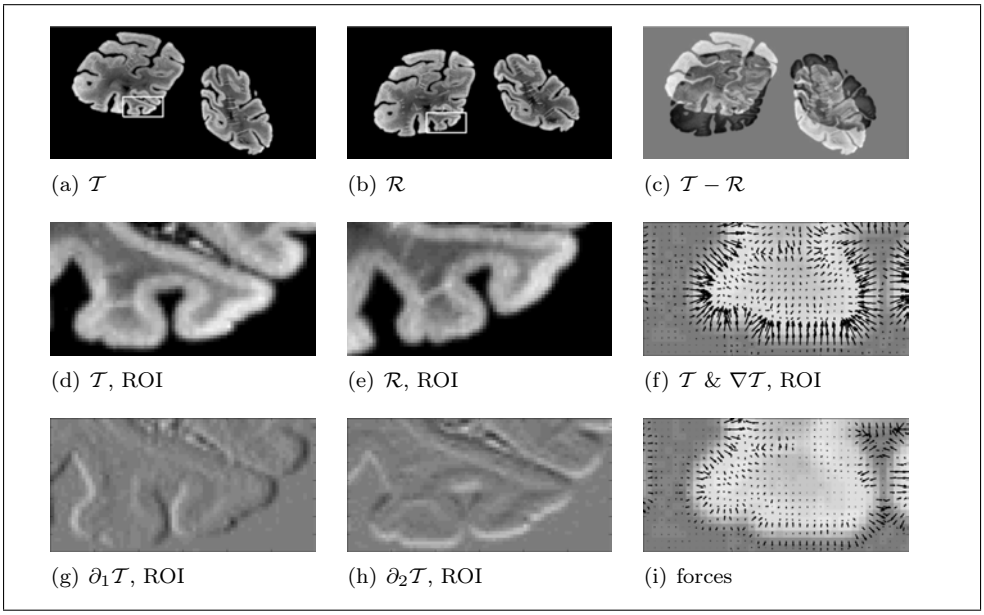


Figure 7.1: *Forces of the SSD:* (a) *template* $\mathcal{T}$ *with a region of interest (ROI);* (b) *reference* $\mathcal{R}$ *with ROI;* (c) *difference* $\mathcal{T} - \mathcal{R}$*; ROIs of* (d) $\mathcal{T}$, (e) $\mathcal{R}$, (f) $\nabla\mathcal{T}$, (g) $\partial_1\mathcal{T}$, (h) $\partial_2\mathcal{T}$*; and the* (i) *force field.*

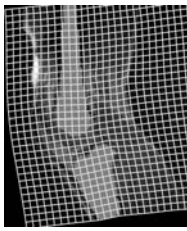
Recalling $\mathcal{J}[y] = \mathcal{D}[\mathcal{T}[y], \mathcal{R}] = \psi[r[y]]$, where $\psi[r] = \frac{1}{2}\|r\|^2_{L_2(\Omega)}$ and $r[y] = \mathcal{T}[y] - R$, the Gâteaux derivative, discretization, and the so-called force fields are discussed.

7.1.1 SSD and Forces

Since image registration is considered an optimization problem, derivatives play a prominent role. Given a suitable perturbation v of y, the Gâteaux derivative of the functional $\mathcal{J}$ is

$$\left.\begin{aligned} d_v\mathcal{J}[y] &= \lim_{\tau\to 0}\frac{1}{\tau}(\mathcal{J}[y+\tau v] - \mathcal{J}[y]) \\ &= \lim_{\tau\to 0}\frac{1}{2\tau}\int_\Omega (\mathcal{T}[y+\tau v] - \mathcal{R})^2 - (\mathcal{T}[y] - \mathcal{R})^2 \, dx \\ &= \lim_{\tau\to 0}\frac{1}{2\tau}\int_\Omega 2(\mathcal{T}[y] - \mathcal{R})(\tau\nabla\mathcal{T}[y]v) + \mathcal{O}(\tau^2) \, dx \\ &= \int_\Omega \underbrace{(\mathcal{T}(y(x)) - \mathcal{R}(x))\nabla\mathcal{T}(y(x))}_{=:f(x,y)} v(x) \, dx, \end{aligned}\right\} \tag{7.2}$$

where f is also called a *force field*; see, e.g., [162]. The force field can be viewed as a steepest descent direction for the minimization of $\mathcal{J}$ in a continuous space. It provides information on how to push points in order to reduce the distance; see Figure 7.1.



7.1.2 Discretized SSD

In the discrete analogue (FAIR 10 (p.72)), the integral is approximated by a midpoint quadrature rule. Let $x_j = (x_j^1, \dots, x_j^d)$, $j = 1, \dots, n$, denote the cell-centered knots of the quadrature rule, and let y_j be an approximation to $y(x_j)$,

$$\begin{aligned} \texttt{xc} &= [x_1^1; \dots; x_n^1; x_1^2; \dots; x_n^2; \quad \dots \quad ; x_1^d; \dots; x_n^d], \\ \texttt{yc} &= [y_1^1; \dots; y_n^1; y_1^2; \dots; y_n^2; \quad \dots \quad ; y_1^d; \dots; y_n^d]. \end{aligned}$$

Hence, with `hd` the cell width, the discrete SSD is given by

$$J(\texttt{yc}) = D^{\mathrm{SSD},h}(T(\texttt{yc}), R(\texttt{xc})) = \frac{1}{2} \cdot \texttt{hd} \cdot \|T(\texttt{yc}) - R(\texttt{xc})\|^2.$$

Particularly in a multilevel setting, a proper discretization of the integral and the scaling `hd` are important. Note that in contrast to the discussion in Chapter 6, it is no longer assumed that the transformation y is parameterizable. However, the derivative of the discretized functional with respect to `yc` can still be computed,

$$dJ(\texttt{yc}) = \texttt{hd}\ \mathrm{d}T(\texttt{yc}) \cdot (T(\texttt{yc}) - R(\texttt{xc})),$$

which is the discrete analogue of the force field f in (7.2). Figure 7.2 shows the discrete SSD as a function versus rotations around the center of the domain and versus translations.

7.2 Cross-Correlation

A problem of the SSD distance measure is that it directly compares $\mathcal{T}(y(x))$ with $\mathcal{R}(x)$, implicitly assuming that gray values of corresponding points x and $y(x)$ do also correspond. Ideally, it is assumed that $\mathcal{T}[y] = \mathcal{R}$. This can be a drawback for multimodal images such as shown in Figure 7.3; for results, see Figure 7.4. The extension presented in this section aims for only a linear dependence of $\mathcal{T}[y]$ and $\mathcal{R}$, i.e., with two scalars $\lambda, \mu \in \mathbb{R}$, only $\lambda\mathcal{T}[y] = \mu\mathcal{R}$ is assumed.

7.2.1 Continuous Normalized Cross-Correlation

The cross-correlation has its origin in a comparison of the reference image and a translated version of the template image. Expanding

$$(\mathcal{T}(x) - \mathcal{R}(x))^2 = \mathcal{T}^2(x) - 2\mathcal{T}(x)\mathcal{R}(x) + \mathcal{R}^2(x)$$

and ignoring the squares on the right-hand side, the cross-correlation is given by

$$\langle \mathcal{T}, \mathcal{R} \rangle = \int_{\mathbb{R}^d} \mathcal{T}(x)\ \mathcal{R}(x)\ dx.$$

In combination with a rigid transformation, this measure could be used directly. However, if the transformation also enables a scaling of the template image,

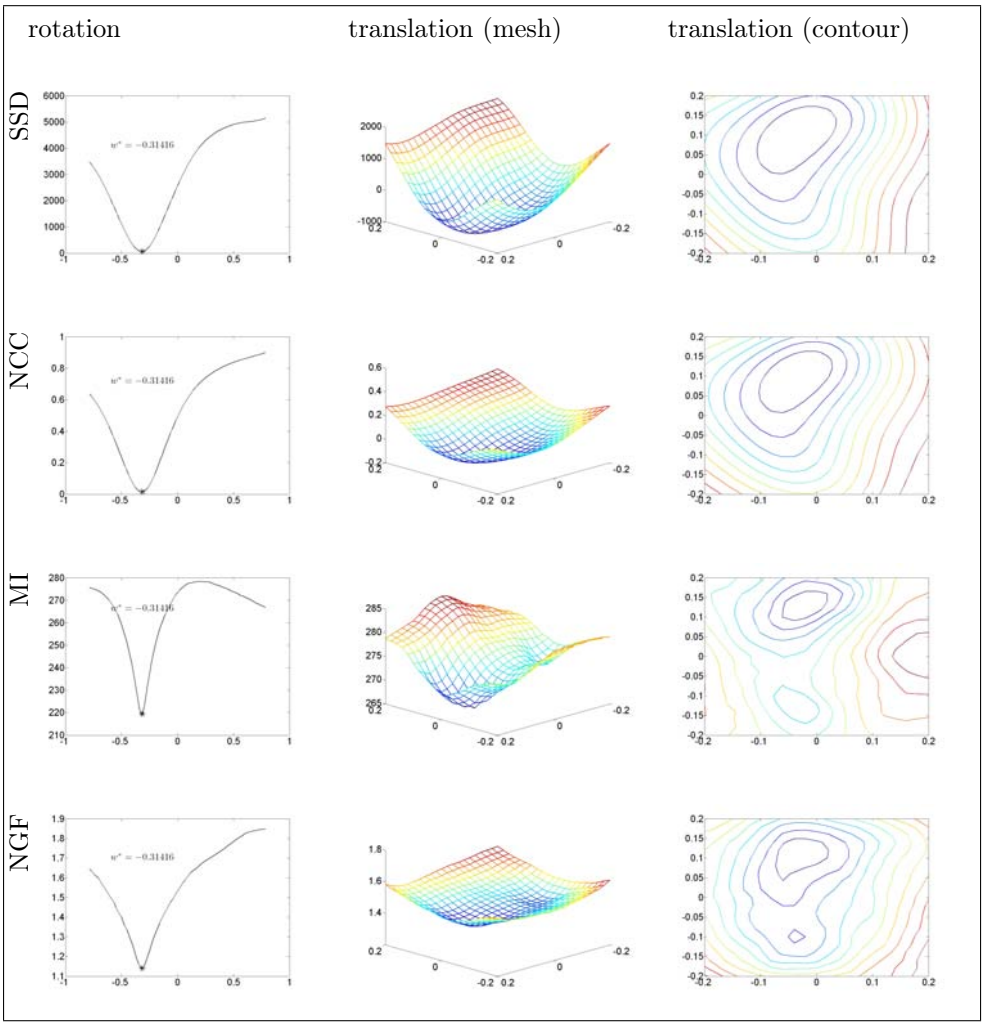


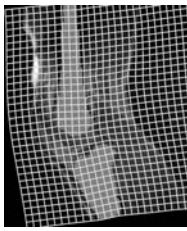
Figure 7.2: *Distance measures for the monomodal images shown in Figure 7.1. Left column: distance versus rotations around the domain center, middle/right columns: mesh/contour plots of distance versus translation.*

the situation becomes more complex. A standard remedy is to maximize the normalized cross-correlation (NCC),

$$\mathrm{NCC}[\mathcal{T},\mathcal{R}] = \frac{\langle \mathcal{T},\mathcal{R}\rangle}{\|\mathcal{T}\|\ \|\mathcal{R}\|}, \tag{7.3}$$

where $\|\mathcal{T}\| = \sqrt{\langle \mathcal{T},\mathcal{T}\rangle}$ and it is assumed that $\mathcal{T}$ and $\mathcal{R}$ are not completely zero. FAIR 13 presents an even more relaxed version which is suitable for minimization.

The square in formula (7.4) focuses on the linear dependency of $\mathcal{T}$ and $\mathcal{R}$. Thus, both choices $\mathcal{T} = \pm\mathcal{R}$ are equally fine. The minus sign converts a maximization problem into the standard minimization problem, and the 1 has been added only to ensure that $0 \le \mathrm{NCC}[\mathcal{T},\mathcal{R}] \le 1$. An implementation is provided in the next subsection.



FAIR 13: Normalized Cross-Correlation (NCC) Distance Measure

Given $\mathcal{T}$ and $\mathcal{R}$, the NCC distance measure is defined by

$$\mathcal{D}^{\mathrm{NCC}}[\mathcal{T},\mathcal{R}] = 1 - \mathrm{NCC}[\mathcal{T},\mathcal{R}]^2 = 1 - \frac{\langle \mathcal{T},\mathcal{R}\rangle^2}{\|\mathcal{T}\|^2\ \|\mathcal{R}\|^2}. \tag{7.4}$$

7.2.2 Discretized Normalized Cross-Correlation

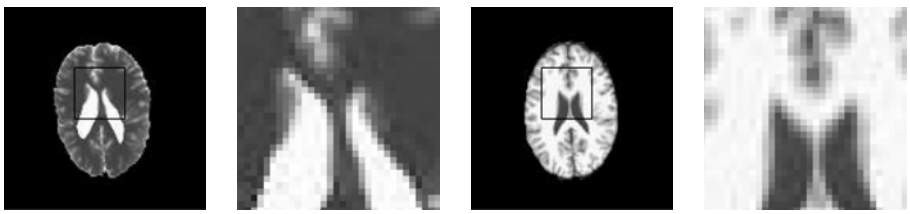
Using the framework presented in Chapter 6, the discretization of the NCC is straightforward. Let $\mathtt{Rc} = R(\mathtt{xc})$ be the discretized reference and let $\mathtt{Tc} = T(\mathtt{yc})$ be the discretized transformed image; (7.3) translates into

$$\mathrm{NCC}^h(\mathtt{Tc},\mathtt{Rc}) = \frac{\langle \mathtt{Tc},\mathtt{Rc}\rangle}{\|\mathtt{Tc}\|\ \|\mathtt{Rc}\|}, \quad \text{with} \quad \|\mathtt{Tc}\| = \sqrt{\mathtt{Tc}^\top \mathtt{Tc}} \quad \text{and} \quad \|\mathtt{Rc}\| = \sqrt{\mathtt{Rc}^\top \mathtt{Rc}}. \tag{7.5}$$

Note that the factor $\mathtt{hd}$ in the integration shows up in the numerator and denominator and hence cancels out. The algorithm is summarized in NCC. The NCC can be rephrased as a composition of $\mathtt{Tc}$ and ψ:

$$D^{\mathrm{NCC}}(\mathtt{Tc}) = \psi(\mathtt{Tc}) = 1 - \frac{(\mathtt{Rc}^\top \mathtt{Tc})^2}{(\mathtt{Rc}^\top \mathtt{Rc})\ (\mathtt{Tc}^\top \mathtt{Tc})}.$$

Figure 7.2 also shows the discrete NCC as a function versus a rotation around the center of the domain and versus translations for images of the same modality. As expected, NCC yields results similar to those of the SSD.



(a) template $\mathcal{T}$ (b) $\mathcal{T}$, ROI (c) reference $\mathcal{R}$ (d) $\mathcal{R}$, ROI

Figure 7.3: *MRI sections of a head. ROI; cf. Example 2.9.*

Another example with images of different modalities is shown in Figure 7.3. Two 2D sections of T1 and T2 weighted MRIs of a human head serve as a test case; cf. Example 2.9. As a matter of fact, the differences between SSD and NCC appear to be very small. In particular, SSD gives unexpectedly good results.

7.3 Mutual Information

Probably the most commonly used distance measure for registration is mutual information, which has been introduced independently by Collignon et al. [90] and

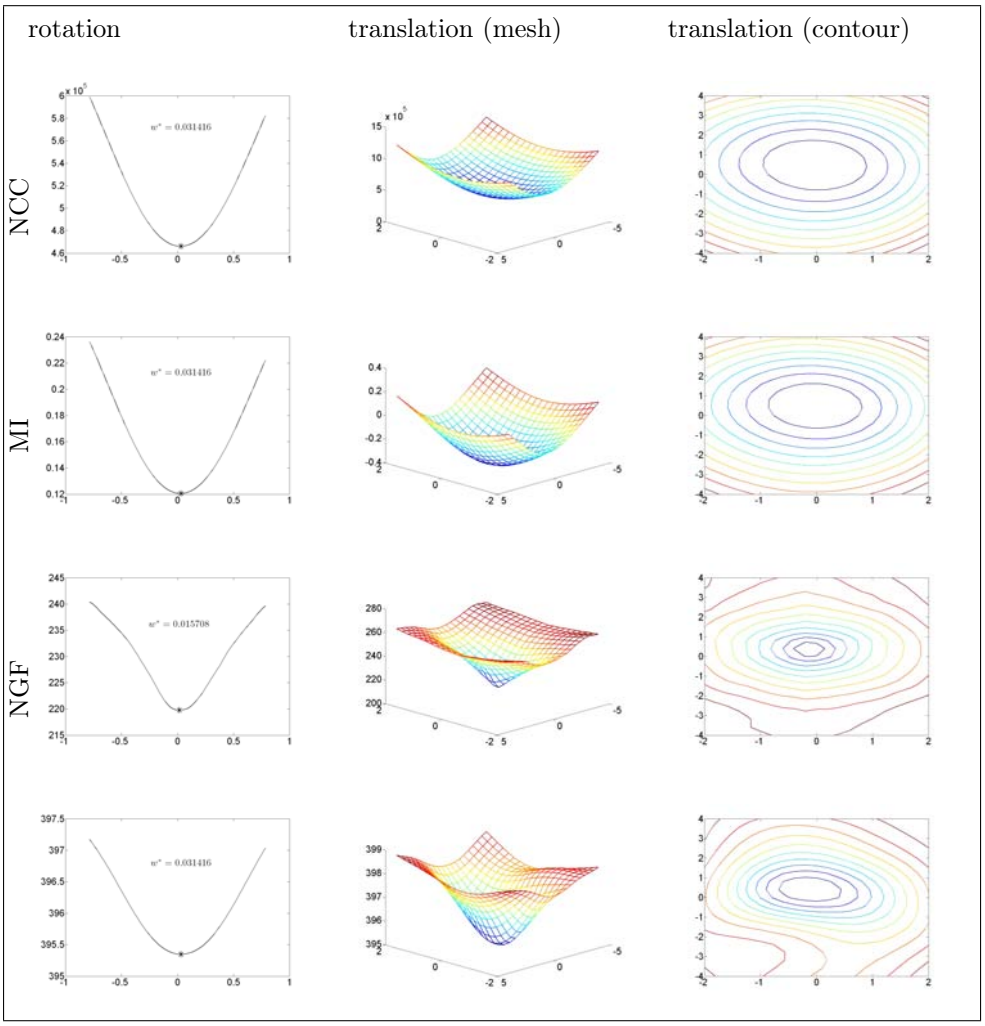


Figure 7.4: *Distance measures for T1-T2 MRIs; see Figure 7.3. Left column: distance versus rotations around the domain center; middle/right columns: mesh/contour plots of distance versus translation.*

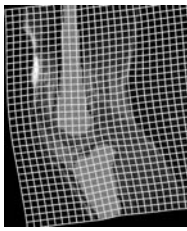
Viola [202]. The concept originated in information theory. Given two sequences $[T_j]$ and $[R_j]$, $j = 1, \ldots, n$, the underlying idea is to measure the mutual information of T and R, i.e., a normalized entropy of the joint density.

Example 7.1 (Histogram)
Let two sequences $[T_j]$ and $[R_j]$ be as follows:

T	△	■	■	△	■	△	★	★	★
R	A	C	C	A	C	A	G	G	G

The first step is to produce a histogram, summarizing the coincidences of all possible n pairs in the sequence,

$$\rho^{\text{hist}}(t, r) = \#\{(T_i, R_j) \mid T_i = t \wedge R_j = r\}/n.$$



In this example, ρ^{hist} takes the following values.

ρ^{hist}	A	C	G
△	1/3	0	0
■	0	1/3	0
★	0	0	1/3

If ρ^{hist} has at most one nonzero entry per row and column, say at position (i, j_i), a one-to-one correspondence between T_i and R_{j_i} can be assumed. In this example, this is the case and one may conclude that $A \triangleq \triangle$, $C \triangleq \blacksquare$, and $G \triangleq \bigstar$. Thus, knowing the sequence T is as good as knowing the sequence R.

The entropy H is a common measure for quantifying the "sharpness" of a histogram ρ^{hist},

$$H[\rho^{\text{hist}}] = -\sum_{t,r} \rho^{\text{hist}}(t,r) \log \rho^{\text{hist}}(t,r) = -\frac{1}{n} \sum_{i,j} \log \rho^{\text{hist}}(T_i, R_j). \tag{7.6}$$

Example 7.2 (Mutual Information)
Suppose that the sequences are completely arbitrary and $\rho^{\text{hist}}(t,r) = 1/n$. Hence $H[\rho^{\text{hist}}] = \log n$. On the other hand, if ρ^{hist} has only one nonzero entry per row and column (say $\rho^{\text{hist}}(T_i, R_{j_i}) = 1/\sqrt{n}$ for all i) as in the above example, then $H[\rho^{\text{hist}}] = 0.5 \log n$, which basically claims that the latter situation is much more organized than the first one. For the extreme case $\rho^{\text{hist}}(T_1, R_{j_1}) = 1$ and zero elsewhere, the entropy takes its minimal value: $H[\rho^{\text{hist}}] = 0$.

The computation of the mutual information involves an additional normalization step and is defined by

$$\hat{\text{MI}}[\rho^{\text{hist}}] = H[\rho_T^{\text{hist}}] + H[\rho_R^{\text{hist}}] - H[\rho^{\text{hist}}],$$

where $\rho_T^{\text{hist}}(t) = \sum_r \rho^{\text{hist}}(t,r)$ and $\rho_R^{\text{hist}}(r) = \sum_t \rho^{\text{hist}}(t,r)$ are the marginal densities. For the case where all occurrences are equally likely, $\hat{\text{MI}}[\rho^{\text{hist}}] = 0$, whereas for the one-to-one case, $\hat{\text{MI}}[\rho^{\text{hist}}] = 0.5 \log n$.

Although this measure seems to be quite intuitive for discrete sequences $[T_j]$, $[R_j]$, the extension to the continuous framework is not straightforward; see, e.g., [172, 171, 170, 173, 135, 126]. All extensions to the continuous case are based on estimations for the generally unknown joined gray value distribution ρ of the continuous images $\mathcal{T}$ and $\mathcal{R}$.

7.3.1 Estimating the Joint Density, Principles

The key quantity is the joint density ρ. Unfortunately, it is generally unknown and can only be estimated. In image registration, two approaches are popular. Both approaches are based on a discretization T of the continuous function $\mathcal{T}$.

The first approach is based on histograms and the second is based on Parzen-window estimators. Histogram-based estimators are commonly used in registration and are therefore briefly described in the next section. However, histogram-based approaches are known to be inferior to Parzen-window-based estimators [185] and are thus not considered in FAIR.

Histogram-Based Density Estimators

A seemingly simple approach is to basically use a histogram with certain bin sizes; see Example 7.3.

Example 7.3 (Density Estimation Based on Histograms)
The function $\mathcal{T} : [0, 2\pi] \to \mathbb{R}$ with $\mathcal{T}(x) = 0.5(\sin(x) + 1)$ is explored. The discretization is based on n samples, $n = 10, 100, 1000$. Figure 7.5 shows the histogram for $g =$ `numberBins` bins of equal width.

This file is `E7_Histogram1D.m`

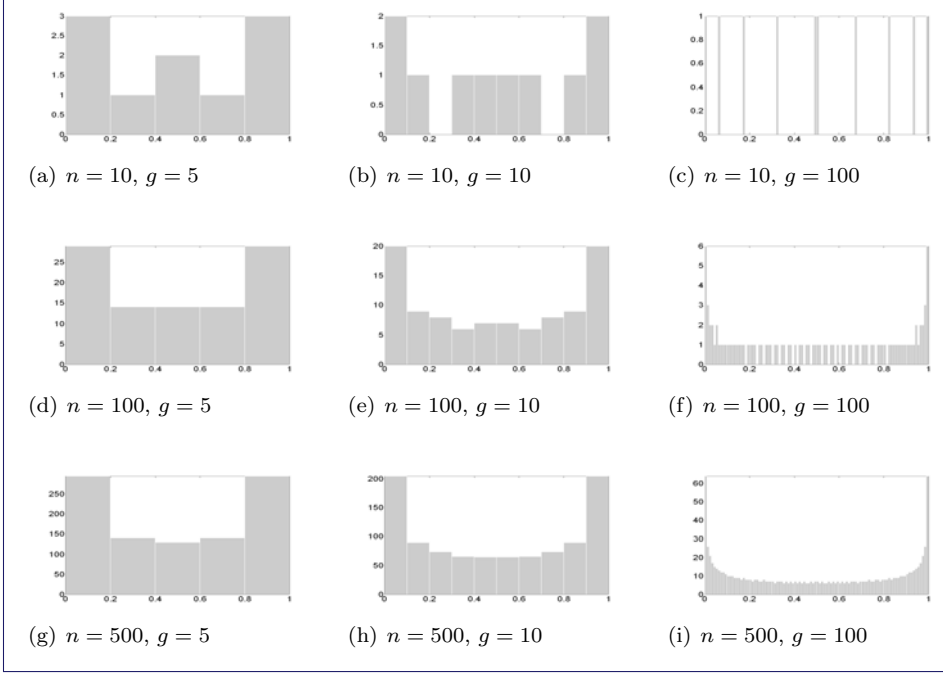
```
%%%
n = 1000;
x = linspace(0,2*pi,n); T = 0.5*(sin(x)+1); % discretized function
minT = 0; maxT = 1;                         % bounds for the bins
numberBins = 5;                             % number of bins
binWidth   = (maxT-minT)/numberBins;        % bin width
bins       = 0:binWidth:maxT;               % the bins
binsExt    = [-inf,bins(2:end-1),inf];      % don't miss anything

rhoHat = histc(T,binsExt); % compute the histogram and plot it
bar(bins+binWidth/2,rhoHat,0.99,'edgecolor','w','facecolor',0.8*[1,1,1]);
axis([minT,maxT,0,inf])
```

This approach has several drawbacks: The histogram-based estimator has inferior approximation properties (see, e.g., [185]); the particular estimator strongly depends on several parameter (such as number of bins and the bin width); and the histogram is based on rounding and thus leads to a nondifferential function which is not suitable for optimization. Therefore, histogram-based density estimators are not supported by FAIR.

Parzen-Window-Based Density Estimators

The basic idea of a Parzen-window estimator is to work with a smooth kernel function which basically spreads out sampled data. The Parzen-window estimator has better approximation properties and can give a smooth estimator which is much better suited for optimization purposes. From a theoretical point of view, the particular kernel function is not very important. However, for registration purposes, a smooth, compactly supported function is preferred; see Example 7.4 and Figure 7.6. In FAIR, a cubic spline serves as a kernel.

Example 7.4 (Parzen-Window Kernel Function)
The basic ingredient is a bell-shaped function, smooth, compactly supported, and of integral one. In FAIR, a spline function is used; see Section 3.4.1 and in

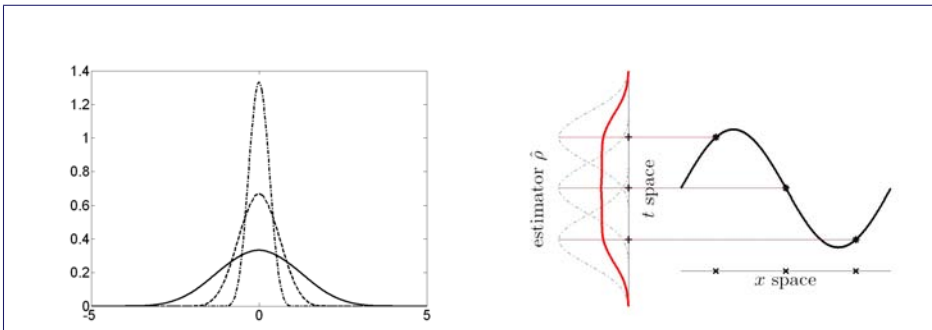


(a) $n = 10,\ g = 5$ (b) $n = 10,\ g = 10$ (c) $n = 10,\ g = 100$

(d) $n = 100,\ g = 5$ (e) $n = 100,\ g = 10$ (f) $n = 100,\ g = 100$

(g) $n = 500,\ g = 5$ (h) $n = 500,\ g = 10$ (i) $n = 500,\ g = 100$

Figure 7.5: *Histograms of discretizations from n samples of a univariate function based on g equally spaced bins, $n = 10, 100, 1000$, $g = 5, 10, 100$.*



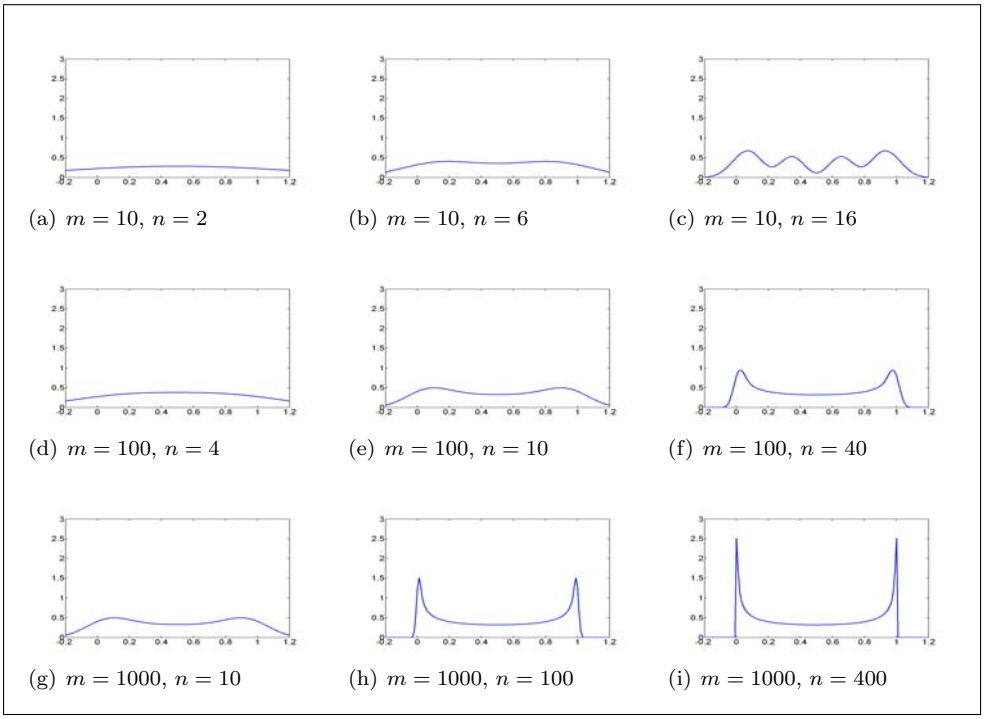
(a) $k(\cdot, \sigma)$ for $\sigma = 2, 1, 0.5$ (b) Parzen-window based density estimator

Figure 7.6: (a) *Spline-based Parzen-window kernel $k(\cdot, \sigma)$ for $\sigma = 2, 1, 0.5$;* (b) *Parzen-window estimator based on three samples.*

particular (3.8) and Figure 7.6. The width of the spline-based kernel k is controlled by a dilatation factor σ,

$$k(t, \sigma) = k(t/\sigma)/\sigma.$$

Since the integral of the dilated spline is one, dilation also affects the height of the function.



(a) $m = 10$, $n = 2$ (b) $m = 10$, $n = 6$ (c) $m = 10$, $n = 16$

(d) $m = 100$, $n = 4$ (e) $m = 100$, $n = 10$ (f) $m = 100$, $n = 40$

(g) $m = 1000$, $n = 10$ (h) $m = 1000$, $n = 100$ (i) $m = 1000$, $n = 400$

Figure 7.7: *Parzen-window density estimators based on different samples of $\mathcal{T}$ (top, middle, bottom for $n = 10, 100, 1000$) with different choices for the window width $\sigma \sim 1/n$; see Example 7.5.*

The idea of a Parzen-window estimator is illustrated in Figure 7.6. Copies of the kernel are placed at the positions $t_j = \mathcal{T}(\texttt{xc}_j)$ in the gray value space, $j = 1, 2, 3$. The estimator is obtained by summing these shifted copies and dividing the sum by the number of copies:

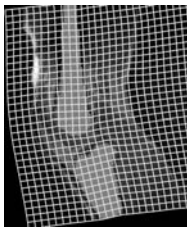
$$\rho(t;\sigma) = \rho(t;\mathcal{T},\texttt{xc},\sigma) = \frac{1}{m}\sum_{j=1}^{m} k(t - t_j, \sigma). \tag{7.7}$$

The Parzen-window estimator depends on the choice of σ. Here, we assume the gray values to be in an interval $[t_0, t_n]$, and we set $\sigma = (t_n - t_0)/n$. The window width σ can be considered as an analogue of the bin width in the histogram approach.

Example 7.5 (Parzen-Window Estimation)
Parzen-window estimators for the density of the modified sine function introduced in Example 7.3 and various choices of m and n are presented. Here, xc is a cell-centered grid of m points, $t_0 = \min\{\mathcal{T}(t), t \in \mathbb{R}\} = 0$, $t_n = \max\{\mathcal{T}(t), t \in \mathbb{R}\} = 1$,

$$\sigma = \frac{1}{n} = \frac{t_n - t_0}{n}, \quad T_j = \mathcal{T}(\texttt{xc}_j), \quad j = 1, \ldots, m.$$

As the results presented in Figure 7.7 indicate, the choices of m and n are crucial. A large window size σ (small n) results in oversmoothing, and a very small window size



($n > m$) results in a spiky density. Even for a relative large sample size ($m = 1000$) different results are obtained.

The problem is that the derivative of the modified sin function vanishes, introducing singularities in the density: the spikes at 0 and 1 get higher with increasing m. Note that similar problems arise for piecewise constant functions, such as images. For a more detailed analysis, see [126].

Although this estimator is smooth, the previous example indicates that it does depend on the sample size n and the window width σ. For image registration, where a multilevel approach is inevitable to avoid suboptimal results, this is bad news. For a fixed σ, the estimator, and hence the objective function, becomes rougher the smaller the sample size is drawn. A remedy is to adapt the window width to the sample size automatically. The leaving-one-out or Generalized Cross-Validation are state-of-the-art techniques for an automatic adaption. However, changing σ while moving from one discretization to another also changes the objective function. Moreover, the optimal value of σ for the continuous case is generally unknown.

7.3.2 Estimating the Joint Density of Two Images

The above Parzen-window estimator can be extended to any dimension. For two images $\mathcal{T}$ and $\mathcal{R}$, let $\mathtt{xc}$ denote an m point discretization of the underlying domain Ω.

FAIR 14: Parzen-Window Estimator

In analogy to (7.7), a 2D Parzen-window estimator for the joint gray value distribution of $\mathcal{T}$ and $\mathcal{R}$ is given by

$$\rho(t,r) = \rho(t,r;\mathcal{T},\mathcal{R},\mathtt{xc},\sigma) = \frac{1}{m}\sum_{j=1}^{m} k(t-\mathcal{T}(\mathtt{xc}_j),\sigma)k(r-\mathcal{R}(\mathtt{xc}_j),\sigma). \tag{7.8}$$

Of course, different kernel functions could be used as well but this is beyond the scope of this book. Since the computation time using MATLAB is already considerable, FAIR provides a C implementation; see `rhoSplineC`.

7.3.3 Mutual Information

The computation of the mutual information is based on the joint density of the images

$$\rho_{[\mathcal{T},\mathcal{R}]}(t,r) = \rho(t,r;\mathcal{T},\mathcal{R},\sigma,\mathtt{xc});$$

cf. FAIR 14 (this page). The marginal densities $\rho_{[\mathcal{T}]}$ and $\rho_{[\mathcal{R}]}$ are

$$\rho_{[\mathcal{T}]}(t) = \int_{\mathbb{R}} \rho_{[\mathcal{T},\mathcal{R}]}(t,r)\,dr \quad \text{and} \quad \rho_{[\mathcal{R}]}(r) = \int_{\mathbb{R}} \rho_{[\mathcal{T},\mathcal{R}]}(t,r)\,dt.$$

FAIR 15: Mutual Information (MI) Distance Measure

Given $\mathcal{T}$ and $\mathcal{R}$, a discretized MI distance measure is given by

$$\mathrm{MI}[\mathcal{T},\mathcal{R}] = \int_{\mathbb{R}} \rho_{[\mathcal{T}]} \log \rho_{[\mathcal{T}]}\, dt + \int_{\mathbb{R}} \rho_{[\mathcal{R}]} \log \rho_{[\mathcal{R}]}\, dr - \int_{\mathbb{R}^2} \rho_{[\mathcal{T},\mathcal{R}]} \log \rho_{[\mathcal{T},\mathcal{R}]}\, d(t,r). \quad (7.9)$$

7.3.4 Discretizing Mutual Information

The main task is the computation of the integral

$$I = \int_{\mathbb{R}^2} \rho_{[\mathcal{T},\mathcal{R}]} \log \rho_{[\mathcal{T},\mathcal{R}]}\; d(t,r).$$

It is assumed that the values of the functions $\mathcal{T}$ and $\mathcal{R}$ are within a known range $[t_0, t_n]$ and $[r_0, r_n]$, respectively, where typical values are $t_0 = r_0 = 0$ and $t_n = r_n = 255$. The integrals can thus be approximated by a midpoint quadrature rule using an n_t-by-n_r grid. Let $\eta_t = (t_n - t_0)/n_t$, $\eta_r = (r_n - r_0)/n_r$,

$$t_i = t_0 + (i - 0.5)\eta_t, \quad r_j = r_0 + (j - 0.5)\eta_r, \quad i = 1, \ldots, n_t, \quad j = 1, \ldots, n_r.$$

Adding a small tolerance to the argument of the logarithm prevents extra considerations for "$0 \log 0$." Thus, with $\rho_i = \rho_{i_1,i_2} = \rho_{[\mathcal{T},\mathcal{R}]}(t_{i_1}, r_{i_2})$,

$$I = \eta_t \eta_r\; \rho^\top \log(\rho + \epsilon) = \eta_t \eta_r \sum_{i=1}^{n_t} \sum_{j=1}^{n_r} \rho_{i,j} \log\left(\rho_{i,j} + \epsilon\right).$$

Reformatting ρ to be an n_t-by-n_t array, the discretized marginal densities can be computed as

$$\rho_{\mathcal{T}}(t_k) = \eta_r \sum_{i=1}^{n_r} \rho_{k,i} \quad \text{and} \quad \rho_{\mathcal{R}}(r_k) = \eta_t \sum_{i=1}^{n_t} \rho_{i,k},$$

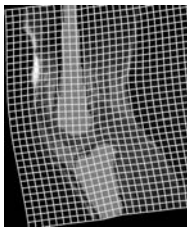
or, using the two matrices S_T and S_R, as $\rho_{\mathcal{T}} = S_T \rho$ and $\rho_{\mathcal{R}} = S_R \rho$, where

$$S_T = \underbrace{[1, \ldots, 1]}_{n_r} \otimes I_{n_t}, \quad S_R = I_{n_r} \otimes \underbrace{[1, \ldots, 1]}_{n_t}.$$

Although building and storing these matrices can be avoided for the computation of MI, they provide a convenient tool for the computation of the derivative.

The MI for the rotation and translation of the test cases are presented in Figures 7.2 and 7.4.

Remark 7.1
From a multiscale point of view, a major problem arises from the fact that MI is defined in the gray value space rather than in the geometrical space. Improving the spacial resolution does not necessarily improve the resolution in the gray value space and vice versa. A proper match of the discretization of the distance measure and the discretization of the regularizer is, however, a central question in multilevel approaches.



7.4 Normalized Gradient Fields

A compromise between the more restricted SSD (aiming for correspondences of gray values) and the very general (and hence highly nonconvex) MI is based on normalized image intensity gradients [123].

This distance measure is based on the observation that the contents of the image $\mathcal{T}$ is also displayed by intensity changes and therefore indicated by the image gradient $\nabla\mathcal{T}$. The image gradient thus plays a dominant role in registration schemes. This can also be seen by looking at the steepest descent direction of a generic distance measure $\mathcal{D}[y] = \mathcal{D}[\mathcal{T}[y], \mathcal{R}]$. The chain rule yields

$$d_y\mathcal{D} = d_{\mathcal{T}}\mathcal{D}\; d_y\mathcal{T},$$

indicating that $\nabla\mathcal{T} = (d_y\mathcal{T})^\top$ is indeed a very important ingredient. In the SSD example, the force is a product of the difference image $\mathcal{T} - \mathcal{R}$ (which might be viewed as a switch, telling in which direction to go) and the gradient $\nabla\mathcal{T}$. For regions where $\mathcal{T}$ is constant, the forces are zero and the registration is completely governed by the regularization (which is discussed in the next chapter). Since the gradient is orthogonal to the level set $L(c) = \{x : \mathcal{T}(x) = c\}$, an alternative interpretation or intuition can be gained from the alignment of level sets.

7.4.1 Continuous Normalized Gradient Fields

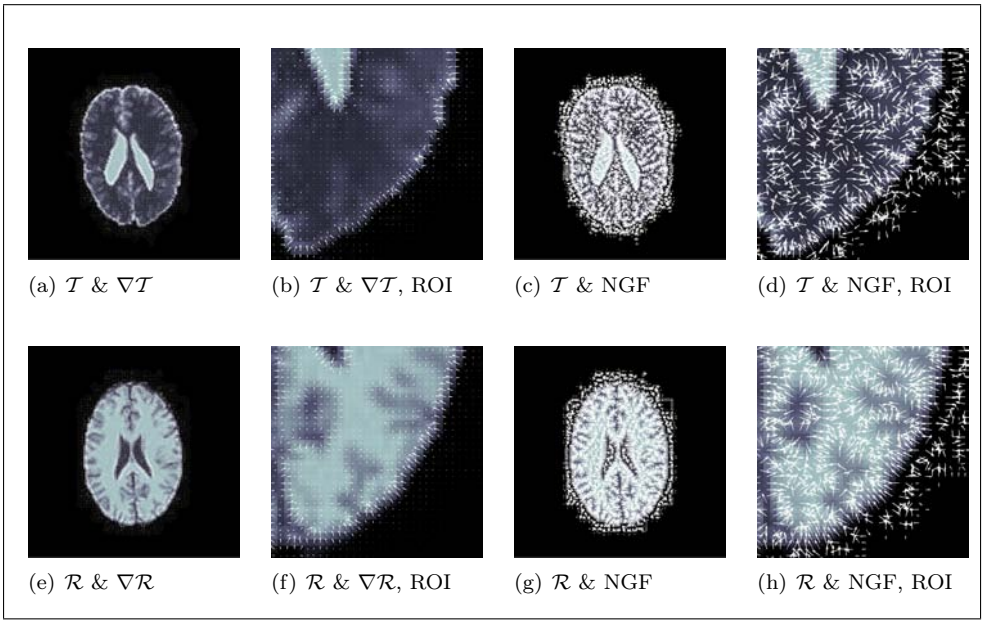
The assumption is that even for images of different modalities, intensity changes appear at corresponding positions. Since intensity changes are indicated by the image gradient, this is a quantity to look at. However, the gradient also measures the strength of the change which is unwanted information for multimodal registration. Therefore, the gradient $\nabla\mathcal{T}$ is replaced by $\nabla\mathcal{T}/|\nabla\mathcal{T}|$ (assuming $\nabla\mathcal{T} \neq 0$). This normalization removes unwanted information and focuses on locations of changes rather than on the strength. Figure 7.8 shows a region of interest in the template image (a) and a corresponding region in the reference image (b) overlaid by the image gradients and the normalized gradient field (c),(d). The normalized gradient field is defined by

$$\mathrm{n}[\mathcal{T}] = \mathrm{n}[\mathcal{T}, \eta] = \frac{\nabla\mathcal{T}}{\sqrt{|\nabla\mathcal{T}|^2 + \eta^2}}, \tag{7.10}$$

where η is an important edge parameter. The edge parameter determines what is to be considered as an edge ($|\nabla T| > \eta$) and what is considered to be within the noise level ($|\nabla T| < \eta$). Without this parameter, image noise blown up by the normalization step would corrupt the valuable information given by the normalized gradient field; see [123] for an extended discussion.

Figure 7.8 shows the gradient and normalized gradient fields for the two MRIs introduced in Figure 7.3. It is obvious that the normalized gradient field indicates intensity change and directions but does not indicate the strength of the change.

The final step is to measure the alignment of the two gradient fields for $\mathcal{T}$ and $\mathcal{R}$. Since there should be no bias in the direction of the intensity change, the ideal situation is $t = \pm r$, where $t := \mathrm{n}[\mathcal{T}](x)$ and $r = \mathrm{n}[\mathcal{R}](x)$. Therefore, the area



(a) $\mathcal{T}$ & $\nabla\mathcal{T}$ (b) $\mathcal{T}$ & $\nabla\mathcal{T}$, ROI (c) $\mathcal{T}$ & NGF (d) $\mathcal{T}$ & NGF, ROI

(e) $\mathcal{R}$ & $\nabla\mathcal{R}$ (f) $\mathcal{R}$ & $\nabla\mathcal{R}$, ROI (g) $\mathcal{R}$ & NGF (h) $\mathcal{R}$ & NGF, ROI

Figure 7.8: *Normalized Gradient Fields (NGF) for T1/T2 weighted MRIs; cf. Example 2.9.*

spanned by the two vectors t and r could be used as a measure for linear dependency. However, minimizing this area also indicates a perfect match if t or r is zero, which is generally unwanted. Therefore, rather than minimizing the area, we maximize the linear dependency of t and r. Normalization steps such as introducing a minus sign (transforming a maximization into a minimization problem) and adding a 1 (giving nonnegative values) lead to

$$d(t, r) = 1 - (t^\top r)^2.$$

Note that this measure gives minimal value zero for $t = \pm r$ and gives maximal value one if t is orthogonal to r, including $t = 0$ or $r = 0$.

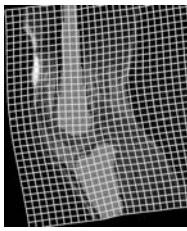
FAIR 16: Normalized Gradient Field (NGF) Distance Measure

Given $\mathcal{T}$ and $\mathcal{R}$, the NGF distance measure is defined by

$$\mathcal{D}^{\mathrm{NGF}}[\mathcal{T}, \mathcal{R}] = \mathrm{NGF}[\mathcal{T}, \mathcal{R}] = \int_\Omega 1 - (\mathrm{n}[\mathcal{T}(x)]^\top \mathrm{n}[\mathcal{R}(x)])^2 \, dx. \tag{7.11}$$

7.4.2 Discretized Normalized Gradient Fields

The NGF distance measure is basically an L_2-norm of a residual r, where r measures the alignment of the normalized gradients of two given images at a position x. The discretization steps are described for $\mathcal{T}$, the discretization of $\mathcal{R}$ is along the same



lines. Let $\mathtt{xc}$ denote a cell-centered grid and $\mathtt{yc} = y(\mathtt{xc})$. The gradients $\nabla\mathcal{T}$ are approximated by finite differences,

$$\partial_j \mathcal{T}[y](\mathtt{xc}) \approx \partial_j^h T(\mathtt{yc}),$$

where with the Kronecker product $\otimes$,

$$\left.\begin{aligned} \partial_1^h &= I_{m^3} \otimes I_{m^2} \otimes D_1, \\ \partial_2^h &= I_{m^3} \otimes D_2 \otimes I_{m^1}, \\ \partial_3^h &= D_3 \otimes I_{m^2} \otimes I_{m^1}, \\ D_j &= \frac{1}{2h^j} \begin{bmatrix} -1 & 1 & & & \\ -1 & 0 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 0 & 1 \\ & & & -1 & 1 \end{bmatrix} \in \mathbb{R}^{m^j, m^j}. \end{aligned}\right\} \tag{7.12}$$

Since $\mathcal{T}[y]$ changes with y, the derivatives have to be recomputed. An alternative way is to use the chain rule. The derivatives of $\mathcal{T}$ are then to be multiplied with the Jacobian ∇y, which is the drawback of this alternative.

Once the gradients have been computed, all computations are pointwise. To this end, let $\mathtt{Tc} = \mathcal{T}(\mathtt{yc}) \in \mathbb{R}^n$, $\mathtt{Rc} = \mathcal{R}(\mathtt{xc}) \in \mathbb{R}^n$,

$$\mathtt{gradT}_i = [(\partial_1^h \mathtt{Tc})_i, (\partial_2^h \mathtt{Tc})_i, (\partial_3^h \mathtt{Tc})_i]^\top, \ \mathtt{gradR}_i = [(\partial_1^h \mathtt{Rc})_i, (\partial_2^h \mathtt{Rc})_i, (\partial_3^h \mathtt{Rc})_i]^\top. \tag{7.13}$$

Hence, with

$$\mathrm{n}[\mathcal{T}[y](x_i)]^\top \mathrm{n}[\mathcal{R}(x_i)] \approx \mathtt{rc}_i := \frac{\mathtt{gradT}_i^\top \mathtt{gradR}_i}{\sqrt{||\mathtt{gradT}_i||^2 + \eta^2}\sqrt{||\mathtt{gradR}_i||^2 + \eta^2}},$$

$\mathtt{hd} = h^1 \cdots h^d$, and $\hat{\omega} = \prod_{i=1}^d (\omega^{2i-1} - \omega^{2d})$, the discretized NGF is

$$\mathrm{NGF}(\mathtt{yc}) = \mathtt{hd} \sum_{i=1}^m (1 - \mathtt{rc}_i^2) = \hat{\omega} - \mathtt{hd} \cdot \mathtt{rc}^\top \mathtt{rc}.$$

7.5 Derivatives of Distance Measures

The computation of derivatives for the distance measures has already been discussed for the SSD in Chapter 6. Now we are ready to discuss the general case. The basic idea is to decompose the distance measure into elementary building blocks,

$$D(\mathtt{yc}) = D(T(\mathtt{yc}), \mathtt{Rc}) = \psi(r(\mathtt{yc})),$$

and to approximate the Hessian by first order information, neglecting the second order derivative of the residual r:

$$dD(\mathtt{yc}) = d\psi(r(\mathtt{yc}))\ dr(\mathtt{yc}), \quad d^2 D(\mathtt{yc}) \approx dr(\mathtt{yc})^\top d^2\psi(r(\mathtt{yc}))\ dr(\mathtt{yc}).$$

Example 7.6 (Derivative of Discrete NGF)
Let $\mathtt{Tc} \in \mathbb{R}^n$ with $n = m_1 \cdots m_d$ be the discrete transformed image and $\partial_\ell^h \in \mathbb{R}^{n,n}$ the discrete gradient operators as in (7.12). In `NGF`, the gradients and regularized lengths are computed:

$$\begin{aligned} \mathtt{gradT} &= [\partial_1^h \mathtt{Tc}, \partial_2^h \mathtt{Tc}, \partial_2^h \mathtt{Tc}] \in \mathbb{R}^{n,3}, & \mathtt{lengthGT}_i &= \sqrt{||\mathtt{gradT}_i||^2 + \eta^2}, \\ \mathtt{gradR} &= [\partial_1^h \mathtt{Rc}, \partial_2^h \mathtt{Rc}, \partial_2^h \mathtt{Rc}] \in \mathbb{R}^{n,3}, & \mathtt{lengthGR}_i &= \sqrt{||\mathtt{gradR}_i||^2 + \eta^2}. \end{aligned}$$

For convenience, the residual is written as $r = r^1 r^2$, where

$$r_i^1 = \mathtt{gradT}_i^\top \mathtt{gradR}_i, \quad r_i^2 = 1/(\mathtt{lengthGT}_i \mathtt{lengthGR}_i).$$

A sparse matrix multiplication is used as a compromise between efficient and convenient computation of the Hadamard or pointwise product:

$$r^1 \odot r^2 = [r_i^1 r_i^2]_i = \operatorname{diag}[r^1] r^2.$$

Thus,

$$\begin{aligned} dr^1 &= \operatorname{diag}[\mathtt{gradR}^1]\ \partial_1^h + \operatorname{diag}[\mathtt{gradR}^2]\ \partial_2^h + \operatorname{diag}[\mathtt{gradR}^3]\ \partial_3^h \\ dr^2 &= -\operatorname{diag}[1/((\mathtt{lengthGT}_i)^3 \cdot \mathtt{lengthGR}_i),\ i] \\ &\qquad \cdot(\operatorname{diag}[\mathtt{gradT}^1]\ \partial_1^h + \operatorname{diag}[\mathtt{gradT}^2]\ \partial_2^h + \operatorname{diag}[\mathtt{gradT}^3]\ \partial_3^h), \\ dr &= \operatorname{diag}[r^2]\ dr^1 + \operatorname{diag}[r^1]\ dr^2 \end{aligned}$$

and the derivatives of the NGF are

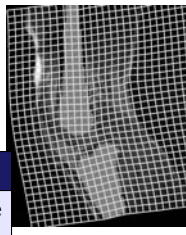
$$dD = -2\ \mathtt{hd}\ r^\top\ dr\ d\mathtt{Tc}, \qquad d^2\psi = 2\ \mathtt{hd}.$$

7.6 Summarizing the Distance Measures

The most commonly used distance measures SSD, NCC, MI, and NGF have been introduced and discussed. FAIR enables a convenient use of different measures including their analytical derivatives. As for the interpolation and transformation modules, a generic distance measure module `distance` is provided. On the basis of a persistent parameter `OPTN`, this function allows a convenient integration of the various measures; see also Section 2.3.7 and `options`.

For the following examples, CT and PET images from the thorax serve as data; cf. Example 2.10. The first example illustrates how to explore different distance measures, and the last example shows how different distance measures can be used within FAIR's MLPIR.

Example 7.7 (Exploring Distance Measures)
In this example, distances of a CT and a rotated PET images are computed; see Table 7.1. Figure 7.9 present some results for different multiscale parameters θ; see Section 3.6. Note that optimizing the smoothed problem is much simpler.



FAIR 17: Distance Measure Toolbox

This function computes an integral-based distance measure by approximating the integral using a midpoint quadrature rule based on a cell-centered grid `xc` of m points for the domain Ω. The distance measure $D(\texttt{Tc}) = \psi(r(\texttt{Tc}))$ is coded as a composition of an outer function ψ and the residual r to enable a Gauss–Newton-type optimization scheme; see Chapter 6. The input and output of the function are summarized in the following table. The function returns its derivative and an approximation to the Hessian given is by $\nabla^2 D \approx H = dr^\top \, d^2\psi \, dr$.

`[Dc,rc,dD,dr,d2psi]= distance(Tc,Rc,omega,m)`	
`Tc` $\in \mathbb{R}^n$	sampled transformed template `Tc` $= T($`yc`$)$
`Rc` $\in \mathbb{R}^n$	sampled reference `Rc` $= R($`xc`$)$
`omega,m`	specify domain and discretization
`Dc` $\in \mathbb{R}$	distance of `Tc` and `Rc`
`rc` $\in \mathbb{R}^p$	residual r, e.g., $r =$ `Tc` $-$ `Rc` for SSD
	or $r = \rho_{T,R}$ = joint density estimator for MI
`dD` $\in \mathbb{R}^{nd}$	derivative of D w.r.t. `Tc`
`dr` $\in \mathbb{R}^{p,nd}$	derivative of `rc` w.r.t. `Tc`
`d2psi` $\in \mathbb{R}^{q,q}$	second derivative of the outerfunction ψ w.r.t. r

Table 7.1 Using different distance measures.

This file is E7_Hands_distance_rotation.m

```
%%%
% set up some data
setupHandData; close all;
level = 6; omega = MLdata{level}.omega; m = MLdata{level}.m;

DM = {'SSD','NCC','MI','NGF'}; % the distance measures
str = @(w) sprintf('T(y(%s^o))',num2str(w*180/pi)); % used in plots for titles

% initialize interpolation, here spline interpolation with various theta's
theta= 0;                                          % play with theta!
inter('reset','inter','splineInter2D','regularizer','moments','theta',theta);
[T,R] = inter('coefficients',MLdata{level}.T,MLdata{level}.R,omega);

% initialize a grid xc and compute Rc = R(xc)
xc  = getCenteredGrid(omega,m);
Rc = inter(R,omega,xc);

% initialize the transformation, here rotation in 2D
center = (omega(2:2:end)-omega(1:2:end))'/2;
trafo('reset','trafo','rotation2D','c',center);

% parameter runs in [-pi/2,pi/2]
wc = pi/2*linspace(-1,1,101)';

% run loop over the following distance measures

for k=1:length(DM),
  fprintf('============== %s ========================\n',DM{k})
  dc = zeros(size(wc));                 % allocate memory for D(w(j))
  for j=1:length(wc),                   % loop over all parameters w(j)
    yc = trafo(wc(j),xc(:));            % compute the transformation
    Tc = inter(T,omega,yc);             % compute transformed template
    dc(j) = feval(DM{k},Tc,Rc,omega,m);  % evaluate distance measure

    % do some plots
    if j == 1,                          % initialize the plot
      figure(k); clf;
      subplot(1,3,1); viewImage(Rc,omega,m); title('reference');
      subplot(1,3,2); ph = viewImage(Tc,omega,m); th = title(str(wc(j)));
      subplot(1,3,3); rh = plot(wc(j),dc(j),'r.','markersize',20);
      axis([wc(1),wc(end),-inf,inf]); hold on;      title(DM{k})
    else                                % update the plot
      set(ph,'cdata',reshape(Tc,m)'); set(th,'string',str(wc(j)));
      subplot(1,3,3); set(rh,'visible','off');
      plot(wc(1:j),dc(1:j),'k-','linewidth',2);
      rh = plot(wc(j),dc(j),'r.','markersize',20);    pause(1/100)
    end;
    fprintf('.'); if ~rem(j,50) || j==length(wc), fprintf('\n'); end;
  end;
  fprintf('============== %s done ==================\n',DM{k})
end;
```

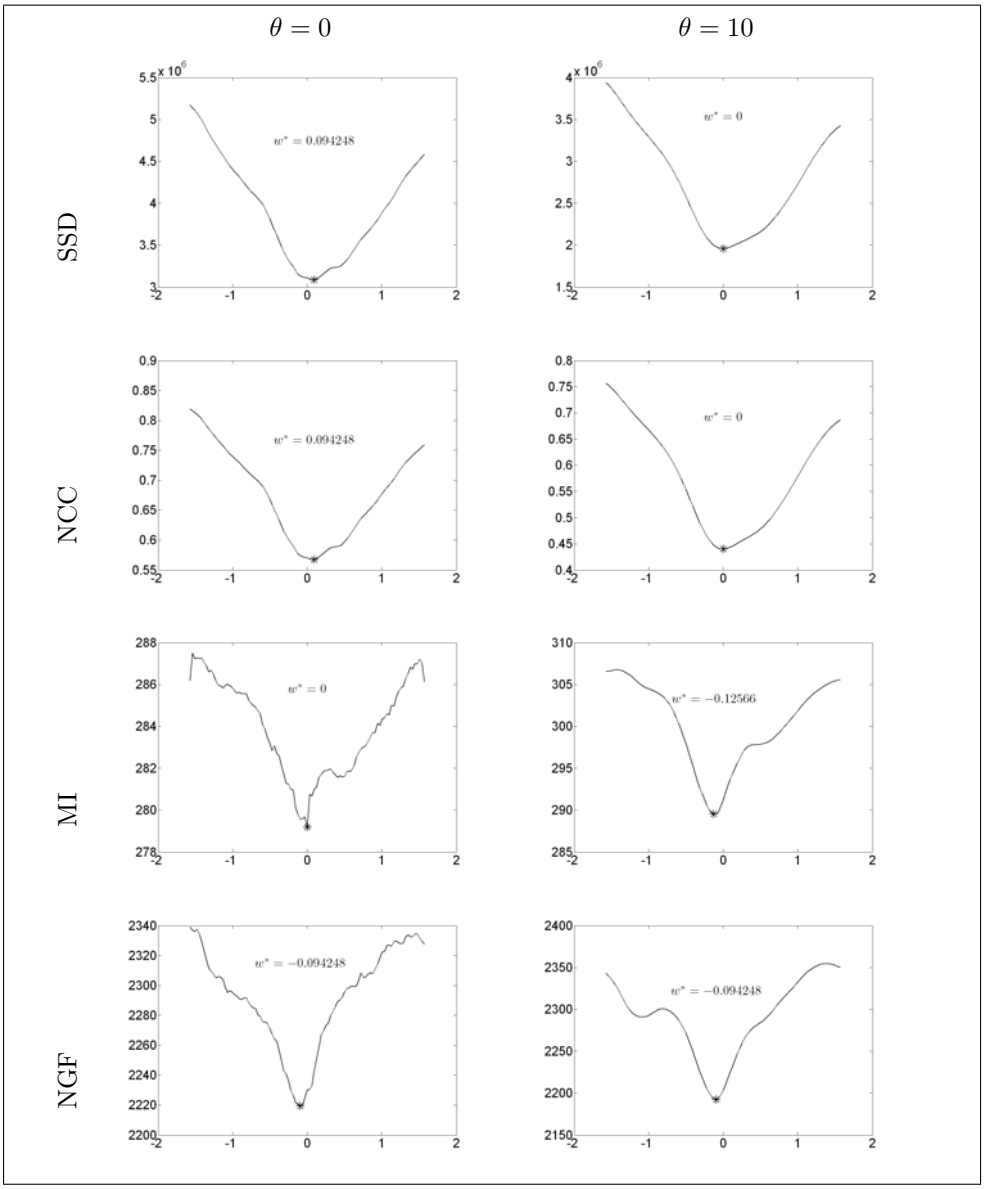
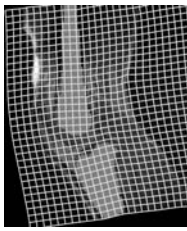


Figure 7.9: *Distance measures versus rotation angle for PET/CT images; see Example 7.8. Left/right columns: using scale parameters* $\theta = 0, 10$.



Example 7.8 (MLPIR Using Various Distances)
This example shows how to incorporate different distance measures into the framework of the multilevel parametric registration discussed in Section 6.6. In addition, multiscale solutions are used. The goal is to find the best affine linear registration using the distance measures presented in this chapter. Results are presented in Figure 7.10.

It appears that the optimal parameters for the transformation do depend on the distance measure. Choosing a proper distance measure for a particular application is thus very important. However, a discussion of distance measure for PET/CT registration is beyond the scope of the book; see, e.g., [184] and the references therein.

This file is E7_PETCT_MLPIR.m

```
%%%
setupPETCTData;                                          % load data

% a list of distance measures to be used
DM = {'SSD','NCC','MI','NGF'};

for dm = 1:length(DM), % run over all distance measures

  % initialize interpolation, using a smooth representation (theta=1e0)
  inter('reset','inter','splineInter2D','regularizer','moments','theta',1e0);

  % initialize transformation, create initial guess and reference for stopping
  trafo('reset','trafo','affine2D'); wStop = trafo('w0'); w0 = wStop;

  % initialize distance and display options
  distance('reset','distance',DM{dm}); distance('disp')

  % run MLPIR using sufficient amount of details (level=5)
  wSmooth =  MLPIR(MLdata,'minLevel',5,'plotIter',0,'plotMLiter',0);

  % refine interpolation (theta=1e-3)
  inter('reset','inter','splineInter2D','regularizer','moments','theta',1e-3);
  level = length(MLdata); omega = MLdata{level}.omega; m = MLdata{level}.m;
  [T,R] = inter('coefficients',MLdata{level}.T,MLdata{level}.R,omega);

  % start PIR, using the result from the smooth problem as starting guess

  % initialize plots
  FAIRplots('set','mode','PIR');
  FAIRplots('init',struct('Tc',T,'Rc',R,'omega',omega,'m',m));

  % optimize
  xc = getCenteredGrid(omega,m);
  Rc = inter(R,omega,xc);
  fctn = @(wc) PIRobjFctn(T,Rc,omega,m,0,[],[],xc,wc); fctn([]);
  [wc,his] = GaussNewtonArmijo(fctn,w0,'Plots',@FAIRplots);

  % visualize results
  yc = trafo(wc,xc);
  R0 = inter(R,omega,xc);
  T0 = inter(T,omega,xc);
  Tc = inter(T,omega,yc);

  figure(11); clf;
  viewImage(T0,omega,m,'axis','off'); hold on;
  plotGrid(yc,omega,m,'spacing',ceil(m/32),'linewidth',2,'color','w');

  figure(12); clf;
  overlayImage2D(Tc,R0,omega,m); axis off;
end;
```

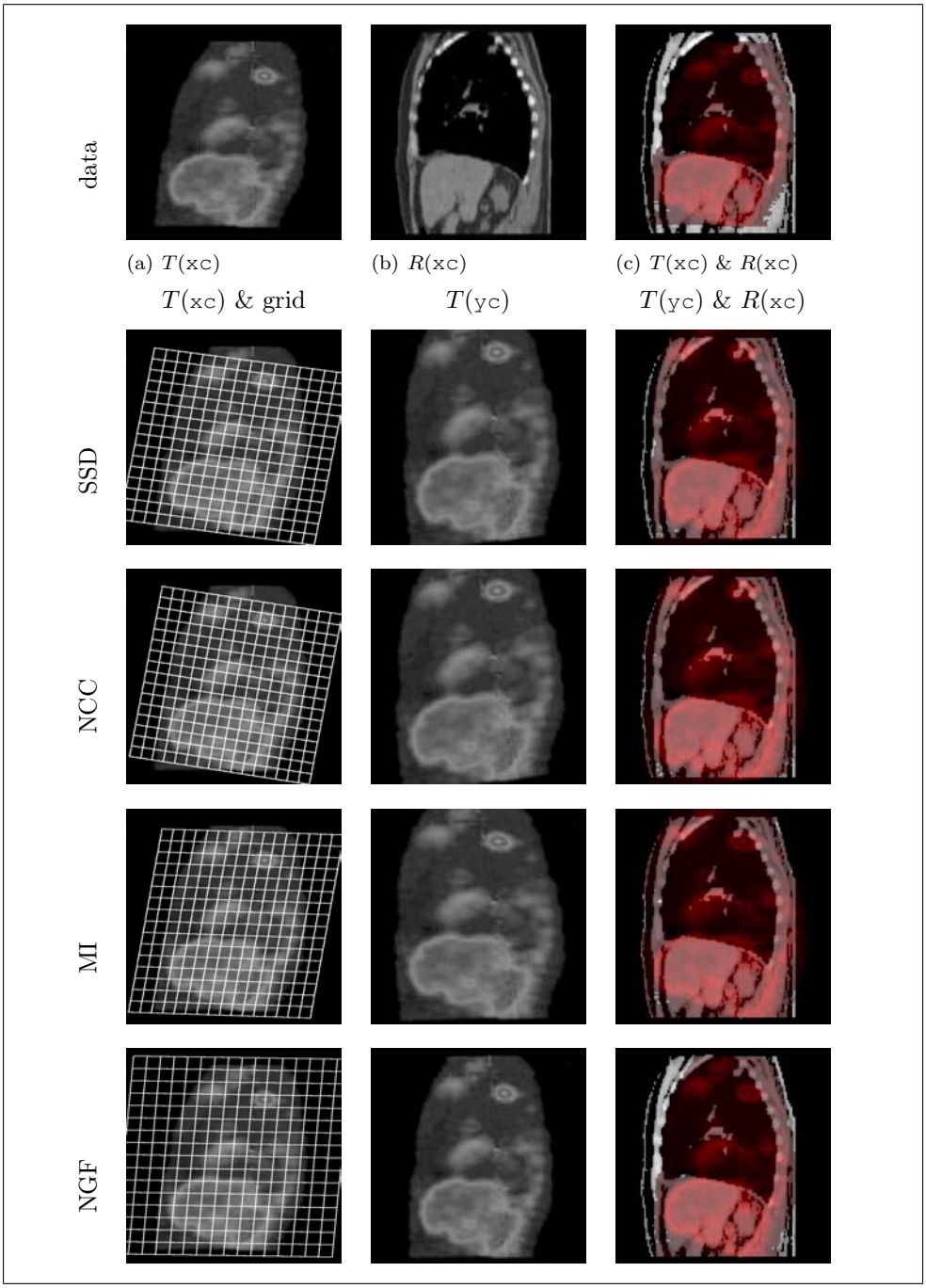
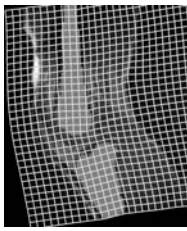


Figure 7.10: *Distance measures for PET/CT data; images from* [184]; *see Example* 7.8. *First row:* (a) *reference*, (b) *template, and* (c) *image "fusion"; following rows: template with overlaid grid for* $\mathtt{yc}$ *(left column), transformed template (middle column), and "fusion" of reference and transformed template (right column). The affine transformations* $\mathtt{yc}$ *are based on MLPIR results.*



Some Guidelines

Providing several distance measures immediately raises the question of which one to be used in a particular application. From an imaging point of view, MI seems to be the most general measure but may causes trouble in identifying a meaningful optimizer. On the other hand, the SSD is well suited for optimization but too narrow for many applications. A practical guideline for a particular application is to choose the distance measure to be as restrictive as possible. In other words, rather than searching for a universal measure, designing a more restrictive and application-specific measure might be the better choice.

7.7 FAIR Tutorials on Distance Measures

FAIR contains the tutorial `BigTutorialDistance` which summarizes a number of smaller tutorials that provide insight into the handling of the tools. Note that some of the more advance tutorials use techniques which are not yet explained.

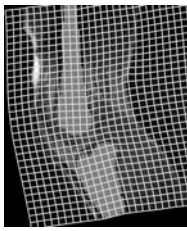
BigTutorialDistance	
E7_Hands_SSDvsRotation	SSD versus rotation (hands)
E7_PETCT_SSDvsRotation	SSD versus rotation (PET/CT)
E7_PETCT_MIvsRotation	MI versus rotation (PET/CT)
E7_US_MIvsRotation	MI versus rotation (US/US)
E7_basic	distances versus rotation
E7_extended	distances versus rotation (ext)
E7_SSDforces	show SSD forces
E6_PETCT_MLPIR	MLPIR, PETCT, affine, SSD
E9_PETCT_MLIR_NGF_mbElas	MLIR, PETCT, NGF, elastic, matrix-based
E9_HNSP_MLIR_TR	MLIR, HNSP, elastic, matrix free

7.8 Exercises

Exercise 7.1
Modify the SSD distance measure such that the outer function $D[r] = \frac{1}{2}\int r^s \, dx$ *is replaced by* $D[r] = \int \log(1 + r^2) \, dx$. *Discuss pros and cons. Implement the new distance measure and its derivative and compare it to SSD and NCC.*

Exercise 7.2
An interesting alternative to the NCC discussed in this chapter is the localized cross-correlation. The idea is to consider correlation pointwise and to convolve it with a window function, for example a Gaussian. Thus, points further away have smaller weight in the computation of the correlation. Explore this idea and implement the localized cross-correlation and its derivatives.

Exercise 7.3
Compute the force fields for NCC, MI, and NGF. Visualize the force fields for the HNSP data of Example 2.8 analogously to Figure 7.1.



Chapter 8
Regularization

The most puzzling point in registration is probably ill-posedness of the problem. To clarify the notation, a problem is well-posed in the sense of Hadamard if it has a solution, the solution is unique and depends continuously on the data; otherwise it is called ill-posed [124]. Section 8.1 provides a small example giving some intuition.

This chapter discusses approaches to address ill-posedness by adding a so-called regularizer $\mathcal{S}$. The idea is to measure quality of candidates and to choose the best candidate with respect to the measure of choice. For image registration, the joint objective functional reads

$$\mathcal{J}[y] = \mathcal{D}[\mathcal{T}[y], \mathcal{R}] + \mathcal{S}[y]. \tag{8.1}$$

Here, $\mathcal{D}$ is a distance measure as discussed in Chapter 7, and the regularizer $\mathcal{S}$ is the topic of this chapter.

From a mathematical point of view, the regularizer should make the registration problem well-posed, i.e., lead to a unique minimizer and preferably to a convex objective function. However, for most practical applications this is probably illusive. The objective function will allow for many local and even global minima, and the results will depend on the starting point, the algorithm, and the implementation.

There is a strong and intended analogue of the spline interpolation discussed in Section 3.4 and to the landmark-based registration of Chapter 5. For spline interpolation or approximation, the problem can be phrased in three variants:

$$D[y] \overset{!}{=} \min \qquad \text{s.t.} \quad S[y] = 0, \tag{8.2}$$

$$S[y] \overset{!}{=} \min \qquad \text{s.t.} \quad D[y] = 0, \tag{8.3}$$

$$D[y] + S[y] \overset{!}{=} \min, \tag{8.4}$$

where $D[y] = \sum_{i=1}^{m} (y(x_i) - d_i)^2$ is the data fitting term and S is the bending energy (3.7). Problem (8.2) asks for the best data fit requiring y to be linear. The solution is the regression line. Problem (8.3) asks for the smoothest function fitting the given data and returns a spline function as the solution. Problem (8.4) finally

results in the multiscale approach discussed in Section 3.6. The thin-plate-spline registration presented in Section 5.3.1 is along the same lines. The only differences as compared to the scalar-valued spline approach are that now the objective y is a vector field and the regularizer S is different compared to (5.3). However, the problem was treated componentwise, and analytic solutions are known in terms of linear combinations of radial basis functions. For more complex distance measures such as MI, asking for an analytic solution is probably too ambitious. Therefore, numerical solutions become important.

This chapter is organized as follows. Section 8.1 provides a brief introduction to ill-posedness in image registration. Section 8.2 introduces the most commonly used regularizer in image registration such as the elastic, diffusion, and curvature regularizer. Section 8.3 addresses discretization issues and Section 8.5 comments on matrix-free implementations.

8.1 Ill-Posedness

Some small examples might give some intuition into why the registration problem is so hard to solve.

Example 8.1 (Forward and Backward Problems)
Suppose three numbers $y_1 = 1$, $y_2 = 2$, and $y_3 = 2$ are given and the problem is to find the sum $T = T(y) = y_1 + y_2 + y_3$. In this example the answer can easily be computed to be five. Now suppose the result is known to be six; then the inverse or backward problem is to find values for y. It is obvious that many solutions exist, so talking about *the* solution is not meaningful.

One could make the assumption that y is constant, i.e., a multiple of $Q = [1, 1, 1]$. Based on this assumption, the solution $y = 2 \cdot [1, 1, 1]$ is uniquely defined.

Example 8.2 (Ill-Posedness)
Another important issue is related to perturbation of the data. Suppose $e = 10^{-16}$ and

$$A = \begin{bmatrix} 1 & 1 \\ 1 & 1+e \end{bmatrix}.$$

The forward problem is to compute $T = T(y) = Ay$. For $y = [2; 0]$, it holds that $T = [2; 2]$. The inverse problem is to compute y given T. Since A is invertible, this problem has the unique solution $y = A^{-1}[2; 2] = [2; 0]$. Perturbing T slightly, say $\tilde{T} = [2; 2 - e]$, the solution of the inverse problem is $\tilde{y} = [3; -1]$. This solution is quite different from y in terms of magnitude and because the components change signs, which can be a serious issue in imaging.

Image registration is inherently ill-posed: For every spatial location $x \in \Omega \subset \mathbb{R}^d$, one is asking for a vector $y(x) \in \mathbb{R}^d$, but generally only a scalar information $\mathcal{T}(y)$ is provided. Since there is no meaningful way to address an ill-posed problem, the standard approach is to regularize, i.e., to modify the problem such that it becomes solvable.

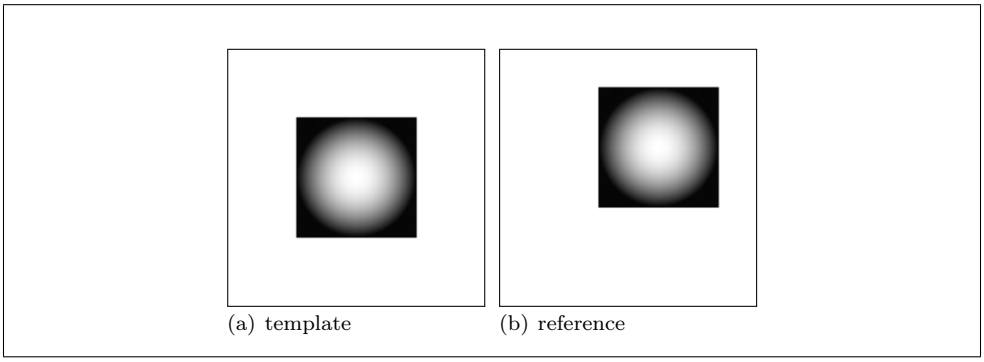


Figure 8.1: *Ambiguity example: squares with texture.*

The analogue for image registration is to restrict the transformation to a certain space, such as the space of rigid transformations. Thus $y = Q(x)w$, and the problem is to find proper coefficients w. A common assumption is that a unique transformation exists in this parameterized space. However, this assumption does not hold, as the following example demonstrates.

Example 8.3 (Ambiguity in PIR)
A more subtle point is emphasized in Figure 8.1, showing a textured square on a background. The ill-posedness of image registration is demonstrated for the very limited set of rigid transformations. One obvious solution to the registration problem is a translation aligning the bottom left corner of the inner square in the template with the bottom left corner in the reference. However, a rotation about 90 degrees of the template image followed by the previous translation yields another perfect match, where now the top left corner matches the bottom left corner in the reference; other solutions can be obtained similarly by rotating the template about 180 or 270 degrees.

Example 8.3 also demonstrates that the ill-posedness of image registration is not related to discretization, local minima of an objective function, the particular algorithmic approach, implementation details, etc. Even for the outstanding human visual system it is impossible to provide *the* solution. The only way to resolve this uncertainty is by providing additional information. The idea is to introduce an additional measure of quality of candidates and to choose the best candidate according to this measure.

Example 8.4 (Simple Regularization)
For the problem in Example 8.1, one may argue for a solution with balanced components. Making the assumption that the components are balanced is precisely how preknowledge enters. One approach would be to model $y = y(w) = [1, 1, 1]w$ and to solve the data fitting term $D(y) = (T(y(w)) - 6)^2 = 0$ for w. Other options are to measure variations of y, for example, with the regularizer

$$S(y) = (y_1 - y_2)^2 + (y_2 - y_3)^2.$$

Then, alternative problems are to minimize the data fit subject to $S(y) = 0$ (which results in the first approach), to minimize S subject to data fit (which picks a solution with minimal variation), or to compromise between data fitting and regularization, i.e., solving

$$D(y) + \alpha S(y) \stackrel{!}{=} \min .$$

This regularization parameter α can be used to put more emphasis on the data fitting or on the preknowledge. In this simple example, all three approaches result in the same solution $y = [2, 2, 2]$.

8.2 L_2-Norm–Based Regularizers

Most regularizers used in current image registration schemes are variants of L_2-norms of derivatives of the displacement $u = y - y^{\text{ref}}$, where y^{ref} allows a bias to a particular solution, e.g., $y^{\text{ref}}(x) = x$ or the result of a parametric preregistration. The purpose of this section is to provide insight into the general concept and to introduce and summarize the most commonly used regularizer such as elastic, diffusion, or curvature.

FAIR 18: L_2-Norm–Based Continuous Regularizers

Regularization considered in FAIR is based on L_2-norms of derivatives of the displacement $u = y - y^{\text{ref}}$:

$$\mathcal{S}[u] = \frac{\alpha}{2} \int_\Omega |\mathcal{B}[u]|^2 \, dx, \tag{8.5}$$

where $\mathcal{B}$ is a differential operator, $|\cdot|$ is a Euclidian norm, and $\alpha > 0$ is a regularization parameter.

The discussion of the regularization parameter α is postponed to the next chapter, where in this chapter $\alpha = 1$ for ease of presentation. Moreover, setting $y^{\text{ref}} = 0$ results in $u = y$.

8.2.1 Examples in One Dimension

Example 8.5 (Norm of y in One Dimension)
Let $d = 1$ and $\mathcal{B}[y] = y$; hence

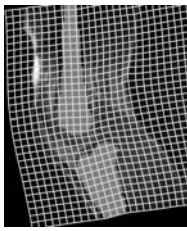
$$\mathcal{S}[y] = \frac{1}{2} \int_{\omega^1}^{\omega^2} (y(x))^2 \, dx.$$

This regularizer is strictly convex and is minimized by $y = 0$.

Example 8.6 (Norm of the Derivative in One Dimension)
Let $d = 1$ and $\mathcal{B} = \partial$, hence $\mathcal{B}[y](x) = \partial y(x) = y'(x)$ and

$$\mathcal{S}[y] = \frac{1}{2} \int_{\omega^1}^{\omega^2} (y'(x))^2 \, dx.$$

This regularizer is convex and is minimized by any constant $y(x) = c$.



8.2.2 Examples in Two Dimensions

Example 8.7 (Norm of y in Two Dimensions)
Let $d = 2$ and $\mathcal{B}[y] = y$; hence

$$\mathcal{S}[y] = \frac{1}{2} \int_\Omega (y^1(x))^2 + (y^2(x))^2 \, dx.$$

This regularizer is strictly convex and is minimized by $y = 0$.

Example 8.8 (Diffusion Operator in Two Dimensions)
The diffusion regularizer basically measures variations of y. This regularizer has been proposed for optical flow problems by Horn and Schunk [140]; for image registration and the connection to Thirion's demons algorithm see [194, 102].

Let $d = 2$, $y = [y^1; y^2]$. The operator $\mathcal{B}$ is a collection of partial derivatives

$$\mathcal{B} = \begin{bmatrix} \nabla & 0 \\ 0 & \nabla \end{bmatrix} = \begin{bmatrix} \partial_1 & 0 \\ \partial_2 & 0 \\ 0 & \partial_1 \\ 0 & \partial_2 \end{bmatrix}.$$

Hence

$$\begin{aligned} \mathcal{S}[y] &= \frac{1}{2} \int_\Omega |\mathcal{B}[y]|^2 \, dx = \frac{1}{2} \int_\Omega \sum_i |\nabla y^i|^2 \, dx \\ &= \frac{1}{2} \int_\Omega (\partial_1 y^1)^2 + (\partial_2 y^1)^2 + (\partial_1 y^2)^2 + (\partial_2 y^2)^2 \, dx. \end{aligned}$$

This regularizer is convex and is minimized by any constant $y(x) = [c^1; c^2]$.

Example 8.9 (Elastic Operator in Two Dimensions)
The elastic regularizer is the elastic potential measuring the energy introduced by deforming an elastic material. It has been introduced to image registration by Broit [80] and is probably the most commonly used regularizer in image registration.

Let $d = 2$ and $y = [y^1; y^2]$. In the diffusion example (Example 8.8), the norms of the gradients ∇y^i are used for regularization. Another important differential operator indicating a change of volume is the divergence $\nabla \cdot y = \partial_1 y^1 + \partial_2 y^2$. The choice

$$\mathcal{B} = \begin{bmatrix} \sqrt{\mu}\, \nabla & 0 \\ 0 & \sqrt{\mu}\, \nabla \\ \sqrt{\lambda + \mu}\, \partial_1 & \sqrt{\lambda + \mu}\, \partial_2 \end{bmatrix},$$

where λ and μ are the so-called Lamé constants. This choice of $\mathcal{S}$ leads to the elastic potential

$$\begin{aligned} \mathcal{S}[y] &= \frac{1}{2} \int_\Omega |\mathcal{B}[y]|^2 \, dx \\ &= \frac{1}{2} \int_\Omega \mu \, \langle \nabla y, \nabla y \rangle + (\lambda + \mu)(\nabla \cdot y)^2 \, dx. \end{aligned}$$

For a physical interpretation see, e.g., [113, 162] and the references therein.

Example 8.10 (Curvature Operator in Two Dimensions)
The curvature regularizer is based on second order derivatives. Let $d = 2$ and $y = [y^1; y^2]$. The choice

$$\mathcal{B} = \begin{bmatrix} \Delta & 0 \\ 0 & \Delta \end{bmatrix} \quad \text{with} \quad \Delta y^i = \partial_{1,1} y^i + \partial_{2,2} y^i$$

results in the curvature regularizer [101, 103],

$$\mathcal{S}[y] = \frac{1}{2} \int_\Omega |\mathcal{B}[y]|^2 \, dx = \frac{1}{2} \int_\Omega (\Delta y^1)^2 + (\Delta y^2)^2 \, dx.$$

While the diffusion and elastic operators have a finite-dimensional nullspace, the nullspace of the curvature operator has infinite dimension and contains all harmonic vector fields, i.e., vector fields for which $\Delta y^i = 0$. The curvature regularizer results in a smoother transformation, which is important when adding additional constraints such as points correspondences; see [103].

8.2.3 Extensions to Higher Dimensions

All of the previously discussed examples naturally extend to higher dimensions d. Here $|x| = \sqrt{x^\top x} = \sqrt{(x^1)^2 + \cdots + (x^d)^2}$ is the Euclidian norm of a vector, I_d denotes the identity matrix in $\mathbb{R}^d$, and $\otimes$ the Kronecker product.

Example 8.11 (Norm of y in Higher Dimensions)
Let $\mathcal{B}[y] = y$; hence

$$\mathcal{S}[y] = \frac{1}{2} \int_\Omega |y|^2 \, dx.$$

This regularizer is strictly convex and is minimized by $y = 0$.

Example 8.12 (Diffusion Operator in Higher Dimensions)
The operator $\mathcal{B}$ is a collection of all partial derivatives

$$\mathcal{B} = I_d \otimes \nabla.$$

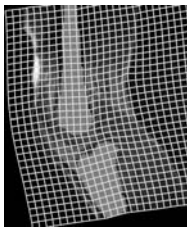
Hence

$$\mathcal{S}[y] = \frac{1}{2} \int_\Omega |\mathcal{B}[y]|^2 \, dx \;=\; \frac{1}{2} \int_\Omega \sum_i |\nabla y^i|^2 \, dx.$$

This regularizer is convex and is minimized by any constant $y(x) = c \in \mathbb{R}^d$.

Example 8.13 (Elastic Operator in Higher Dimensions)
With the so-called Lamé constants μ, λ, and

$$\mathcal{B} = \begin{bmatrix} \sqrt{\mu} I_d \otimes \nabla \\ \sqrt{\lambda + \mu} \;\; \nabla \cdot \end{bmatrix},$$



the elastic potential is

$$\mathcal{S}[y] = \frac{1}{2}\int_\Omega |\mathcal{B}[y]|^2 \, dx \;=\; \frac{1}{2}\int_\Omega \mu \left\langle \nabla y, \nabla y \right\rangle + (\lambda+\mu)(\,\nabla\cdot\, y)^2 \, dx.$$

Example 8.14 (Curvature Operator in Higher Dimensions)
The curvature regularizer is based on second order derivatives. Choosing

$$\mathcal{B} = I_d \otimes \Delta$$

results in the curvature regularizer

$$\mathcal{S}[y] = \frac{1}{2}\int_\Omega |\mathcal{B}[y]|^2 \, dx = \frac{1}{2}\sum_i \int_\Omega (\Delta y^i)^2 \, dx.$$

The remarks from Example 8.10 do apply to the general case.

8.2.4 Thin-Plate-Spline and Curvature Regularizers

Another example for a second order regularizer is the energy of a thin-plate spline, which has been used in many applications; see, e.g., [97, 73, 175, 162]. This section describes the relation between the curvature and the thin-plate-spline regularizers. As the analysis shows, the difference is mainly in the boundary conditions.

Thin-Plate-Spline Energy for $d = 2$

The thin-plate-spline regularizer was introduced in Section 5.3.1 but can also be defined via the inner bilinear form of the univariate functions $f, g : \mathbb{R}^2 \to \mathbb{R}$,

$$\langle f, g \rangle_{\mathrm{TPS}} = \int_\Omega \partial_1^2 f \partial_1^2 f + 2\partial_1\partial_2 f \partial_1\partial_2 g + \partial_2^2 f \partial_2^2 g \, dx,$$

considered on the whole space, i.e., $\Omega = \mathbb{R}^d$. The energy $\mathcal{E}[f] = \frac{1}{2}\langle f, f \rangle_{\mathrm{TPS}}$ measures the bending energy of a thin plate. This bilinear form makes the functional rotationally invariant, an important feature in image processing. In contrast to the curvature regularizer where the energy is obtained from $\nabla^\top \nabla = \partial_1^2 + \partial_2^2$, as discussed above, the energy is obtained from

$$\mathcal{B} = I_2 \otimes \mathcal{B}^1 \quad \text{with} \quad \mathcal{B}^1 = \begin{bmatrix} \partial_1^2 \\ \sqrt{2}\partial_1\partial_2 \\ \partial_2^2 \end{bmatrix}$$

and is given by

$$\mathcal{S}[y] = \int_{\mathbb{R}^2} |\mathcal{B}y|^2 \, dx.$$

Variation of the Thin-Plate-Spline Energy

Assuming that the domain Ω is finite and that f and g are appropriate, the first variation of $\mathcal{S}$ can be computed with the help of the divergence theorem (n denoting the outer normal on the boundary and $d^2 f$ the Hessian of f) as follows:

$$\begin{aligned}
\langle f, g \rangle_{\mathrm{TPS}} &= \langle \mathcal{B}f, \mathcal{B}g \rangle_{L_2(\mathbb{R}^2)} \\
&= \int_\Omega (\partial_1 \nabla f)^\top (\nabla[\partial_1 g]) + (\partial_2 \nabla f)^\top (\nabla[\partial_2 g])\ dx \\
&= \int_{\partial\Omega} n^\top (\partial_1 \nabla f)\ \partial_1 g + n^\top (\partial_2 \nabla f)\ \partial_2 g\ dA \\
&\qquad - \int_\Omega \partial_1 \Delta f, [\partial_1 g] + \partial_2 \Delta f, [\partial_2 g]\ dx \\
&= \int_{\partial\Omega} n^\top (d^2 f) \nabla g\ dA - \int_\Omega (\nabla \Delta f)^\top \nabla g\ dx \\
&= \int_{\partial\Omega} n^\top (d^2 f) \nabla g\ dA - \int_{\partial\Omega} n^\top (\nabla \Delta f) g\ dA + \int_\Omega (\Delta^2 f)\ g\ dx.
\end{aligned}$$

It is well known that a function of finite energy needs to be asymptotically linear for $x \to \infty$. For those functions the boundary integrals vanish and the derivative of $\mathcal{S}$ in direction g is given by $d_g \mathcal{S}[f] = \int_\Omega (\Delta^2 f)\ g\ dx$.

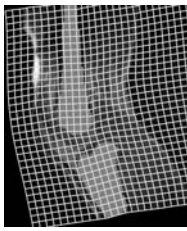
Curvature versus Thin-Plate-Spline Energy

For $d = 2$, the curvature energy can be obtained via a bilinear form of the components of the vector field y. Analogously to the above computations,

$$\begin{aligned}
\langle f, g \rangle_{\mathrm{curvature}} &= \langle \Delta f, \Delta g \rangle_{L_2(\Omega)} = \int_\Omega (\Delta f)\ (\Delta g)\ dx \\
&= \int_{\partial\Omega} (\Delta f)\ (n^\top \nabla g)\ dA - \int_\Omega (\nabla \Delta f)^\top (\nabla g)\ dx \\
&= \int_{\partial\Omega} (\Delta f)\ (n^\top \nabla g) - n^\top (\nabla \Delta f)\ g\ dA + \int_\Omega (\Delta^2 f)\ g\ dx.
\end{aligned}$$

Interestingly, this integral has the very same main part as the form for the thin-plate-spline integral. However, while the spline counterpart is designed to give linear behavior at infinity, the curvature form is not. Note, however, that if $\Omega \longrightarrow \mathbb{R}^2$, the integrability condition still yields a linear behavior. The original paper [103] suggests choosing $\Delta f = n^\top \nabla \Delta f = 0$ on the boundary, which yields a simple implementation. A proper treatment of the boundary conditions is presented in [132].

The difference between the curvature and thin-plate-spline approaches is thus in the boundary conditions. Practically, the appropriate boundary conditions are unknown and thus both approaches are likewise questionable.



8.3 Discretizing L_2-Norm–Based Regularizers

In contrast to the landmark-based registration schemes discussed in Chapter 5, no closed form solution is to be expected for the more general approach discussed in this chapter. Therefore, numerical solutions and thus discretizations of the continuous model become important. Three steps are to be discussed. The numerical computation of the integral which is performed straightforwardly by a midpoint quadrature rule on a cell-centered grid $\mathtt{xc}$, the computation of $\mathcal{B}[u](\mathtt{xc})$ which is performed by concatenating the components of $\mathcal{B}[u]$, and the discretization of the derivatives $\partial_k u^i$.

For the description and interpretation of the various operators, a matrix representation is convenient. For example, if $g = \partial f$ denotes the derivative of a function f, a discrete analogue reads $g^h = \partial^h f^h$, where f^h and g^h are discretizations of the functions f and g, respectively, and ∂^h is a discrete version of the derivative operator ∂. Of course, f^h and g^h are to be vectors (though perhaps of different lengths) and, since ∂ is a linear operator, ∂^h is a matrix. However, it is important to understand that this is only a descriptive tool. For large image sizes storage become prohibited, even though these matrices can be stored efficiently using a sparse matrix format. Section 8.5 gives a brief overview on operator-based techniques which avoid storing the operator and only require the implementation of the action of an operator.

8.3.1 Discretizing First Order Derivatives

1D Case

The central ingredient for computing an approximation of the derivative of a function $u : \mathbb{R} \to \mathbb{R}$ at a cell-centered grid point is a central finite difference

$$\partial u(x_j^c) = \frac{u(x_j^c + 0.5h) - u(x_j^c - 0.5h)}{h} + \mathcal{O}(h^2),$$

where $h = (\omega^2 - \omega^1)/m$ denotes the cell size. The cell-centered difference yields order h^2 accuracy. A drawback is that the function u has to be known on the nodal grid $\mathtt{xc}^n$, whereas the derivative is approximated on the cell-centered grid $\mathtt{xc}^c$,

$$x_j^n = \omega^1 + jh, \quad j = 0, \dots, n, \quad \text{and} \quad x_j^c = \omega^1 + (j - 0.5)h, \quad j = 1, \dots, n;$$

see also Figure 8.2. With $u_j = u(jh)$ it holds that

$$\partial u(x_j^c) \approx (u_j - u_{j-1})/h, \quad j = 1, \dots, n. \tag{8.6}$$

Introducing the discrete 1D derivative operator

$$\partial_m^h = \frac{1}{h} \begin{bmatrix} -1 & 1 & & \\ & \ddots & \ddots & \\ & & -1 & 1 \end{bmatrix} \in \mathbb{R}^{m,m+1},$$

we see that (8.6) can be summarized conveniently by $\partial u(\mathtt{xc}^c) = \partial_m^h u(\mathtt{xc}^n) + \mathcal{O}(h^2)$.

Note that the alternative long stencil $\partial u(ih) \approx (u_{i+1} - u_{i-1})/(2h)$ returns a zero derivative for the high oscillatory input $[0; 1; 0; 1; \dots; 1; 0]$ and in addition requires a particular handling of boundary points. Therefore it is not recommended.

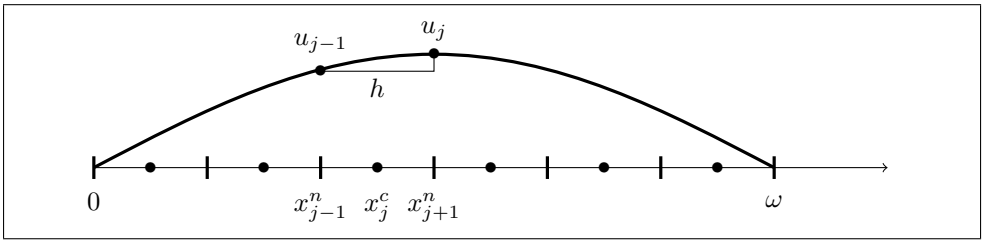
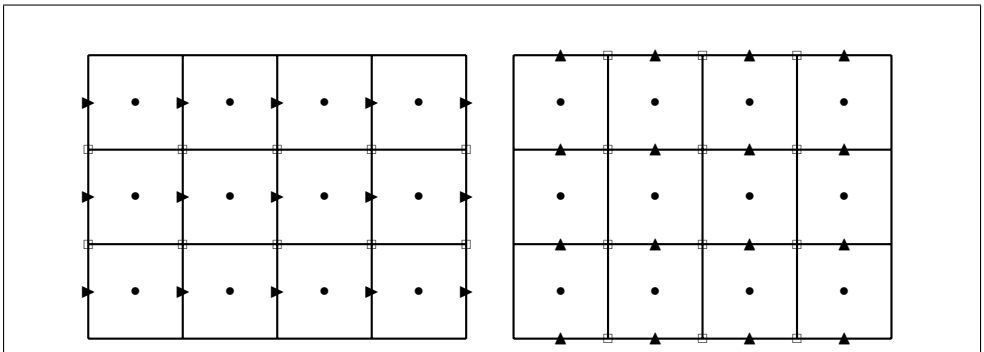


Figure 8.2: *Cell-centered finite difference approximation of a derivative.*

2D Case

The goal is to provide approximations to the derivative at cell-centered grid points for $u = [u^1, u^2]$. Unfortunately, this is impossible. A trade-off is to discretize the functions u^1 and u^2 on the staggered grids $\mathtt{xc}^{s,1}$ and $\mathtt{xc}^{s,2}$, respectively; see also Figure 8.3. The grids are defined by $x_j = [j^1 h^1, j^2 h^2]$ and

$$x^{s,1}_j = x_{j^1, j^2 - 0.5}, \quad j^1 = 0, \ldots, m^1, \quad j^2 = 1, \ldots, m^2,$$
$$x^{s,2}_j = x_{j^1 - 0.5, j^2}, \quad j^1 = 1, \ldots, m^1, \quad j^2 = 0, \ldots, m^2.$$



(a) staggered grid $\mathtt{xc}^{s,1}$ (▶) for u^1 (b) staggered grid $\mathtt{xc}^{s,2}$ (▶) for u^2

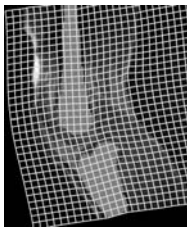
Figure 8.3: *Grids for $d = 2$: cell-centered $\mathtt{xc}^c$ (●), nodal $\mathtt{xc}^n$ (□), and staggered grids $\mathtt{xc}^{s,1}$ (▶) and $\mathtt{xc}^{s,2}$ (▲).*

Setting $u^i_j = u^i(x_j)$ it holds that

$$\partial_1 u^1(x_{j^1 - 0.5, j^2 - 0.5}) \approx (u^1_{j^1, j^2 - 0.5} - u^1_{j^1 - 1, j^2 - 0.5})/h^1,$$
$$\partial_2 u^2(x_{j^1 - 0.5, j^2 - 0.5}) \approx (u^2_{j^1 - 0.5, j^2} - u^2_{j^1 - 0.5, j^2 - 1})/h^2.$$

The computation of $\partial_2 u^1$ and $\partial_1 u^2$ is along the same lines:

$$\partial_2 u^1(x_{j^1, j^2}) \approx (u^1_{j^1, j^2 + 0.5} - u^1_{j^1, j^2 - 0.5})/h^2,$$
$$\partial_1 u^2(x_{j^1, j^2}) \approx (u^2_{j^1 + 0.5, j^2} - u^2_{j^1 - 0.5, j^2})/h^1.$$



Note, however, that these derivatives are located on a subset of the nodal grid; see □ in Figure 8.3.

The higher-dimensional operators can be summarized using these discrete approximations and Kronecker products:

$$\begin{array}{lcccl}
\partial_1^{h,1} & = & I_{m^2} & \otimes & \partial_{m^1}^{h^1} \in \mathbb{R}^{m^1 m^2,(m^1+1)m^2}, \\
\partial_2^{h,1} & = & \partial_{m^2-1}^{h^2} & \otimes & I_{m^1+1} \in \mathbb{R}^{(m^1+1)(m^2-1),(m^1+1)m^2}, \\
\partial_1^{h,2} & = & I_{m^2+1} & \otimes & \partial_{m^1-1}^{h} \in \mathbb{R}^{(m^1-1)(m^2+1),m^1(m^2+1)}, \\
\partial_2^{h,2} & = & \partial_{m^2}^{h} & \otimes & I_{m^1} \in \mathbb{R}^{m^1 m^2,m^1(m^2+1)}.
\end{array}$$

Since different grids were used for the discretization of u^1 and u^2, we also use two different discrete gradients:

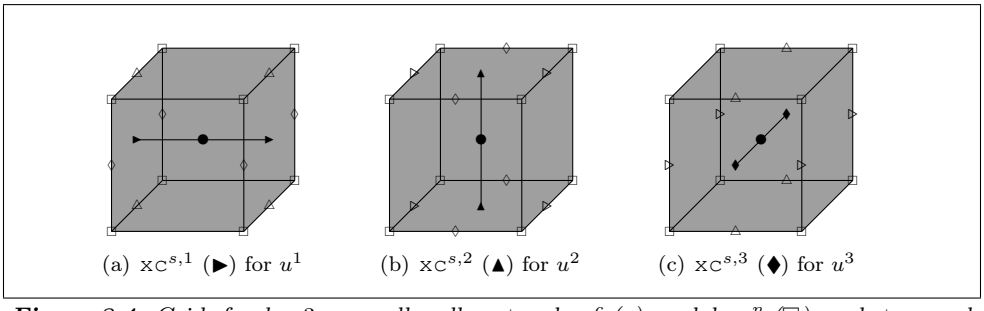
$$\nabla^{h,1} = \begin{bmatrix} \partial_1^{h,1} \\ \partial_2^{h,1} \end{bmatrix} \quad \text{and} \quad \nabla^{h,2} = \begin{bmatrix} \partial_1^{h,2} \\ \partial_2^{h,2} \end{bmatrix}.$$

3D Case

The 3D case is along the same lines. For $u = [u^1, u^2, u^3]$, staggered grids $\mathtt{xc}^{s,1}$, $\mathtt{xc}^{s,2}$, and $\mathtt{xc}^{s,3}$ are introduced; see also Figure 8.4. With $x_j = x_{j^1,j^2,j^3} = [j^1 h^1, j^2 h^2, j^3 h^3]$ and $u_j^i = u_{j^1,j^2,j^3}^i = u^i(x_j)$,

$$\begin{aligned}
x_j^{s,1} &= x_{j^1,j^2-0.5,j^3-0.5}, \quad j^1 = 0,\ldots,m^1,\ j^2 = 1,\ldots,m^2,\ j^3 = 1,\ldots,m^3, \\
x_j^{s,2} &= x_{j^1-0.5,j^2,j^3-0.5}, \quad j^1 = 1,\ldots,m^1,\ j^2 = 0,\ldots,m^2,\ j^3 = 1,\ldots,m^3, \\
x_j^{s,3} &= x_{j^1-0.5,j^2-0.5,j^3}, \quad j^1 = 1,\ldots,m^1,\ j^2 = 1,\ldots,m^2,\ j^3 = 0,\ldots,m^3.
\end{aligned}$$

The discrete derivatives are summarized in Table 8.1. The higher-dimensional operators $\partial_k^{h,i} \in \mathbb{R}^{p_{i,k},n_i}$ are Kronecker products of identity matrices and the discrete 1D derivative operator $\partial_{m^k}^{h_k}$; see Table 8.2.



(a) $\mathtt{xc}^{s,1}$ (▶) for u^1 (b) $\mathtt{xc}^{s,2}$ (▲) for u^2 (c) $\mathtt{xc}^{s,3}$ (◆) for u^3

Figure 8.4: *Grids for $d = 3$ on a cell: cell-centered $\mathtt{xc}^c$ (●), nodal $\mathtt{xc}^n$ (□), and staggered grids $\mathtt{xc}^{s,1}$ (▶), $\mathtt{xc}^{s,2}$ (▲), and $\mathtt{xc}^{s,3}$ (◆).*

Table 8.1 Staggered grid-based discrete derivatives in three dimensions; the symbols in the last column refer to the location of the output as shown in Figure 8.4.

$$
\begin{aligned}
\partial_1 u^1(x_{j^1-0.5,j^2-0.5,j^3-0.5}) &\approx (u^1_{j^1,j^2-0.5,j^3-0.5} - u^1_{j^1-1,j^2-0.5,j^3-0.5})/h^1, && (a,\bullet)\\
\partial_2 u^2(x_{j^1-0.5,j^2-0.5,j^3-0.5}) &\approx (u^2_{j^1-0.5,j^2,j^3-0.5} - u^2_{j^1-0.5,j^2-1,j^3-0.5})/h^2, && (b,\bullet)\\
\partial_3 u^3(x_{j^1-0.5,j^2-0.5,j^3-0.5}) &\approx (u^3_{j^1-0.5,j^2-0.5,j^3} - u^3_{j^1-0.5,j^2-0.5,j^3-1})/h^3, && (c,\bullet)\\
\partial_2 u^1(x_{j^1,j^2,j^3-0.5}) &\approx (u^1_{j^1,j^2+0.5,j^3-0.5} - u^1_{j^1,j^2-0.5,j^3-0.5})/h^2, && (a,\triangle)\\
\partial_3 u^1(x_{j^1,j^2-0.5,j^3}) &\approx (u^1_{j^1,j^2-0.5,j^3+0.5} - u^1_{j^1,j^2-0.5,j^3-0.5})/h^3, && (a,\Diamond)\\
\partial_1 u^2(x_{j^1,j^2,j^3-0.5}) &\approx (u^2_{j^1+0.5,j^2,j^3-0.5} - u^2_{j^1-0.5,j^2,j^3-0.5})/h^1, && (b,\triangleright)\\
\partial_3 u^2(x_{j^1-0.5,j^2,j^3}) &\approx (u^2_{j^1-0.5,j^2,j^3+0.5} - u^2_{j^1-0.5,j^2,j^3-0.5})/h^3, && (b,\Diamond)\\
\partial_1 u^3(x_{j^1,j^2-0.5,j^3}) &\approx (u^3_{j^1+0.5,j^2-0.5,j^3} - u^3_{j^1-0.5,j^2-0.5,j^3})/h^1, && (c,\triangle)\\
\partial_2 u^3(x_{j^1-0.5,j^2,j^3}) &\approx (u^3_{j^1-0.5,j^2+0.5,j^3} - u^3_{j^1-0.5,j^2-0.5,j^3})/h^2, && (c,\triangleright).
\end{aligned}
$$

Table 8.2 Staggered grid-based discrete derivative operators in three dimensions.

$$
\begin{aligned}
\partial_1^{h,1} &= I_{m^3} \otimes I_{m^2} \otimes \partial^{h^1}_{m^1}, && p_{1,1} = m^1m^2m^3, \quad n_1 = (m^1+1)m^2m^3,\\
\partial_2^{h,2} &= I_{m^3} \otimes \partial^{h^2}_{m^2} \otimes I_{m^1}, && p_{2,2} = m^1m^2m^3, \quad n_2 = m^1(m^2+1)m^3,\\
\partial_3^{h,3} &= \partial^{h^3}_{m^3} \otimes I_{m^2} \otimes I_{m^1} && p_{3,3} = m^1m^2m^3, \quad n_3 = m^1m^2(m^3+1),\\
\partial_2^{h,1} &= I_{m^3} \otimes \partial^{h^2}_{m^2-1} \otimes I_{m^1+1}, && p_{1,2} = (m^1+1)(m^2-1)m^3,\\
\partial_2^{h,1} &= \partial^{h^3}_{m^3-1} \otimes I_{m^2} \otimes I_{m^1+1}, && p_{1,3} = (m^1+1)m^2(m^3-1),\\
\partial_1^{h,2} &= I_{m^3} \otimes I_{m^2+1} \otimes \partial^{h^1}_{m^1-1}, && p_{2,1} = (m^1-1)(m^2+1)m^3,\\
\partial_3^{h,2} &= \partial^{h^3}_{m^3-1} \otimes I_{m^2+1} \otimes I_{m^1} && p_{2,3} = m^1(m^2+1)(m^3-1),\\
\partial_1^{h,3} &= I_{m^3+1} \otimes I_{m^2} \otimes \partial^{h^1}_{m^1-1}, && p_{3,1} = (m^1-1)m^2(m^3+1),\\
\partial_2^{h,3} &= I_{m^3+1} \otimes \partial^{h^2}_{m^2-1} \otimes I_{m^1}, && p_{3,2} = m^1(m^2-1)(m^3+1).
\end{aligned}
$$

8.3.2 Discretized Diffusion and Elastic Operators

With the discretized first order differential operators $\partial_k^{h,i}$ on the staggered grid $\mathtt{xc}^s = [\mathtt{xc}^{s,1}; \mathtt{xc}^{s,2}; \mathtt{xc}^{s,3}]$, the discretization of the diffusion and the elastic operators is straightforward. This is shown for the elastic operator for $d = 3$ in Table 8.3. With the nonphysical setting $\lambda = -\mu$, the diffusion operator follows as a special case. The code is summarized in `getElasticMatrixStg`.

Although the matrix B can be quite large, it has a very small number of nonzero elements. The main building block is the discrete 1D derivative which has two nonzeros per row. For $d = 2$ it is thus possible to operate this matrix with MATLAB. However, for $d = 3$, storing the matrix can be too expensive. An efficient implementation therefore relies on matrix-free operations, where only the action of B on a vector u has to be implemented. Efficient $\mathcal{O}(n)$ multigrid-based solutions techniques are discussed in the literature; see [133, 122] and the references therein. The matrix-free code is more evolved and discussed briefly in Section 8.5.

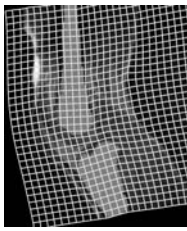


Table 8.3 Discretized 3D elastic operator in matrix form and a plot of the 540 nonzero entries using `spy`; $m = [2, 3, 4]$.

$$B = \begin{bmatrix} \sqrt{\mu}\,\partial_1^{h,1} & 0 & 0 \\ \sqrt{\mu}\,\partial_2^{h,1} & 0 & 0 \\ \sqrt{\mu}\,\partial_3^{h,1} & 0 & 0 \\ 0 & \sqrt{\mu}\,\partial_1^{h,2} & 0 \\ 0 & \sqrt{\mu}\,\partial_2^{h,2} & 0 \\ 0 & \sqrt{\mu}\,\partial_3^{h,2} & 0 \\ 0 & 0 & \sqrt{\mu}\,\partial_1^{h,3} \\ 0 & 0 & \sqrt{\mu}\,\partial_2^{h,3} \\ 0 & 0 & \sqrt{\mu}\,\partial_3^{h,3} \\ \sqrt{\mu+\lambda}\partial_1^{h,1} & \sqrt{\mu+\lambda}\partial_2^{h,2} & \sqrt{\mu+\lambda}\partial_3^{h,3} \end{bmatrix}$$

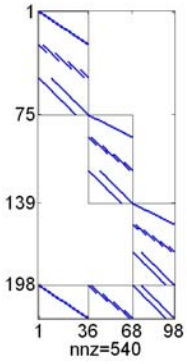
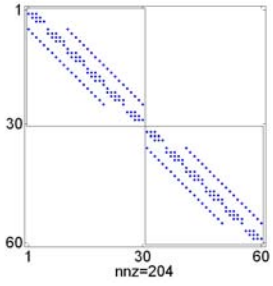


Table 8.4 Discretized 2D curvature operator in matrix form and a plot of the 204 nonzero entries using `spy`; $m = [5, 6]$.

$$B = \begin{bmatrix} \partial_{1,1}^h & 0 & 0 & 0 \\ 0 & \partial_{2,2}^h & 0 & 0 \\ 0 & 0 & \partial_{1,1}^h & 0 \\ 0 & 0 & 0 & \partial_{2,2}^h \end{bmatrix}$$



8.3.3 Discretized Curvature Operator

Since the curvature operator is based on second order derivatives, the situation is slightly different. This discussion is for $d = 2$; the extension to $d = 3$ is left as an exercise. Let $\mathtt{xc}^c$ denote a cell-centered grid with $x_j^c = \left[(j^1 - 0.5)h^1, (j^2 - 0.5)h^2\right]$ and $u_j^i = u^i(x_j^c)$. Up to boundary points it holds that

$$\partial_{1,1} u^i(x_j^c) \approx \frac{u^i_{j^1-1,j^2} - 2u^i_{j^1,j^2} + u^i_{j^1+1,j^2}}{(h^1)^2},$$
$$\partial_{2,2} u^i(x_j^c) \approx \frac{u^i_{j^1,j^2-1} - 2u^i_{j^1,j^2} + u^i_{j^1,j^2+1}}{(h^2)^2}.$$

Since both components of u are discretized on the same grid and in addition are treated analogously, coding is much easier, as for the elastic operator. However, the incorporation of the appropriate boundary conditions defeats this advantage; see

[132] for an appropriate approach. For ease of presentation, Dirichlet zero boundary conditions on $u = y - y^{\text{ref}}$ are assumed in this section: $u(\partial\Omega) = 0$. Hence the discrete operators are

$$\partial^h_{1,1} = I_{m^2} \otimes \partial^{2,h^1}_{m^1} \quad \text{and} \quad \partial^h_{2,2} = \partial^{2,h^1}_{m^2} \otimes I_{m^1}$$

with

$$\partial^{2,h}_m = \frac{1}{(h^1)^2} \begin{bmatrix} -2 & 1 & & \\ 1 & \ddots & \ddots & \\ & \ddots & -2 & 1 \\ & & -1 & 2 \end{bmatrix} \in \mathbb{R}^{m,m}.$$

Thus, it holds that $\partial_{k,k} u^i(\texttt{xc}^c) \approx \partial^h_{k,k} \texttt{uc}^i$, where $\texttt{uc}^i = u^i(\texttt{xc}^c)$; see also Table 8.4. The code is summarized in `getCurvatureMatrix`.

8.3.4 Discretized L_2-Norm–Based Regularizers

The final step is to approximate the integral

$$\mathcal{S}[u] = \int_\Omega \langle \mathcal{B}u, \mathcal{B}u \rangle \; dx = \texttt{hd} \; \|B \; \texttt{uc}\|^2 + \mathcal{O}(\texttt{hd}^2), \tag{8.7}$$

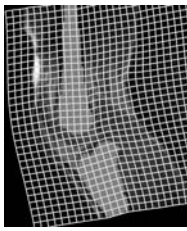
where uc is a discretization of $u = y - y^{\text{ref}}$ and $\texttt{hd} = h^1 \cdots h^d$.

Example 8.15 (Discretized Diffusion Operator in Two Dimensions)
Let $m = [4, 3]$; hence $x^{s,i} \in \mathbb{R}^{m^i_s}$ with $m^1_s = (4+1)3 = 15$ and $m^2_s = 4(3+1) = 16$, $\texttt{uc} = u(x^s) \in \mathbb{R}^{31}$. The matrix B is a 4-by-2 block matrix, where the first column acts on $\texttt{uc}^1 = u^1(x^{s,1}) \in \mathbb{R}^{15}$ and the second column acts on $\texttt{uc}^2 = u^2(x^{s,2}) \in \mathbb{R}^{16}$. Suppose that the output $z = B \; \texttt{uc}$ is decomposed into $z^1, \ldots, z^4$ according to the block structure of B and assume for simplicity that the lengths of the pieces are equal. With $f := (\partial_1 u^1)^2 + (\partial_2 u^1)^2 + (\partial_1 u^2)^2 + (\partial_2 u^2)^2$ and the above approximation,

$$\begin{aligned} \mathcal{S}(u) &= \int_\Omega f(x) \; dx \approx \texttt{hd} \sum_j f(x_j) \\ &\approx \hat{h} \sum_j \{(z^1_j)^2 + (z^2_j)^2 + (z^3_j)^2 + (z^4_j)^2\} = \texttt{hd} \; z^\top z = \texttt{hd} \|B \; \texttt{uc}\|^2. \end{aligned}$$

8.4 Summarizing the Regularization

Regularization is an important and inevitable part of image registration. The regularizers presented in this chapter are based on L_2-norms of derivatives of the displacement $u = y - y^{\text{ref}}$, which are assembled in a differential operator $\mathcal{B}$. The discretizations uc of u and B of $\mathcal{B}$ are used to derive the discretized regularizer. For the first order diffusion and elastic regularizers staggered grids are used, and for the second order curvature regularizer cell-centered grids are used. A reference y^{ref} can be used to shift the nullspace of the regularizer. Practical choices are $y^{\text{ref}}(x) = x$ (update is penalized) or the result of a preregistration.



FAIR enables a convenient use of different regularizers like elastic, diffusion, and curvature. As for the interpolation and distance modules, a unified regularization module `regularizer` is provided and is configured using a persistent parameter `OPTN`; see Section 2.3.7 and `options`.

FAIR 19: Regularization Toolbox

This module computes the function value, the gradient, and the Hessian of an ℓ_2 norm–based regularizer

$$S(\mathtt{uc}) = \frac{\alpha}{2}\,\mathtt{hd}\,\|B\,\mathtt{uc}\|^2,\ dS(\mathtt{uc}) = \alpha\,\mathtt{hd}\,B^\top B\,\mathtt{uc},\ d^2S(\mathtt{uc}) = \alpha\,\mathtt{hd}\,B^\top B, \quad (8.8)$$

where `uc` is a discretization of the displacement $y - y^{\mathrm{ref}}$ on a cell-centered or staggered grid and $\mathtt{hd} = h^1 \cdots h^d$.

`[Sc,dS,d2S]= regularizer(yc,omega,m)`	
`yc` $\in \mathbb{R}^n$	transformation
`omega,m`	specify the domain
`Sc` $\in \mathbb{R}$	function value
`dS` $\in \mathbb{R}^n$	derivative w.r.t. `uc`
`d2S` $\in \mathbb{R}^{n,n}$	second derivative w.r.t. `uc`

A simple example indicating the usage of module `regularizer` is presented. More examples and discussions are postponed to the next chapter, where the overall optimization approach is explained.

Example 8.16 (Regularization in Two and Three Dimensions)
This example indicates how to use `regularizer` and checks the implementation of the derivative.

This file is `E8_regularizationMF.m`

```
%%%%%%%%%%%%%%%%%%
% initialize the regularization and create a starting  point
regularizer('reset','regularizer','mfElastic','alpha',1,'mu',1,'lambda',0);
y0 = @(omega,m) randn(size(getStaggeredGrid(omega,m)));

% 2D example, initialize physical domain and number of discretization points
omega = [0,1,0,1]; m = [16,12];   %

% test derivative of 2D implementation
fctn = @(yc) regularizer(yc,omega,m);  checkDerivative(fctn,y0(omega,m));

% 3D example, initialize physical domain and number of discretization points
omega = [0,1,0,1,0,1]; m  = [16,12,8];

% test derivative of 3D implementation
fctn = @(yc) regularizer(yc,omega,m); checkDerivative(fctn,y0(omega,m));
```

8.5 Matrix-Free Operations

This is a technical section that is not necessary for a general understanding of the registration problem and thus may be skipped in a first reading. However, prac-

tically, the techniques to be explained in this section make the difference between doable and undoable.

Particularly for analysis purposes, it can be advantageous to have access to the discrete operator B in matrix form. However, storing the matrices is a waste of memory, even when using a sparse format. Fortunately, storing can be avoided. A matrix-free implementation of the operations is the topic of this section. Note that $S(y) = \frac{1}{2}\,\mathtt{hd}\; y^\top B^\top B y$ and thus both the action of B and the adjoint $B^\top$ are needed.

The key ingredient are the multiplications by the discrete derivative operator ∂_m^h and its transpose, where

$$\partial_m^h = \frac{1}{h}\begin{bmatrix} -1 & 1 & & \\ & \ddots & \ddots & \\ & & -1 & 1 \end{bmatrix} \in \mathbb{R}^{m,m+1}.$$

This task is being accomplished using the matrix-free computations

$$\begin{aligned} \partial_m^h\ \mathtt{yc} &= \mathtt{[yc(2:m+1)-yc(1:m)]/h}, \\ (\partial_m^h)^\top\ \mathtt{zc} &= \mathtt{[(-zc(1);zc(2:m+1)-zc(1:m);zc(m+1)]/h}. \end{aligned}$$

Note that both operations are of order m and require no intermediate memory.

In the following sections, the operations are outlined for dimension two, the extension to dimension three is straightforward but lengthy.

An implementation of the matrix-free operations is given in `mfBy`. The administrative function `regularizer` is used to control whether the matrix-based or matrix-free version is to be used; see Examples 8.16 and 8.17. In the matrix-based code, the matrix B is generated and stored as a persistent variable. In the matrix-free version, the operations By and $B^\top z$ are performed using `mfBy`.

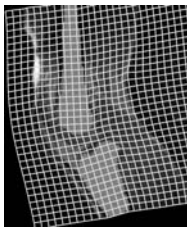
Example 8.17 (Matrix-Free Regularization)
Replacing `mbElastic` by `mfElastic` in Example 8.16 enables the usage of the matrix-free implementation.

8.5.1 Matrix-Free Elastic Operator

The 2D elastic operator can be decomposed into five components: the four partial derivatives $\partial_j^{h,i}$ of the components of the transformation and the divergence. To enable easy access to these components, `yc` is decomposed and reshaped:

```
y1 = reshape(Y(1:n),m(1)+1,m(2)),
y2 = reshape(Y(n+1:end),m(1),m(2)+1).
```

The shortcut `[x]=reshape(X,p)` with `p` chosen accordingly is used to keep the following formulae short. Note that `xc` = `[xc](:)`. Ignoring the Lamé constants, it



holds that

$$\begin{aligned}
[B_1\ \texttt{yc}] &= [\partial_1^{h,1}\ \texttt{y1(:)}] = [I_{m^2} \otimes \partial_{m^1}^{h^1}\ \texttt{y1(:)}] = \partial_{m^1}^{h^1}\ \texttt{y1} \\
&= \texttt{[y1(2:m(1)+1,:)-y1(1:m(1),:)]/h(1),} \\
[B_2\ \texttt{yc}] &= [\partial_2^{h,1}\ \texttt{y1(:)}] = [\partial_{m^2-1}^{h^2} \otimes I_{m^1+1}\ \texttt{y1(:)}] = \texttt{y1}\ (\partial_{m^2-1}^{h^2})^\top \\
&= \texttt{[y1(:,2:m(2))-y1(:,1:m(2)-1)]/h(2),} \\
[B_3\ \texttt{yc}] &= [\partial_1^{h,2}\ \texttt{y2(:)}] = [I_{m^2+1} \otimes \partial_{m^1-1}^{h^2}\ \texttt{y2(:)}] = \partial_{m^1-1}^{h^1}\ \texttt{y2} \\
&= \texttt{[y2(2:m(1),:)-y2(1:m(1)-1,:)]/h(1),} \\
[B_4\ \texttt{yc}] &= [\partial_2^{h,2}\ \texttt{y2(:)}] = [\partial_{m^2}^{h^2} \otimes I_{m^1}\ \texttt{y2(:)}] = \texttt{y2}(\partial_{m^2-1}^{h^2})^\top \\
&= \texttt{[y2(:,2:m(2)+1)-y2(:,1:m(2))]/h(2),} \\
B_5\ \texttt{yc} &= B_1\ \texttt{yc} + B_3\ \texttt{yc}, \\
B\ \texttt{yc} &= [B_1\ \texttt{yc}; B_2\ \texttt{yc}; B_3\ \texttt{yc}; B_4\ \texttt{yc}; B_5\ \texttt{yc}].
\end{aligned}$$

For the adjoint, $\texttt{yc} = B^\top\ \texttt{zc}$, the input is decomposed into five components representing $B_i\ \texttt{yc}$ and the output is decomposed into two components, respectively. Thus

$$B^\top\ \texttt{zc} = \begin{bmatrix} (\partial_1^{h,1})^\top \texttt{z1} + (\partial_2^{h,1})^\top \texttt{z2} + (\partial_1^{h,1})^\top \texttt{z5} \\ (\partial_1^{h,2})^\top \texttt{z3} + (\partial_2^{h,2})^\top \texttt{z4} + (\partial_2^{h,2})^\top \texttt{z5} \end{bmatrix}.$$

The computation of $(\partial_j^{h,i})^\top\ \texttt{zi}$ is analogous to the computation of $\partial_j^{h,i}\ \texttt{yi}$ and thus omitted.

8.5.2 Matrix-Free Curvature Operator

The curvature approach is easier to implement since a cell-centered grid is used, the operator is self-adjoint, and the boundary conditions are simplified. The 2D case is straightforward, while the 3D case requires some extra remarks.

Efficient 2D Implementation

The curvature operator is much more structured than the elastic operator. A tremendous simplification can be achieved by exploiting the Kronecker calculus, and in particular an identity

$$[A \otimes B\ \texttt{x}] = A[\texttt{x}]B^\top,$$

where again $[\texttt{x}] = \texttt{reshape(x,p)}$. The 2D curvature operator is essentially a sum of two products, where one factor is an identity,

$$B = I_2 \otimes (I_{m^2} \otimes \partial_{m^1}^{2,h^1} + \partial_{m^2}^{2,h^2} \otimes I_{m^2})$$

and

$$\partial_m^{2,h} = \frac{1}{h^2} \begin{bmatrix} -2 & 1 & & \\ 1 & \ddots & \ddots & \\ & \ddots & -2 & 1 \\ & & -1 & 2 \end{bmatrix} \in \mathbb{R}^{m,m}.$$

Thus,

$$B\ \mathtt{yc} = \begin{bmatrix} \partial^{2,h^1}_{m^1}\ \mathtt{y1} + \mathtt{y1}\ (\partial^{2,h^2}_{m^2}) \\ \partial^{2,h^1}_{m^1}\ \mathtt{y2} + \mathtt{y2}\ (\partial^{2,h^2}_{m^2}) \end{bmatrix},$$

and since B is self-adjoint, this also gives the multiplication with $B^\top$.

The above formulae are not precisely matrix free since they involve the multiplications by ∂^{2,h^i} and its transpose. However, since these sparse matrices are only of sizes m^i-by-m^i, building and storing may not be an issue.

Efficient 3D Implementation

Unfortunately, the 3D extension is not that simple. The following procedure illustrates how to efficiently compute

$$[A_3 \otimes A_2 \otimes A_1\ \mathtt{x}], \quad \text{where} \quad A_i = I_{m^1} \quad \text{for} \quad i \neq k$$

for a fixed given k.

The idea is to permute the 3D array [xc] such as to put the kth dimension in the first position, to reshape the data to make it two dimensional, to multiply this representation by A_k, and to invert the previous steps; see Example 3.15 in Section 3.6.3.

8.5.3 Matrix-Free Solver for the Linear Systems

The numerical optimization scheme requires the solution of a quasi-Newton system

$$H\,\mathtt{dy} = -\nabla J,$$

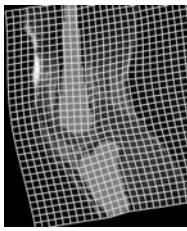
where ∇J denotes the current gradient of the joint objective function, dy an update for the current iterate, and H an approximation to the Hessian which could be either a matrix or a structure assembling the necessary information for a matrix-free operation.

There is a huge variety of solvers for linear systems which do not require building and storing the coefficient matrix [69]. A discussion of those is far beyond the scope of this book.

GaussNewtonArmijo uses a nested function solveGN which computes the solution of the quasi-Newton scheme, where a parameter solver controls the method to be used. The default (solver=[]) is to use the MATLAB backslash operation. However, a multigrid scheme is implemented for the matrix-free elastic regularizer and a preconditioned conjugate gradient scheme for the matrix-free curvature regularizer; see [207, 78, 195] for multigrid techniques, [122] for details on the specific implementation, and [69] for an introduction to Krylov subspace methods.

8.6 FAIR Tutorials on Regularization

FAIR contains the tutorial BigTutorialRegularizer which summarizes a number of smaller tutorials that provide insight into the handling of the tools.



BigTutorialRegularizer	
E8_forces	elastic response to a force field
E8_matrices	creating operator B, matrix based and matrix free
E8_setup	regularizer setup
E8_checkOperations	regularizer: multigrid example

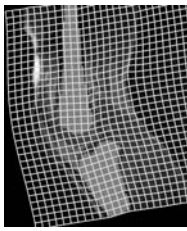
8.7 Exercises

Exercise 8.1
Verify that a necessary condition for a minimizer of the discrete registration problem $J(y) = \min$ *is characterized by* $\alpha B^\top B\, y = f$ *with appropriate* $f(y)$*. Assuming that* f *is given (for example, as one of the force fields from Exercise 7.3) compute and visualize the corresponding transformation.*

Exercise 8.2
Extend the regularizer as follows and discuss restriction on α*, and the pros and cons of a spatially dependent* α*:*

$$\mathcal{S}[u] = \frac{1}{2} \int_\Omega \alpha(x) \mathcal{B}[u]^\top \mathcal{B}[u]\, dx.$$

Exercise 8.3
Implement another second order regularizer.



Chapter 9

Nonparametric Image Registration

All ingredients for the nonparametric image registration (NPIR) have been prepared and are ready for use. Chapter 3 addressed solutions to the forward problem: computing the transformed template image $\mathcal{T}[y]$ for a given transformation y. Chapter 7 provided a general framework for distance measures $\mathcal{D}$, quantifying the similarity of the transformed template and reference images. Moreover, examples for commonly used distance measures are provided. Chapter 8 explained the necessity of regularization, introduced a general framework for L_2-norm based regularization, and provided examples for commonly used regularizers. Using the optimization framework discussed in Section 2.1, the registration problem becomes one of minimizing the joint functional

$$\mathcal{J}[y] = \mathcal{D}[\mathcal{T}[y], \mathcal{R}] + \mathcal{S}[y - y^{\text{ref}}], \tag{9.1}$$

where y^{ref} enables a bias towards a particular solution, for example $y^{\text{ref}}(x) = x$. The obvious difference between the parametric case $y(x) = y(w, x)$, as discussed in Chapter 6, and the nonparametric case is that the transformation is no longer parameterized.

Numerical schemes are required to solve both the parametric and the nonparametric problem, and a discretization of the domain is a first step. The regularization for the parametric case can be either neglected (for low-dimensional transformation spaces and at one's own risk; see Example 8.3 as a warning) or transferred to the coefficient space cf. Section 6.5. In contrast, the discretization for NPIR depends on the chosen regularizer. For example, the elastic regularizer is discretized on staggered grids and the curvature regularizer is discretized on cell-centered grids. To bypass this ambiguity and to enable a unified treatment of various grids, a cell-centered grid interpolation $\mathtt{P}$ is introduced in Section 9.1.1. Let $\mathtt{xc} = \mathtt{xc}^h$ denote a certain discretization of the domain and let $\mathtt{yc} \approx y(\mathtt{xc})$. The discrete version of (9.1) reads

$$J(\mathtt{yc}) = D(T(\mathtt{P} \cdot \mathtt{yc}), R(\mathtt{P} \cdot \mathtt{xc})) + S(\mathtt{yc} - \mathtt{yRef}); \tag{9.2}$$

the discretizations of $\mathcal{D}$ and $\mathcal{S}$ have been discussed in Chapters 7 and 8, respectively.

Note that all ingredients depend on the discretization h. However, for ease of presentation, this dependence is not explicitly indicated.

The overall goal of this chapter is to derive a multilevel minimization scheme, where a coarse to fine sequence of discretizations of (9.1) is attacked. A major point is that on a coarse discretization, the focus is on coarse and important features such that the scheme is not fooled by local features and trapped by local minima. The next step is to lift or prolongate the coarse solution to a finer discretization, where it serves as a perfect starting point. The computations on the finer discretization can be interpreted as a correction step and should yield a minimizer in proximity of the coarse solution. The continuous problem (9.1) links the different discretizations.

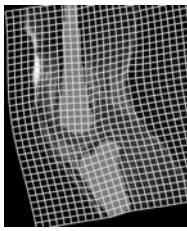
Section 9.1 introduces the objective function for NPIR. As it turns out, the objective function can easily be assembled from the templates provided in the previous chapters. A minor difficulty arises from the fact that the components of the transformation may be discretized on different grids. More specifically, the computations are to be performed on a certain grid, which depends on the regularization and can be cell centered, nodal, or staggered. Since the interpolation requires cell-centered points, a grid interpolation operator `P` to a cell-centered grid becomes necessary; see Section 9.1.1. The reference image on the computational grid is constant and for efficiency reasons computed only once as $\texttt{Rc} = R(\texttt{P} \cdot \texttt{xc})$. The section also comments on matrix-free computations, which turns out to be essential for the 3D examples presented later.

As pointed out, major difficulties in image registration are ill-posedness and the handling of a variety of local or even global minima. Multilevel and/or multiscale techniques are thus essential and it is not recommended to solve a problem using one fixed scale or level. However, for illustration purposes, Section 9.2 shows how to use the nonparametric framework on one fixed level using the sum of squared differences (SSD) distance and an elastic regularizer. A further drawback of the presented approach is that it does ignore characteristics of the elastic regularizer. The linear elasticity theory is based on relations of points and is thus blind to rigid transformations. Example 9.3 shows how to take advantage of a preregistration. More examples are provided in the following sections.

Section 9.3 illustrates how to use a multiscale approach. A numerical solution is computed on a coarse scale, where the optimization process is not trapped by local minima. This solution then serves as a starting point for a finer presentation of the data. Details are added and the current solution is refined. Thus, going from one scale to another can also be interpreted as a correction step.

The next and final step is to also use a multilevel strategy, which is the topic of Section 9.4. Representations of the template and reference images are given for a certain level; see `getMultilevel` for the generation of a multilevel representation. The section describes the outline of the multilevel approach and the missing details such as the prolongation operation, i.e., the transfer from coarse to fine grid based on interpolation. Experiments are presented in Section 9.5.

Numerical optimization of nonconvex functions is a complex problem and the tools to be used depend on characteristics of the registration problems. For the L_2-norm–based distance measures and regularizers, a Gauss–Newton-type technique is expected to perform reasonably well. However, for more nonlinear distances such



as MI, alternatives may result in better performance. Section 9.6 thus presents the ℓ-BFGS and trust-region methods [165].

Finally, Section 9.7 presents some results for 3D registration problems. All computations are performed using Mac OS 10.5.5 (2.5 GHz Intel Core 2 Duo, 2 GB 667 MHz DDR2 SDRAM) and MATLAB 7.6.0.324 (R2008a). The computation time (including all graphical output) is given in seconds.

9.1 Numerical Optimization of Nonparametric Image Registration

This section shows how to use the optimization framework from Section 6.3 for NPIR. A key ingredient is the objective function which needs to be flexible enough to handle different distance measures, regularizations, and discretizations. A grid interpolation operator `P` is introduced to transfer the problem discretization to cell centers as required for the interpolation schemes; see Section 9.1.1. Section 9.1.2 summarizes the coding of the NPIR objective function from a general perspective, and Section 9.1.3 unveils some implementation details.

9.1.1 Grid to Grid Interpolation

The image interpolation schemes assume that all components of y are given for the same spatial location x. This assumption is fulfilled for a cell-centered or nodal discretization. However, for a staggered discretization, the components are discretized on different grids and an additional interpolation step becomes necessary. Using the averaging operator

$$P_\ell = \frac{1}{2}\begin{bmatrix} 1 & 1 & & \\ & \ddots & \ddots & \\ & & 1 & 1 \end{bmatrix} \in \mathbb{R}^{m^\ell, m^\ell+1}$$

for a nodal direction and the identities I_q for the cell-centered directions, the formula for dimensions two and three reads $\texttt{yc}^c = \texttt{P} \cdot \texttt{yc}^s$, where

$$\texttt{P}^{2D} = \begin{bmatrix} I_2 \otimes P_1 & 0 \\ 0 & P_2 \otimes I_1 \end{bmatrix} \text{ and } \texttt{P}^{3D} = \begin{bmatrix} I_3 \otimes I_2 \otimes P_1 & 0 & 0 \\ 0 & I_3 \otimes P_2 \otimes I_1 & 0 \\ 0 & 0 & P_3 \otimes I_2 \otimes I_1 \end{bmatrix}.$$

The function `stg2center` provides this operation in three different ways as matrix-based, matrix-free action of `P`, and matrix-free action of the transpose $\texttt{P}^\top$:

- `P=stg2center(m)` builds and stores the matrix `P` explicitly;
- assuming `yc` is staggered, `yc=stg2center(yc,m)` computes `P*yc`;
- assuming `yc` is cell centered, `yc=stg2center(yc,m)` computes `P'*yc`.

Example 9.1 (Matrix-Free Grid Transfer)
Assuming that `yc` is a 2D staggered grid discretization of y, the following code illustrates how to compute `P*yc` without building `P`.

MATLAB fragment for matrix free P*yc

```
% extracts the components of Y and reformats to 2d arrays
n  = (m(1)+1)*m(2);
y1 = reshape(yc(1:n),    [m(1)+1,m(2)]); % the first  component of yc is m(1)+1-by-m(2)
y2 = reshape(yc(n+1:end),[m(1),m(2)+1]); % the second component of yc is m(1)-by-m(2)+1
% average the j-th component in j-th direction
y1 = (y1(1:end-1,:)+y1(2:end,:))/2;
y2 = (y2(:,1:end-1)+y2(:,2:end))/2;
% return the results
yc = [y1(:);y2(:)];
```

In order to provide the same implementation of the objective function for all potential grids, the superfluous setting `P` $= 1$ is used in case of a cell-centered discretization. Moreover, the function `center` summarizes the various cases and enables a convenient and unified call. Table 9.1 summarizes these operations and their matrix-based and matrix-free usage.

Table 9.1 Grid interpolation operators for matrix-based and matrix-free implementations.

Grid	Matrix-based	Matrix-free
cell-centered	`P = 1`	`P = @(y)y`
staggered	`P = stg2center(m)`	`P = @(y)stg2center(y,m)`
nodal	`P = nodal2center(m)`	`P = @(y)nodal2center(y,m)`
operation	`yc = P*yc`	`yc = P(yc)`

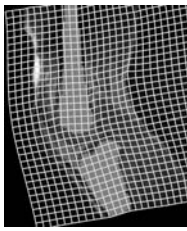
9.1.2 NPIR Objective Function

This section presents the implementation of (9.2). It is assumed that a spatial discretization `xc` is chosen according to the regularization; see Chapter 8 for details. The goal is to compute a discrete approximation `yc` to the transformation y: `yc` $\approx y($`xc`$)$. As specified in Chapter 7, the discretized distance measure depends on samples $T($`P` $\cdot$ `yc`$)$ and $R($`P` $\cdot$ `xc`$)$, where the projection operator `P` is used to interpolated an arbitrary grid on cell centers. The discretization of the regularizer as summarized in Section 8.3 and FAIR 19 (p.131) is used.

The following code presents a simplified version of the objective function for a Gauss–Newton-type implementation. A more sophisticated code which also enables matrix-free computations is provided by `NPIRobjFctn`; see also FAIR 20 (p.141).

NPIRobjFctn

```
function [Jc,dJ,H] = NPIRobjFctn(T,Rc,omega,m,yRef,yc)

% initialize P based on the spatial discretization
P = gridInterpolation(regularizer,omega,m);

% compute transformed template image and derivative
[Tc,dT] = inter(T,omega,P*yc);

% compute distance measure, residual and derivative
[Dc,rc,dD,dr,d2psi] = distance(Tc,Rc,omega,m);

% compute regularizer and derivative
[Sc,dS,d2S] = regularizer(yc-yRef,omega,m);

% evaluate joint function, gradient, and approximation to Hessian
Jc = Dc + Sc;

% derivatives
dT = dT*P; dr = dr*dT; dD = dD*dT; dJ = dD + dS;
H  = dr'*d2psi*dr + d2S;
```



FAIR 20: Objective Function for Nonparametric Image Registration (NPIR)

The discretized objective function for NPIR is

$$J^h(\texttt{yc}) = D^h(T(\texttt{P} \cdot \texttt{yc}), R(\texttt{P} \cdot \texttt{xc})) + S^h(\texttt{yc} - \texttt{yRef}),$$

where `yc` is the current transformed grid, D^h is the discretized distance measure, S^h is a discretized regularizer, and `yRef` is a discretization of y^{ref}. The specific interpolation, distance measure, and regularization are supplied by `inter`, `distance`, and `regularizer`. A grid projection operator `P` is used to transfer the current grid to a cell-centered grid; cf. Table 9.1.

`[Jc,para,dJ,H]= NPIRobjFctn(T,Rc,omega,m,yRef,yc)`	
`T,Rc`	coefficients of template image and discretized reference `Rc` $= R($`P` $\cdot$ `xc`$)$
`omega,m`	define domain Ω and discretization
`yRef` $\in \mathbb{R}^n$	reference for regularization, e.g., `yRef` $=$ `xc`
`yc` $\in \mathbb{R}^n$	current discrete transformation, `yc` $\approx y($`xc`$)$
`Jc` $\in \mathbb{R}$	current objective function value
`para`	a structure collecting variables for visualization
`dJ` $\in \mathbb{R}^{1,n}$	derivative of J w.r.t. `yc`
`H`	approximation to the Hessian

9.1.3 Practical Issues in Coding the NPIR Objective Function

Although the above implementation works in principle, a more sophisticated one is used in FAIR; see `NPIRobjFctn` and FAIR 20 (this page). The function has an additional output which is used to store intermediates for later visualization; see Section 6.3.4 for details on visualization. If the objective function is called with an empty transformation `yc=[]`, it reports the current configuration of the interpolation, distance measure, and regularization. The input `Rc` $= R($`P` $\cdot$ `xc`$)$ for the reference is sampled and computed outside and only once. To avoid an inefficient recomputing of the cell-centered grid interpolation operator, a persistent variable `P` is used which is only updated if the grid changes. Moreover, a function `center` is used to interpolate `yc` on a cell-centered grid and to avoid a case statement handling the matrix-based and matrix-free codes. Finally, a flag `doDerivatives` is used to avoid unnecessary computations of derivatives.

The computation of the discretized regularization is described in Section 8.3 and the computation of the transformed image and the distance is along the same lines as described for the parametric setting; see Section 6.3.

All ingredients optionally return the derivative with respect to the input. In order to compute the derivative of the objective function, it thus remains to apply the chain rule and to add:

$$\texttt{dD} = d_y D = d_T D \cdot d_x T \cdot \texttt{P} \quad \text{and} \quad \texttt{dJ} = d_y J = \texttt{dD} + \texttt{dS}.$$

A note on the approximation to the Hessian is in place. For the regularization part, the analytic derivative of the discretized regularizer `d2S` is used, whereas for the distance measure part, the Gauss–Newton-type approximation introduced in Chapter 7 is used: the distance is phrased as a composition $D(y) = \psi(r(y))$ and the approximation $d^2D \approx dr^\top d2\psi\ dr$ is used. For the matrix-based Hessian version, the approximative Hessian is thus simply

$$H = dr^\top d^2\psi\ dr + \texttt{d2S}.$$

The matrix-free Hessian code is slightly more complicated. Since `P` is an operator and not a matrix, the computation of `dD` has to be modified. Taking advantage of the equality $AB = (B^\top A^\top)^\top$ and the implementation of the transpose $\texttt{P}^\top$, the computation of the derivative is replaced by

$$\texttt{dD} = \texttt{P}((\partial_T D \cdot \partial_x T)^\top)^\top.$$

Using the same approximation to d^2D as for the matrix-based scheme can become expensive and is thus replaced by the diagonal matrix $M = \frac{1}{2}\ \mathrm{diag}[\texttt{P}^\top \hat{M}]$, where $\hat{M} = \mathrm{diag}[dr\ d^2\psi\ dr]$. All information about the action of the approximative Hessian is assembled in a structure `H` and is ready for use in the linear solver within the optimizer, for example a Gauss–Newton scheme; cf. Section 6.3.3. Ignoring some constants, the action of the approximative Hessian on a vector, for example, is given by

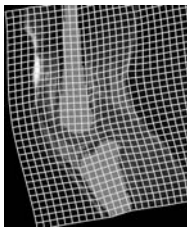
```
H*yc=M*yc+mfBy(mfBy(yc,m,'By'),m,'BTy'),
```

where the matrix-free operations $z = B \cdot$ `yc` and $B^\top \cdot z$ are used; see Section 8.5 for details. Note that only the second derivative of the distance measure D is approximated; the regularization part is exact.

BFGS-type methods as explained in Section 9.6 estimate the Hessian on the fly. FAIR thus also provides an objective function `NPIRBFGSobjFctn`, which does not compute an approximation to the Hessian of D. However, d^2S is the metric of the regularizer and therefore important to the registration problem; see Section 9.6 for details.

9.2 NPIR Experiments on Fixed Level

As pointed out in Chapter 8, major difficulties in image registration are ill-posedness and the handling of a variety of local or even global minimizers. Multilevel and/or multiscale techniques are thus essential and it is not recommended to solve a problem using one fixed scale or level. However, for illustration purposes, this section shows how to use the nonparametric framework on one fixed level using the SSD distance and an elastic regularizer. A further drawback of the approach presented in this section is that it does ignore characteristics of the elastic regularizer. The linear elasticity theory is based on relations of points and thus is blind to rigid transformations. Example 9.3 shows how to take advantage of a preregistration and to compensate for rigid components of the transformation.



The following examples use the visualization as introduced in Section 6.3.4 and the Gauss–Newton optimization scheme as introduced in Section 6.3. The reduction is defined by

$$\text{reduction} = J(\texttt{yc})/J(\texttt{yStop}), \tag{9.3}$$

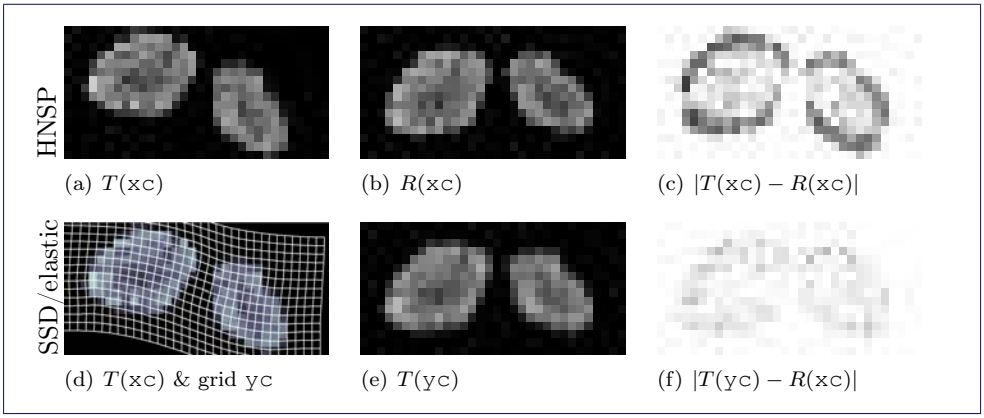
where `yStop` enables a fixed global stopping criterion; here `yStop` = `xc`. The regularization is biased towards the identity, e.g., `yRef` = `xc`.

Example 9.2 (NPIR of HNSP Data on Fixed Level)
The code in Table 9.2 illustrates how to run the NPIR for the configuration described in Figure 9.1. The driver initializes the building blocks such as data, interpolation, distance, and regularizer. Note that the sampled reference $\texttt{Rc} = R(\texttt{P}\cdot\texttt{xc})$ is supplied to the objective function to avoid repetitive computations. The objective function is assembled as discussed in the previous section. The results are shown in Figure 9.1. Since the problem is rather easy, the results are visually pleasing. Note that

Table 9.2 Driver for Example 9.2; see Figure 9.1 for results.

This file is E9_HNSP_NPIR.m

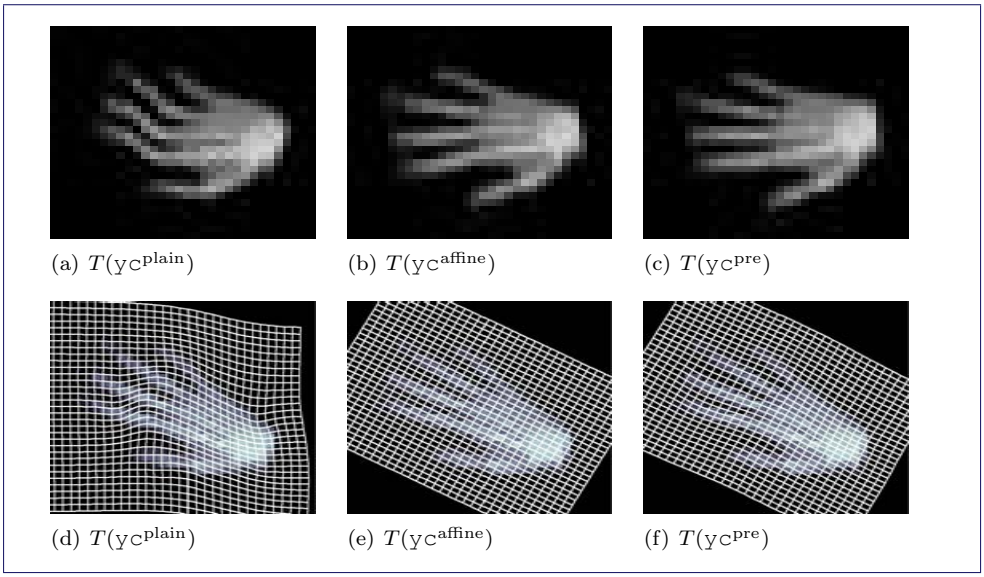
```
% ========================================================================
% Example for NPIR, Nonparametric Image Registration
% (c) Jan Modersitzki 2009/04/06, see FAIR.2 and FAIRcopyright.m.
% \url{http://www.cas.mcmaster.ca/~fair/index.shtml}
%
%   - data                HNSP, Omega=(0,2)x(0,1), level=4, m=[32,16]
%   - viewer              viewImage2D
%   - interpolation       linearInter2D
%   - distance            SSD
%   - regularizer         mbElastic
%   - optimizer           Gauss-Newton
% ========================================================================

% set up data and initialize image viewer
setupHNSPData; FAIRdiary; level = 4; omega = MLdata{level}.omega; m = MLdata{level}.m;

% initialize the interpolation scheme and coefficients
inter('reset','inter','splineInter2D');
[T,R] = inter('coefficients',MLdata{level}.T,MLdata{level}.R,omega,'out',0);
xc    = getCenteredGrid(omega,m);
Rc    = inter(R,omega,xc);

% initialize distance measure
distance('set','distance','SSD');

% initialize regularization, note: yc-yRef is regularized, elastic is staggered
regularizer('reset','regularizer','mbElastic','alpha',1e4,'mu',1,'lambda',0);
y0   = getStaggeredGrid(omega,m); yRef = y0; yStop = y0;


% set up and initialize plots
FAIRplots('reset','mode','NPIR-Gauss-Newton','omega',omega,'m',m,'fig',1,'plots',1);
FAIRplots('init',struct('Tc',T,'Rc',R,'omega',omega,'m',m));

% build objective function, note: T coefficients of template, Rc sampled reference
fctn = @(yc) NPIRobjFctn(T,Rc,omega,m,yRef,yc); fctn([]); % report status

% -- solve the optimization problem ----------------------------------------
[yc,his] = GaussNewtonArmijo(fctn,y0,'maxIter',500,'Plots',@FAIRplots,'yStop',yStop);
% report results
iter = size(his.his,1)-2; reduction = 100*fctn(yc)/fctn(y0);
fprintf('reduction = %s%% after %d iterations\n',num2str(reduction),iter);
diary off
```



(a) $T(\mathtt{xc})$ (b) $R(\mathtt{xc})$ (c) $|T(\mathtt{xc}) - R(\mathtt{xc})|$

(d) $T(\mathtt{xc})$ & grid $\mathtt{yc}$ (e) $T(\mathtt{yc})$ (f) $|T(\mathtt{yc}) - R(\mathtt{xc})|$

Figure 9.1: *NPIR results for HNSP data; cf. Example 2.8:* `splineInter2D`, `SSD`, `mbElastic/staggered`, $[\alpha, \mu, \lambda] = [10000, 1, 0]$, $\Omega = (0, 2) \times (0, 1)$, *level* $= 4$, $m = [32, 16]$, $\#iter = 16$, reduction $= 25.45\%$.

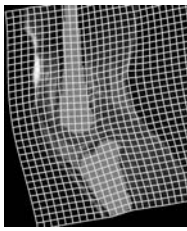


(a) $T(\mathtt{yc}^{\mathrm{plain}})$ (b) $T(\mathtt{yc}^{\mathrm{affine}})$ (c) $T(\mathtt{yc}^{\mathrm{pre}})$

(d) $T(\mathtt{yc}^{\mathrm{plain}})$ (e) $T(\mathtt{yc}^{\mathrm{affine}})$ (f) $T(\mathtt{yc}^{\mathrm{pre}})$

Figure 9.2: *NPIR results for hand data; cf. Example 2.7: plain NPIR (left), affine linear (middle), and combined approaches (right); the configuration for the plain approach:* `splineInter2D`, `SSD`, `mbElastic/staggered`, $[\alpha, \mu, \lambda] = [10000, 1, 0]$, $\Omega = (0, 20) \times (0, 25)$, *level* $= 5$, $m = [32, 32]$, $\#iter = 12$, reduction $\approx 47.31\%$.

the transformed image $T(\mathtt{yc})$ is very similar to the one obtained for a regularized spline transformation ($\alpha = 10^6$); cf. Figure 6.14. The transformations, however, are different. In particular, the elastic transformation appears to be one-to-one while the spline transformation is not.



The next example illustrates how to take advantage of a preregistration. Note that the linear elasticity model assumes that the transformation has no rigid parts. Therefore, it is advised to always use a preregistration.

Example 9.3 (NPIR of Hand Data on Fixed Level)
This example explores the more challenging hand data; cf. Example 2.7. Local minima are to be expected whenever "fingers" meet each other during transformation. The results are shown in Figure 9.2.

As to be expected, the nonparametric registration yields unreasonable results. An affine linear registration gives a better overall impression but does not resolve the nonlinear challenges presented in this example. Finally, a combination of an affine linear preregistration followed by the nonparametric registration yields acceptable though still unsatisfying results. Here, the result of the affine preregistration is used as a starting point and also as a reference for regularization. Note that the stopping is not based on the initial guess `yAffine` but on the global stopping point `yStop=xc`.

9.3 Multiscale Image Registration

Local minima present a serious problem for image registration. An interesting option to reduce the risk of being trapped by local minima is to use a multiscale representation of the data. Let $\theta \in \mathbb{R}$ denote a scale parameter. Starting with a large θ and representations $\mathcal{T}(\theta)$ and $\mathcal{R}(\theta)$ showing only global and important features, a numerical solution $y(\theta)$ is computed. This solution then serves as a starting guess for a representation with smaller θ presenting more details. The following example presents results for a multiscale registration of the hand data.

Example 9.4 (Multiscale Image Registration of Hand Data)
In this example, the hand data is represented on a coarse scale $\theta = 10^3$; see Figure 9.3 and Table 9.3. The left column in Figure 9.3 shows the reference and the transformed template after a two-phase affine linear and elastic registration, both based on the representation of the data on this scale. The numerical minimizer for this scale serves as a starting point for an elastic registration of a finer scale. This procedure is repeated for $\theta = 100, 10, 1, 0$. Note that only a small number of correction steps ($\#\text{iter} \approx 2$) are needed on each scale. The computations are performed for a fixed discretization $m = [128, 128]$ and the numerical solution of the coarser scale is used for regularization on the finer scale.

9.4 Multilevel Image Registration

This section presents the handling of multilevel image registration (MLIR). The first examples presented in Section 9.2 were performed on a fixed and coarse level. Section 9.3 introduced a multiscale approach. There, a solution of a smooth representation of the problem serves as a starting point for a representation with more

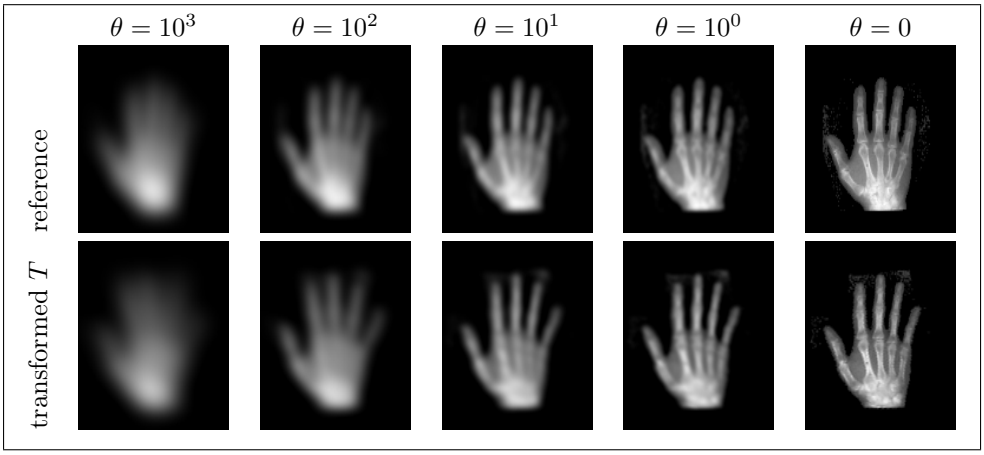


Figure 9.3: *Multiscale image registration results for hand data; cf. Example 2.7; scale* $\theta = 10^3, 10^2, 10, 1, 0$*; visualization of reference image on different scales (top row) and transformed template (bottom row):* `splineInter2D`*, regularized by moments and* $\theta = 10^3, 10^2, 10, 1, 0$*,* `SSD`*,* `mbElastic/staggered`*,* $[\alpha, \mu, \lambda] = [10000, 1, 0]$*,* $\Omega = (0, 20) \times (0, 25)$*, level* $= 7$*,* $m = [128, 128]$*,* $\#iter = [6, 2, 2, 4, 2, 2]$*,* reduction $\approx 2.36\%$*.*

details. Starting with a very smooth representation, this procedure is iterated until all details provided by the initial data are resolved. From an optimization point of view, it is all about smoothing the objective function. A smooth problem is supposed to be easy and based on the good starting point, the more detailed problem can be solved quickly and efficiently. In contrast to this multiscale approach which uses the same discretization on each scale, a multilevel approach also represents the smoother function using a coarser discretization. Note that a multilevel strategy can be the key to a successful registration for many applications.

The FAIR multilevel approach is based on a family of discretizations for the continuous optimization problem (9.1):

$$J^h(y^h) = D(T^h, R^h; y^h) + S^h(y^h - y^{\mathrm{ref},h}), \tag{9.4}$$

where—at least in theory—h goes to zero.

The ideal situation is outlined as follows. Starting with a coarse and thus smooth discretization J^H of the objective function, a numerical optimizer y^H is computed quickly and efficiently: the coarse problem is very small and, on top of that, the objective function is smooth. A starting guess for a finer level is obtained by propagating the current solution, $y_0^h = P_h^H y^H$; the prolongation operator P_h^H is the topic of Section 9.4.2. Since this initial guess is supposed to be in a close neighborhood of a minimizer for the fine problem, only a few steps of a fast convergent optimization scheme should be needed to compute the numerical solution y^h. These steps are iterated until the finest level is reached, the accuracy is within the desired tolerance, or the machine runs out of memory.

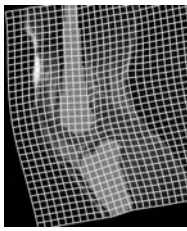


Table 9.3 Driver for Example 9.4; see Figure 9.3 for results.

This file is E9_Hands_MSIR.m

```
% ==========================================================================
% Example for MLSIR, Multiscale Image Registration
% (c) Jan Modersitzki 2009/04/06, see FAIR.2 and FAIRcopyright.m.
% \url{http://www.cas.mcmaster.ca/~fair/index.shtml}
%
%   - data                Hands, Omega=(0,20)x(0,25), level=7, m=[128,128]
%   - viewer              viewImage2D
%   - interpolation       splineInter2D
%   - distance            SSD
%   - pre-registration    affine2D
%   - regularizer         mbCurvature
%   - optimization        Gauss-Newton
% ==========================================================================

% Example for Multiscale Nonparametric Image Registration with preregistration
% (c) Jan Modersitzki 2008/12/30, see FAIRcopyright.m.

% set-up data and image viewer, distance, regularizer, preregistration
setupHandData;  FAIRdiary;
level = 7; omega = MLdata{level}.omega; m = MLdata{level}.m;
distance('set','distance','SSD');
regularizer('reset','regularizer','mbElastic','alpha',1e4,'mu',1,'lambda',0);
trafo('reset','trafo','affine2D'); w0 = trafo('w0');

% the y's are used for:  y0/initial guess, yRef/regularization, yStop/stopping
y0   = getStaggeredGrid(omega,m); yRef = y0; yStop = y0;

% discretization of scale space
theta = [logspace(3,0,4),0];

% initialize the interpolation scheme and coefficients
inter('set','inter','splineInter2D','regularizer','moments');
[T,R] = inter('coefficients',MLdata{level}.T,MLdata{level}.R,omega,'theta',theta(1));
xc    = getCenteredGrid(omega,m);
Rc    = inter(R,omega,xc);

% -- the PIR preregistration ------------------------
beta = 0; M = []; wRef = []; xc = getCenteredGrid(omega,m);
fctn = @(wc) PIRobjFctn(T,Rc,omega,m,beta,M,wRef,xc,wc);
[wc,his] = GaussNewtonArmijo(fctn,w0,'maxIter',500);
reduction = fctn(wc)/fctn(w0);
yc    = grid2grid(trafo(wc,getNodalGrid(omega,m)),m,'nodal','staggered');
Yc = {yc}; ITER = max(his.his(:,1)); REDUCTION = reduction;
% parameter for NPIR
NPIRpara = {'maxIter',500,'Plots',@FAIRplots,'yStop',yStop}

% loop over scales
for j=1:length(theta),
  % compute representation of data on j'th scale
  [T,R] = inter('coefficients',MLdata{level}.T,MLdata{level}.R,omega,'theta',theta(j));
  xc    = getCenteredGrid(omega,m);
  Rc    = inter(R,omega,xc);

  % build objective function and regularizer
  yRef = yc;
  fctn = @(yc) NPIRobjFctn(T,Rc,omega,m,yRef,yc);

  % -- solve the optimization problem ----------------------------------------
  FAIRplots('set','mode','NPIR-GN-elastic','omega',omega,'m',m,'fig',j+3,'plots',1);
  FAIRplots('init',struct('Tc',T,'Rc',R,'omega',omega,'m',m));
  [yc,his]  = GaussNewtonArmijo(fctn,yc,NPIRpara{:});
  reduction = fctn(yc)/fctn(yStop);
  Yc{end+1} = yc; ITER(end+1) = max(his.his(:,1)); REDUCTION(end+1) = reduction;
end;
diary off
```

9.4.1 Outline of MLIR

This section presents a summary of the code; see also Table 9.4. Implementation details are provided in FAIR 21 (this page) and `MLIR`. The first phase is the initialization of parameters such as a flag for performing a parametric preregistration, maximum number of iterations, or optimization schemes to be used. The interesting part is the loop over the different levels running from coarse to fine, which is explained next.

The loop over the different levels can be divided into four steps. The first step is the initialization of the current grid `xc` and the coefficients for $\mathcal{T}$ and $\mathcal{R}$ and the computation of `Rc` $= R($`P` $\cdot$ `xc`$)$, where `P` denotes a cell-centered grid interpolator; cf. Section 9.1.1. On the coarsest level, an optional parametric preregistration is performed, which returns optimal parameters `wc` for the parametric model. The second step is to initialize the reference transformation `yRef` used for regularization and as an initial guess on the coarsest level. If no preregistration is performed, `yRef` $=$ `xc`, where `xc` denotes the current grid. Otherwise, `yRef` $= y($`wc`$,$ `xc`$)$. Note that the optimal parameters do not depend on the discretization (though the numerical approximation does), but the representation of the transformation does. Note also that the transformation model requires the same discretization for all components of the grid. Therefore, the code computes `yRef` as an interpolation of a transformed nodal grid on the current grid. The third step is to initialize a good starting guess. The preregistration result `yRef` is used on the coarsest level and on finer levels, a prolongated version of the solution from the previous level is used; details are provided in the next section. The fourth and final step is the computation of a numerical optimizer on the current level.

FAIR 21: Multilevel Image Registration (MLIR)

This function minimizes $\mathcal{J}[y]$ (cf. (9.1)) using a sequence of discretizations J^h for $h = h_{\text{coarse}} : h_{\text{fine}}$.

`[yc,wc,his] = MLIR(MLdata,varargin)`	
`MLdata`	multilevel presentation of the data; see `getMultilevel`
`varargin`	optional list of input parameters, e.g., `{'plots',0}`
`yc`	numerical optimizer
`wc`	parameters of optional preregistration
`his`	structure collecting the iteration history

9.4.2 Prolongation Operator

A simple interpolation could be used for the prolongation operator. However, particularly for the staggered grid, boundary conditions for the transformation enter into play. A simplified approach in terms of coding is based on the interpolation of the displacement $u(x) = y(x) - x$ and the assumption that u is zero outside the domain Ω.

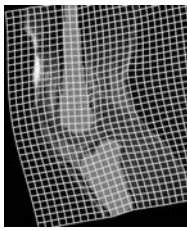


Table 9.4 Code fragment for MLIR; see FAIR 21 (p. 148) and `MLIR` for details.

```
function [yc,wc,his] = MLIR(MLdata,varargin)

% set up default parameter
[not presented here, see code for details]

% initialization
[not presented here, see code for details]

for level=minLevel:maxLevel,

  message(sprintf('%s: level %d from %d to %d, %s',...
    mfilename,level,minLevel,maxLevel,dimstr(MLdata{level}.m)));

  % STEP 1: store old grid, update m, initialize grid and data coefficients
  [not presented here, see code for details]

  if level == minLevel && parametric,
    wc = PIR(fctn,w0);
  elseif level == minLevel, % no pre-registration
    wc = [];
  end;

  % STEP 2: compute yRef = xc or yc = trafo(wc,xc) on the appropriate grid
  [not presented here, see code for details]

  % STEP 3: initialize starting guess y0
  if level == minLevel,
    y0 = yRef;                              %  best known so far
  else
    y0 = xc + mfPu(yc - xOld,omega,m/2);  % prolongate yc (coarse) to y0 (current)
  end;

  % STEP 4: call NPIR
  [yc,his] = NPIR(fctn,y0);

  % update iteration history
  [not presented here, see code for details]

end;%For level
```

The displacement field u is known on a coarse grid $\texttt{xc}^H$ and to be interpolated on a finer grid $\texttt{xc}^h$, where it is assumed that $H = 2h$. Fixing the ith component of u, $v = u^i$, the values are expanded in each direction using linear interpolation. FAIR supplies the function `mfPu` which enables a matrix-free computation of $P_h^H \texttt{uc}^H$ and $(P_h^H)^\top \texttt{uc}^h$. With $\texttt{xOld} = \texttt{xc}^h$, the computation of the initial guess thus reads

$$\texttt{y0 = xc + mfPu(yc - xOld,omega,m/2).}$$

Details of the grid-dependent prolongation operator are given in the following sections.

Nodal Grid

The situation is illustrated in Figure 9.4. Let v be given on the coarse grid $\texttt{xc}^H$ (big open squares). Intermediate values ($\times$) are computed for a finer discretization in the x_1 direction,

$$V_{i_1,i_2} = v_{i_1,i_2} \quad \text{and} \quad V_{i_1+0.5,i_2} = 0.5 v_{i_1,i_2} + 0.5 v_{i_1+1,i_2}. \tag{9.5}$$

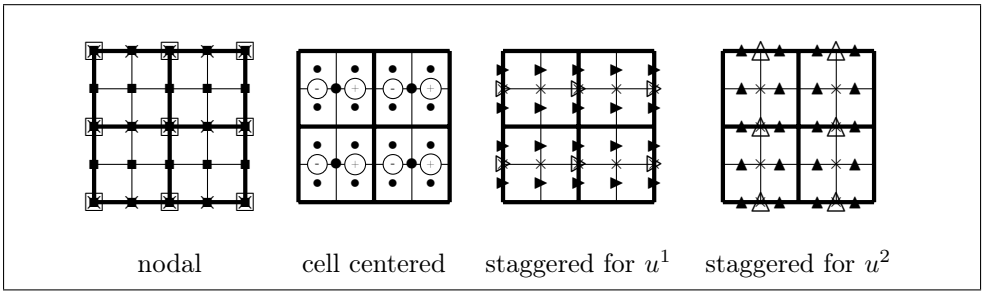


Figure 9.4: *Prolongation of 2D grids.*

Finally, the h discretization (small black squares) is then obtained by expanding the intermediates in the x_2 direction,

$$W_{i_1,i_2} = V_{i_1,i_2} \quad \text{and} \quad W_{i_1,i_2+0.5} = 0.5V_{i_1,i_2} + 0.5V_{i_1,i_2+1}. \tag{9.6}$$

Note that the coarse grid is part of the fine grid; no boundary conditions are required.

Cell-Centered Grid

The situation is illustrated in Figure 9.4. Let v be given on the coarse grid $\mathtt{xc}^H$ (big black dots). Intermediate values are computed for a finer discretization in x_1 direction ($\oplus$ and $\ominus$):

$$\left.\begin{array}{rcl} V_{i_1-0.25,i_2} & = & 0.75v_{i_1-0.5,i_2} + 0.25v_{i_1+0.5,i_2}, \\ V_{i_1+0.25,i_2} & = & 0.25v_{i_1-0.5,i_2} + 0.75v_{i_1+0.5,i_2}. \end{array}\right\} \tag{9.7}$$

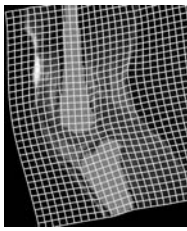
The h discretization (small black dots) is obtained by expanding the intermediates in the x_2 direction. The formulae read

$$\left.\begin{array}{rcl} W_{i_1,i_2-0.25} & = & 0.75V_{i_1,i_2-0.5} + 0.25V_{i_1,i_2+0.5}, \\ W_{i_1,i_2+0.25} & = & 0.25V_{i_1,i_2-0.5} + 0.75V_{i_1,i_2+0.5}, \end{array}\right\} \tag{9.8}$$

where the displacement is assumed to be zero outside the domain Ω. Note that the set of fine grid points does not contain the coarse grid points.

Staggered Grids

The situations are illustrated in Figure 9.4. For u^1, intermediates are computed in the x_1 direction using the nodal interpolation, (9.5) ($\times$). The h discretization (small black left-right triangles) is obtained from (9.8). For u^2, intermediates are computed in the x_2 direction using the nodal interpolation ($\times$). The final h discretization (small black bottom-top triangles) is obtained using the cell-centered interpolation for the intermediates.



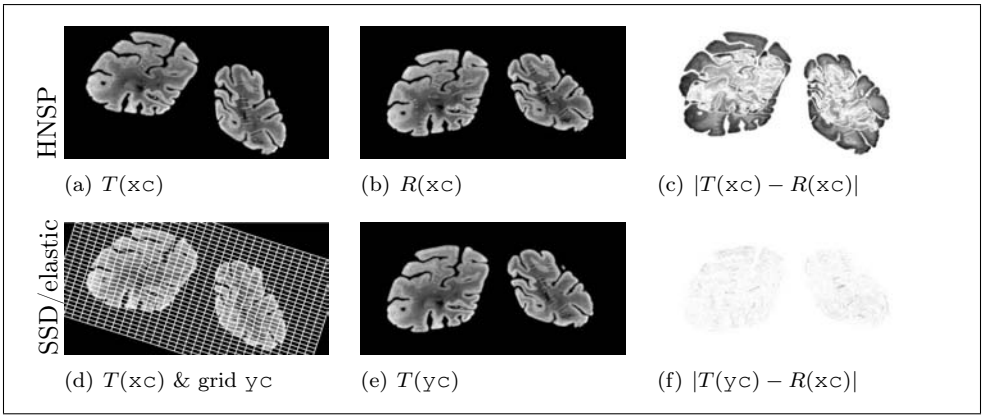
9.5 MLIR Experiments

The examples from Sections 9.2 and 9.3 are continued. The reduction in the objective function is measured by (9.3) and the number of iterations is listed as a vector, where the ith component lists the number of iterations performed on the ith level.

Example 9.5 (MLIR: HNSP, SSD, Elastic)
The driver initializes the building blocks and calls `MLIR` for registration. Results and configuration are shown in Figure 9.5. As mentioned before, the data is well suited for registration, and the impressive results are thus not surprising.

This file is `E9_HNSP_MLIR_SSD_mbElas.m`

```
% ============================================================================
% Example for MLIR, MultiLevel Image Registration
% (c) Jan Modersitzki 2009/04/04, see FAIR.2 and FAIRcopyright.m.
% \url{http://www.cas.mcmaster.ca/~fair/index.shtml}
%
%   - data                HNSP, Omega=(0,2)x(0,1), m=[ 512 256]
%   - viewer              viewImage2D
%   - interpolation       splineInter2D
%   - distance            SSD
%   - pre-registration    affine2D
%   - regularizer         mbElastic
% ============================================================================

close all, help(mfilename);

setupHNSPData
inter('reset','inter','splineInter2D','regularizer','moments','theta',1e-2);
distance('reset','distance','SSD');
trafo('reset','trafo','affine2D');
regularizer('reset','regularizer','mbElastic','alpha',5e2,'mu',1,'lambda',0);

[yc,wc,his] = MLIR(MLdata,'maxLevel',8);
```



(a) $T(\mathtt{xc})$ (b) $R(\mathtt{xc})$ (c) $|T(\mathtt{xc}) - R(\mathtt{xc})|$

(d) $T(\mathtt{xc})$ & grid $\mathtt{yc}$ (e) $T(\mathtt{yc})$ (f) $|T(\mathtt{yc}) - R(\mathtt{xc})|$

Figure 9.5: *MLIR results for HNSP data; cf. Example 2.8:* `splineInter2D`, `SSD`, *pre:* `affine2D`, `mbElastic/staggered`, $[\alpha, \mu, \lambda] = [500, 1, 0]$, $\Omega = (0, 2) \times (0, 1)$, *level* $\ell = 3 : 8$, $m = [512, 256]$, $\#iter = [2, 2, 5, 2, 2, 2]$, reduction $= 1.04\%$, *time* $= 36\,s$.

Example 9.6 (MLIR: Hands, SSD, Elastic and Curvature)
This example is along the same lines as the previous one; results and configuration for an elastic and curvature registration are shown in Figure 9.6. In contrast to the previous example, the registration task is nontrivial and the results are impressive.

Although the transformed images $\mathcal{T}(y^{\text{elas}})$ and $\mathcal{T}(y^{\text{curv}})$ are very close and also $J^{\text{elas}}(y^{\text{elas}}) \approx J^{\text{curv}}(y^{\text{curv}})$, the transformation appears to be different: $y^{\text{elas}} \not\approx y^{\text{curv}}$. The elastic transformation is closer to the affine preregistration and more rough, i.e., it shows greater local variation, while the curvature transformation is smoother.

The previous two examples illustrate how to run FAIR with two different regularizations and also demonstrate an important point in image registration: Although the transformation can differ significantly, the observable output, i.e., the transformed images, look very similar. As discussed in Chapter 8, without addi-

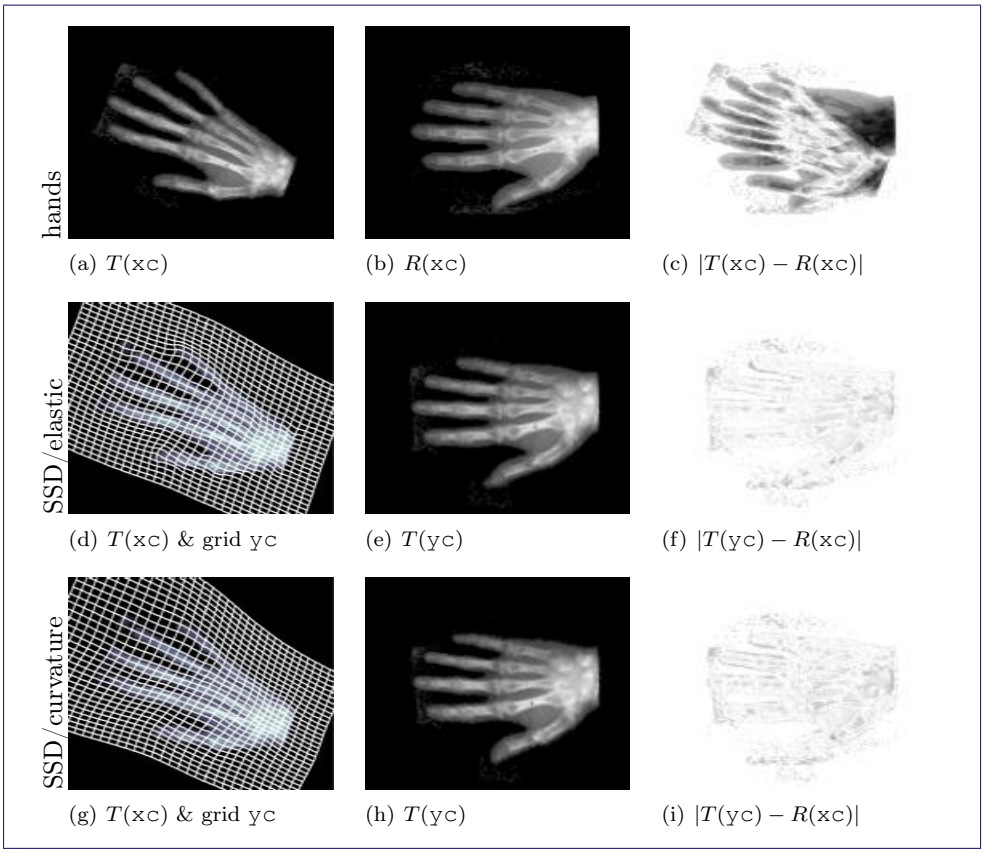
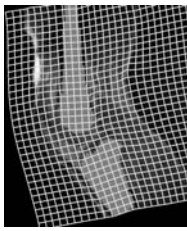


Figure 9.6: *MLIR results for hand data (cf. Example 2.7,) elastic and curvature:* `splineInter2D`, `SSD`, *preregistration:* `affine2D`, `mbElastic/staggered`, $[\alpha, \mu, \lambda] = [1000, 1, 0]$, $\Omega = (0, 20) \times (0, 25)$, *level* $\ell = 3 : 7$, $m = [128, 128]$, $\#iter = [2, 3, 4, 2, 2]$, reduction $= 4.55\%$, *time* $= 12\,s$; *for* `mbCurvature/cell centered`: $\alpha = 1000$, $\#iter = [4, 4, 3, 2, 2]$, reduction $= 4.51\%$, *time* $= 18\,s$.



tional assumptions or knowledge, it is impossible to decide which transformation is "the best." From a modeling point of view, the regularizer quantifies the meaning of "best." In the previous example, the elastic transformation is optimal (assuming that the data was deformed elastically) and the curvature is best (assuming that the data has been deformed by a curvature-type energy); see Chapter 10 for a more detailed discussion.

9.6 Alternative Numerical Optimizers

This section briefly sketches some alternatives to the Gauss–Newton-type optimization used in the previous sections. For a more detailed description of these and other methods on numerical optimization, see the literature summarized in Section 1.4.4 and in particular [165].

Especially when using an MI-based distance measure, the Gauss–Newton approximation of the Hessian can become computationally expensive. A BFGS method provides an interesting option, as it estimates the Hessian on the fly and does not require explicit computations. The next sections provide some details and implementations of a limit memory BFGS method (ℓ-BFGS) and a trust-region method.

9.6.1 *l*-BFGS

An excellent overview on BFGS method is provided in [165, §8.1]. The central idea is to approximate the inverse of the Hessian by an update obtained from an initial approximation H_0 and a sequence of differences of search directions and gradients; see [165] for details and `lBFGS` for an implementation.

The central idea is to replace the objective function by a quadratic model

$$J(y+s) \approx q(s) := J(y) + dJ(y)\ s + \frac{1}{2} s^\top H s$$

and to continue with an updated iterate $y \leftarrow y + tz$, where t is determined by a line search. An obvious though possibly expensive choice for H would be the Hessian d^2J or a Gauss–Newton-type approximation. The BFGS idea is to replace this operator by an approximation which is obtained by using only gradient information. A reasonable choice yielding fast convergence is given by the Broyden family. This is a class of operators satisfying the secant equation

$$H_k s_k = z^k, \quad z^k = \nabla J(y^{k+1}) - \nabla J(y^k),$$

where y^k denotes the kth iterate; see, e.g., [95, 145].

In order to take full advantage of this option, only a small modification of the objective function is necessary: Obviously, the Gauss–Newton approximation is no longer needed. Thus, the call of the distance function simplifies to

```
[Dc,rc,dD] = distance(Tc,Rc,omega,m).
```

Since the Hessian of the regularizer d^2S is used as an initial approximation H_0 to the Hessian, the regularization part stays unchanged. In both the matrix-based and the

matrix-free implementation, the operator d^2S is computed only once on each level. Thus, the output of the objective function does not change but the interpretation of the approximative Hessian does. In the Gauss–Newton version, this approximation is $H = dr^\top d^2\psi \; dr + \texttt{d2S}$, whereas in the BFGS version $H = \texttt{d2S} + \beta I$. Here, a scaled identity is added to enforce a positive definite approximative Hessian. Note that the scheme used in FAIR updates approximations of the inverse Hessian, and the formulae are thus presented for $\hat{H} \approx H^{-1}$, $\hat{H}_0 = H_0^{-1}$.

Following [165, §8.1], the BFGS search direction is computed from

$$\texttt{dy}^k = -\hat{H}_k \nabla J(y^k) \tag{9.9}$$

and the new approximation to the inverse Hessian is

$$\hat{H}_{k+1} = (I - \rho_k s_k z_k^\top)\hat{H}_k(I - \rho_k z_k s_k^\top) + \rho_k s_k s_k^\top, \tag{9.10}$$

where $s_k = y^{k+1} - y^k$, $z_k = \nabla J(y^{k+1}) - \nabla J(y^k)$, $\rho_k = 1/(z_k^\top s_k)$, and it assumed that $z_k^\top s_k \neq 0$.

FAIR uses an implementation of a limited memory BFGS (ℓ-BFGS) scheme [156] and a recursive implementation of $\hat{H}_{k+1}$. During iteration, the differences $s = \texttt{yc} - \texttt{yOld}$ and $z = \texttt{dJ} - \texttt{dJold}$ are computed. If $z^\top s > 0$, these vectors provide valuable information on the Hessian and are added to the list of BFGS vectors, which stores at most the last L vectors. The new search direction is computed in three steps based on (9.9) and (9.10). Let $d_0 = -\texttt{dJ}^\top$ denote the current gradient and $\alpha = (s^\top d_0)/(z^\top s)$. Thus, $d_1 = (I - \rho_k z_k s_k^\top)d_0 = d_0 - \alpha z$. The next step is to compute $d_2 = \hat{H}_{k-1} d_1$ using only the previous BFGS vectors or by solving $\hat{H} d_2 = d_1$ in case the list of BFGS vectors is empty. The final step is to compute

$$\texttt{dy} = (I - \rho_k s_k z_k^\top)d_2 + \rho_k s_k s_k^\top d_0 = d_2 + (\alpha - (z^\top d_2)/(z^\top s))s.$$

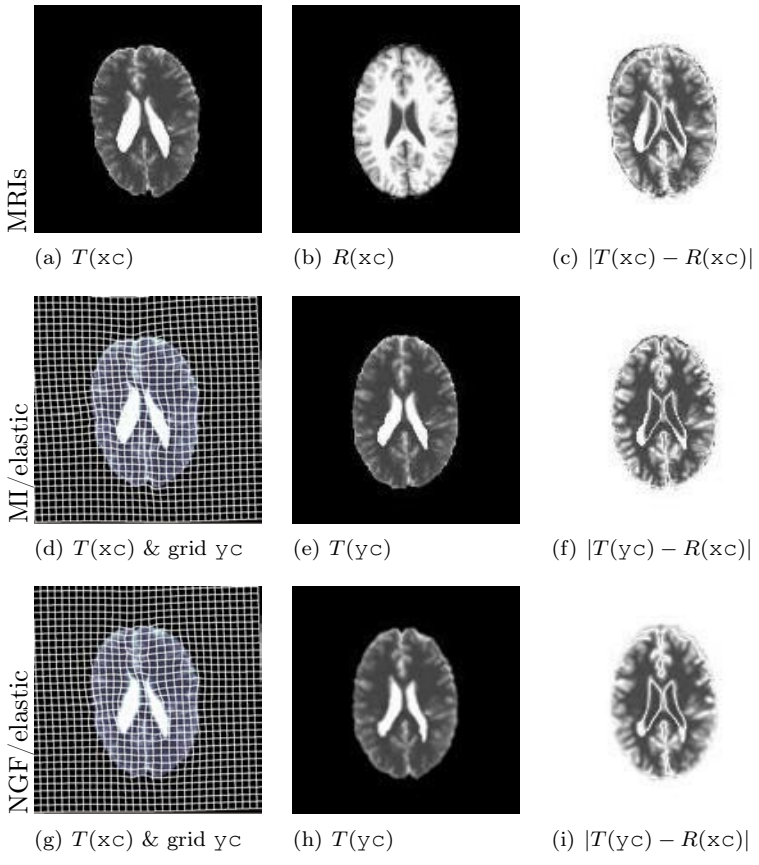
The code is presented in `lBFGS`.

9.6.2 MLIR Using an ℓ-BFGS Scheme

The next examples illustrate how to use different distance measures combined with elastic and curvature regularizations and the ℓ-BFGS scheme within FAIR.

Example 9.7 (MLIR: MRIs Using ℓ-BFGS, MI and NGF, Elastic) The data is taken from the T1 and T2 weighted MRIs of Example 2.9. MI and NGF are used as distance measures. Note that the negative MI is implemented for minimization. Thus, the reduction measure (9.3) becomes meaningless and is not presented in this example. The parameterization of MI is presented in the code below and the parameters for NGF are $\eta = 50$ for the edge parameter and moment regularized spline approximation with $\theta = 0.1$ for the image representation; cf. Section 3.6.

The result for MI and NGF are comparable both in terms of transformation and performance; see Figure 9.7. As is typical for multimodal registration, the problem is highly nonlinear, and as a result many iterations are required to satisfy the stopping criteria.



(a) $T(\mathrm{xc})$ (b) $R(\mathrm{xc})$ (c) $|T(\mathrm{xc}) - R(\mathrm{xc})|$

(d) $T(\mathrm{xc})$ & grid yc (e) $T(\mathrm{yc})$ (f) $|T(\mathrm{yc}) - R(\mathrm{xc})|$

(g) $T(\mathrm{xc})$ & grid yc (h) $T(\mathrm{yc})$ (i) $|T(\mathrm{yc}) - R(\mathrm{xc})|$

Figure 9.7: *MLIR results for MRI data (cf. Example 2.9) using MI and NGF distances, elastic regularization, and ℓ-BFGS optimizer:* `splineInter2D`*, preregistration:* `rigid2D`*,* `mbElastic/staggered`*, $\Omega = (0,128)\times(0,128)$, level $\ell = 4:7$, $m = [128,128]$; for MI: $[\alpha,\mu,\lambda] = [0.0001,1,0]$, time $= 35\,s$; for NGF: $[\alpha,\mu,\lambda] = [0.1,1,0]$, time $= 35\,s$.*

This file is `E9_MRIhead_MLIRlBFGS_MI_mfElas.m`

```
% ========================================================================
% Example for MLIR, MultiLevel Image Registration
% (c) Jan Modersitzki 2009/04/06, see FAIR.2 and FAIRcopyright.m.
% \url{http://www.cas.mcmaster.ca/~fair/index.shtml}
%
%   - data                MRI (head), Omega=(0,128)x(0,128), level=4:7, m=[128,128]
%   - viewer              viewImage2D
%   - interpolation       splineInter2D
%   - distance            MI
%   - pre-registration    rigid2D
%   - regularizer         mbElastic
%   - optimization        lBFGS
% ========================================================================

close all, help(mfilename);

setupMRIData
inter('reset','inter','splineInter2D','regularizer','none','theta',1e-3);
distance('reset','distance','MI','nT',8,'nR',8);
trafo('reset','trafo','rigid2D');
regularizer('reset','regularizer','mfElastic','alpha',1e-4,'mu',1,'lambda',0);
[yc,wc,his] = MLIR(MLdata,...
  'PIR',@lBFGS,'PIRobj',@PIRBFGSobjFctn,...
  'NPIR',@lBFGS,'NPIRobj',@NPIRBFGSobjFctn,...
  'minLevel',4,'maxLevel',7,'parametric',1,'plotMLiter',0);
```

Example 9.8 (MLIR: PET/CT Using ℓ-BFGS)
The data is taken from 2D cross sections of PET and CT images; cf. Example 2.10. MI and NGF are used as distance measures combined with elastic and curvature regularizations.

All techniques are combined with a multilevel spline representation for the images, $\Omega = (0, 140) \times (0, 151)$, level $\ell = 4 : 7$, thus $m = [128, 128]$ on the finest level. A rigid preregistration and the elasticity parameters $\mu = 1$ and $\lambda = 0$ are used in all cases. A staggered grid discretization is used for the elastic regularizer and a cell-centered discretization for the curvature regularizer.

The MI examples use a spline-based interpolation scheme and a 32-by-32 spline grid for the Parzen-window estimator. The NGF examples are based on the edge parameter $\eta = 25$ and a moment regularized spline approximation of the data with $\theta = 0.01$; see Chapter 7 for details. The choices of the regularization parameter α, the numbers of iterations spent on the four levels, and the overall time are presented in the next table; see Figure 9.8 for visualization.

Distance/regularizer	α	#iterations	Time
MI/elastic	10^{-4}	$[25, 25, 4, 25]$	79 s
MI/curvature	10^{-1}	$[25, 8, 8, 25]$	64 s
NGF/elastic	0.5	$[1, 2, 3, 2]$	21 s
NGF/curvature	10	$[1, 2, 8, 1]$	28 s

The parameters are hand picked and the choices lead to comparable results for the MI and NGF pairings. The MI approaches have a tendency to match the outer contours of the PET to the outer contour of the CT, while the NGF approaches aim to match the outer contour of PET to the contour of the inner part of the body. As to be expected, the elastic approaches stay closer to the rigid preregistration, as the elasticity regularizer also penalizes linear transformations. In contrast, the curvature regularizer is smoother but more aggressive in terms of the boundary conditions.

A typical driver function reads as follows.

This file is E9_PETCT_MLIRlBFGS_NGF_mbCurv.m

```
% ============================================================================
% Example for MLIR, MultiLevel Image Registration
% (c) Jan Modersitzki 2009/04/06, see FAIR.2 and FAIRcopyright.m.
% \url{http://www.cas.mcmaster.ca/~fair/index.shtml}
%
%   - data                PETCT, Omega=(0,140)x(0,151), level=4:7, m=[128,128]
%   - viewer              viewImage2D
%   - interpolation       splineInter2D
%   - distance            NGF
%   - pre-registration    rigid2D
%   - regularizer         mbCurvature
%   - optimization        lBFGS
% ============================================================================


close all, help(mfilename);

setupPETCTdata
inter('reset','inter','splineInter2D','regularizer','moments','theta',1e-2);
distance('reset','distance','NGF','edge',25);
trafo('reset','trafo','rigid2D');
regularizer('reset','regularizer','mbCurvature','alpha',10);
[yc,wc,his] = MLIR(MLdata,...
  'NPIR',@lBFGS,'NPIRobj',@NPIRBFGSobjFctn,...
  'minLevel',4,'maxIterNPIR',25,'parametric',1,'plotMLiter',0);
```

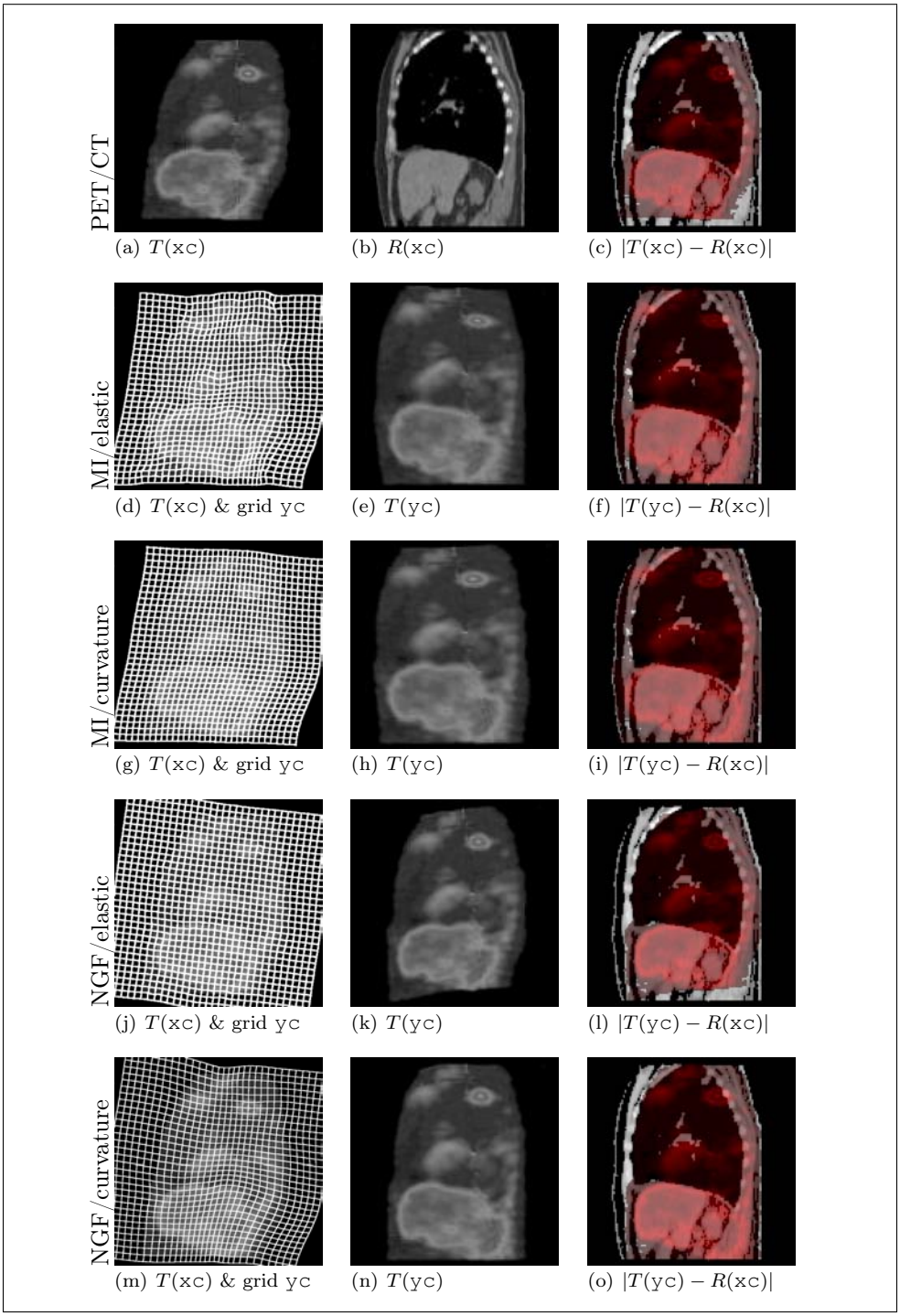


Figure 9.8: *MLIR results for PET/CT data (cf. Example 2.10) using MI and NGF distances combined with elastic and curvature regularization and ℓ-BFGS optimizer; see Example 9.8 for details.*

9.6.3 Trust-Region Methods

A trust-region (TR) method presents another exciting option for numerical optimization. As for the ℓ-BFGS method discussed in the previous section, only the basic ideas are outlined; details can be found, for example, in [165, §4] or any good book on numerical optimization; see Section 1.4.4.

The basic idea of a Newton-type method is to approximate the objective function in a neighborhood of the current iterate y by a quadratic polynomial

$$q(z) = J(y) + dJ(y)z + \frac{1}{2}z^\top Hz, \tag{9.11}$$

where H is an approximation to the Hessian, and to minimize this quadratic. This works terrifically, provided the function is well behaved or the current iterate is sufficiently close to a minimizer. Fast quadratic convergence can be obtained if H is sufficiently close to the Hessian d^2J. Problems are to be expected if the Hessian d^2J is not positive definite, as a Newton-type method then may fail. The Gauss–Newton approach bypasses this problem by replacing the Hessian by the positive definite $H^{GN} = dr^\top \psi\, dr + \alpha B^\top B + \beta I$, and the BFGS method replaces the Hessian by rank updates of an initially positive definite $\alpha B^\top B + \beta I$ which preserves the positive definiteness. Both approaches neglect information obtained in a step of negative curvature.

The trust-region idea is to use the above model only in a certain neighborhood of the current iterate. More precisely, the length of a step is constrained by an adaptively controlled trust-region; see Figure 9.9. The update is defined by the solution of

$$q(z) = \min \quad \text{subject to} \quad |z| \le \delta,$$

where a typical initial choice is $\delta = \max\{1, \|y_0\|\}$. This constraint turns an indefinite or negative definite approximation of the Hessian into valuable information. For example, in the situation presented in Figure 9.9, the minimizer of the constrained problem is $y - \delta$, the search direction leads to the minimizer and the length of the step is bounded by the trust-region radius δ. In principle, this idea works with any approximation to the Hessian, e.g., the Gauss–Newton or ℓ-BFGS approach. FAIR supplies a Gauss–Newton approximation.

The adaption of the trust-region is controlled by the ratio ρ of the actual andpredicted reduction,

$$\rho = \frac{J(y) - J(y+z)}{q(0) - q(z)}.$$

This ratio being reasonably large ($\rho > \rho_{\text{enlarge}}$) implies a good correspondence between model and objective function, and the trust-region is thus enlarged ($\delta \leftarrow \delta s_{\text{enlarge}}$) and the iterate is updated by z. Otherwise, the trust-region is reduced to $\delta' = \delta s_{\text{reduce}}$ and a new update z' is computed. FAIR provides an implementation based on Steihaug's approach [188]; see also [165, p. 75]. The idea is to solve the linear system $Hz = -dJ^\top$ using a preconditioned conjugate gradient scheme [137, 69]. Starting with $x_0 = 0$, the lengths of the accumulated CG iterates x_k is monotonically increasing. Thus, z satisfying $\|z\| \le \delta'$ can be computed retrospectively by

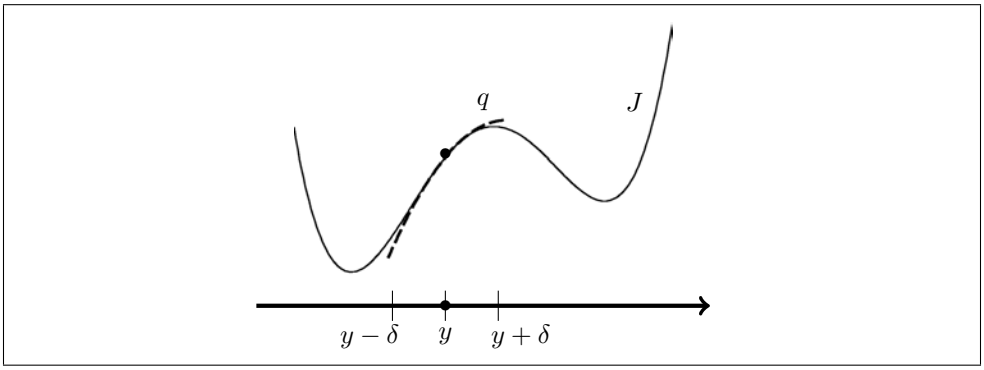


Figure 9.9: *Objective function J, quadratic model q, and trust-region $\{z : |z - y| \leq \delta\}$.*

picking j such that $\| \sum_{k=1}^{j} x_k \| \leq \delta'$; see `TrustRegion` for implementation details. Typical parameters which are also used in this implementation are $\rho_{\text{enlarge}} = 0.25$, $s_{\text{enlarge}} = 2$, $s_{\text{reduce}} = 0.5$.

9.7 Examples in Three Dimensions

Example 9.9 (MLIR, 3D Knee: Gauss–Newton, ℓ-BFGS, Trust-Region) These experiments are based on 3D knee data provided by Thomas Netsch, Philips Research, Hamburg, Germany. The data of size $m = [128, 64, 128]$ is given on the domain $\Omega = (0, 128) \times (0, 62) \times (0, 128)$ and is represented on levels $\ell = 3 : 6$. A 3D rendering of the sampled template and reference as well as a 2D visualization based on orthogonal views of the data are shown in Figure 9.10: the second row visualizes sagittal views of the template, reference, and the difference, and the last two rows visualize the results of registrations based on Gauss–Newton, ℓ-BFGS, and trust-region approaches. The building blocks are: `SSD`, `affine3D` preregistration, `mfElastic/staggered`, and $[\alpha, \mu, \lambda] = [500, 1, 0]$.

The number of iterations († indicates stopping due to maximum number of iterations), the reduction (cf. (9.3)), and the computation times are summarized as follows.

Optimizer	# iterations	reduction	Time
Gauss–Newton	[11, $10^\dagger$, $10^\dagger$, $10^\dagger$, $10^\dagger$]	10.63%	915 *s*
ℓ-BFGS	[11, 9, $10^\dagger$, $10^\dagger$, $10^\dagger$]	10.81%	1130 *s*
Trust-Region	[11, $10^\dagger$, 8, $10^\dagger$, $10^\dagger$]	16.52%	2031 *s*

Example 9.10 (MLIR, 3D Brain: Gauss–Newton, ℓ-BFGS, Trust-Region) These experiments are based on 3D brain data provided by Ron Kikinis, Surgical Planning Laboratory, Brigham and Women's Hospital, Boston. The data of size $m = [128, 64, 128]$ is given on the domain $\Omega = (0, 20) \times (0, 10) \times (0, 20)$ and is represented on levels $\ell = 3 : 6$. As for the previous example, a 3D rendering of the sampled template and reference as well as a 2D visualization based on orthogonal views of the data are shown in Figure 9.11: the second row visualizes sagittal views of the template, reference, and the difference and the last two rows visualize the results of registrations based on Gauss–Newton (G), ℓ-BFGS, and trust-region (TR) approaches. The experiments are based on `SSD`, `affine3D` preregistration, `mfElastic/staggered`, and $[\alpha, \mu, \lambda] = [1000, 1, 0]$.

The number of iterations († indicates stopping due to maximum number of iterations), the reduction (cf. (9.3)), and the computation times are summarized as follows.

Optimizer	# iterations	reduction	Time
Gauss–Newton	[2, $10^\dagger$, $10^\dagger$, $10^\dagger$, $10^\dagger$]	42.91%	975 s
ℓ-BFGS	[2, 8, $10^\dagger$, $10^\dagger$, $10^\dagger$]	46.58%	1322 s
Trust-Region	[11, 6, 6, $10^\dagger$, 2]	42.96%	1969 s

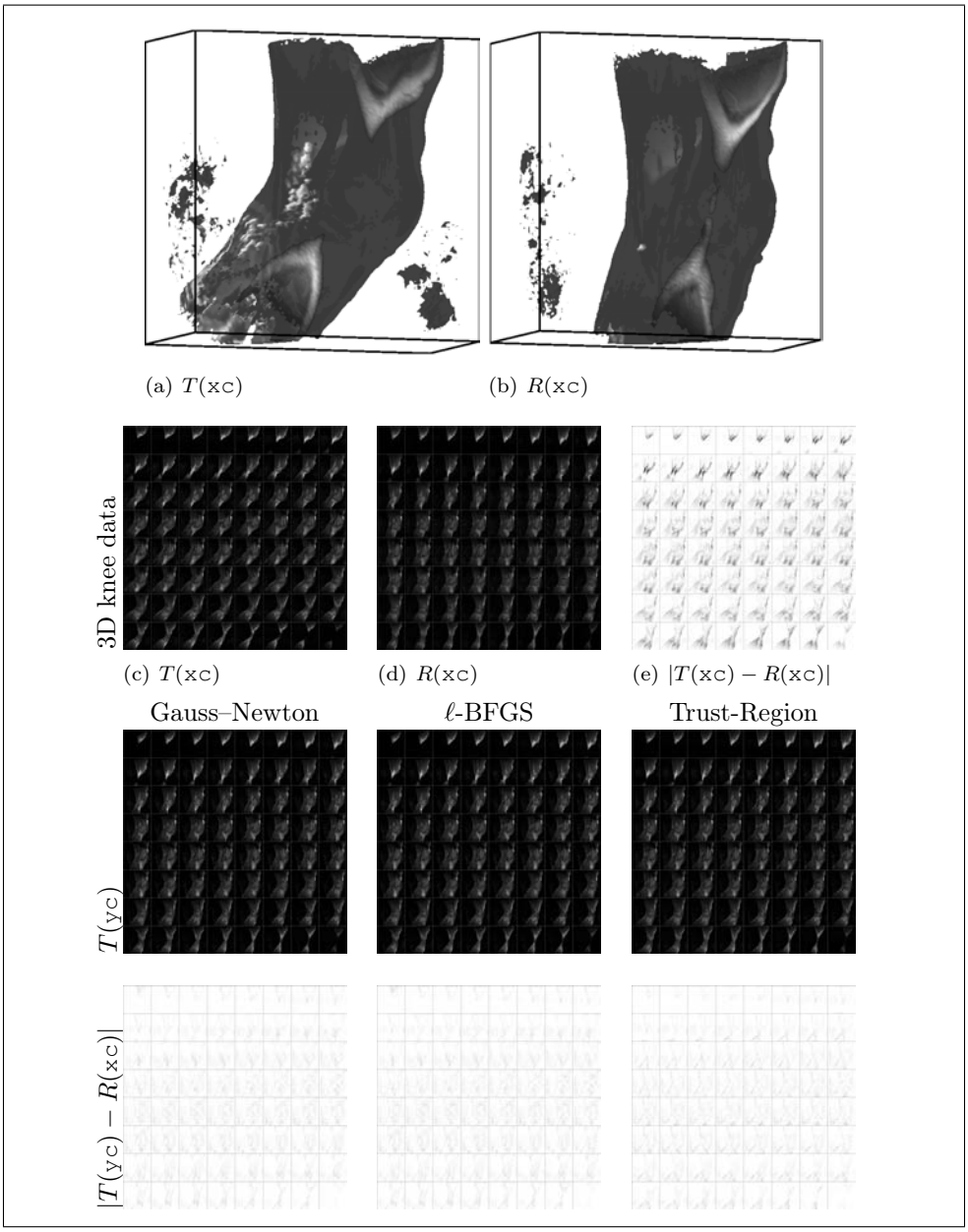


Figure 9.10: *MLIR results for 3D knee data (cf. Example 9.9) using the SSD distance combined with an elastic regularizer; 3D (top row) and 2D (second row) visualizations of the template and reference images. Transformed template (third row) and differences to the reference image (bottom row) are shown for optimization results obtained with Gauss–Newton (left), ℓ-BFGS (middle), and trust-region (right) approaches.*

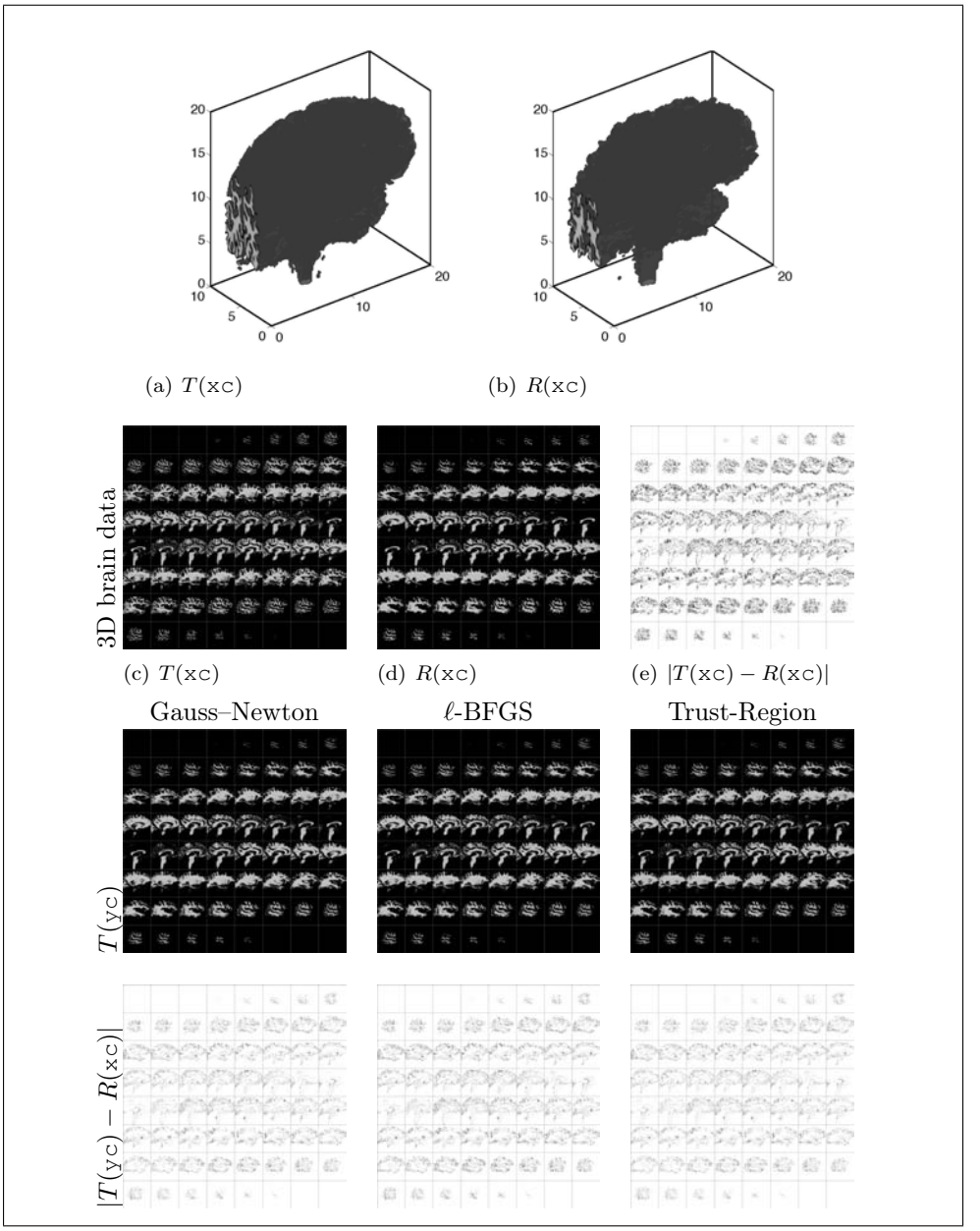
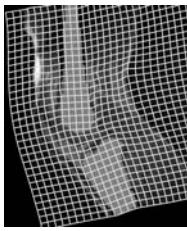


Figure 9.11: *MLIR results for 3D brain data (cf. Example 9.10) using the SSD distance combined with a curvature regularizer; 3D (top row) and 2D (second row) visualizations of the template and reference images. Transformed template (third row) and differences to the reference image (bottom row) are shown for optimization results obtained with Gauss–Newton (left), ℓ-BFGS (middle), and trust-region (right); see Example 9.8 for details.*



9.8 Summarizing the Nonparametric Image Registration

This chapter combines the modules prepared in the previous chapters. It is discussed how the joint functional $\mathcal{J}$ in (9.1) can be discretized and how different computational grids can be treated in a unified fashion. Numerical optimization schemes such as Gauss–Newton, ℓ-BFGS, and trust-region are discussed. Moreover, the optimization strategy is embedded into a multiscale and multilevel frameworks, adding additional regularity and reducing the computational costs. Finally, a variety of numerical examples is presented.

FAIR 22: Driver for Multilevel Image Registration

This driver initializes the data (including the visualization), parameterizes the interpolation, parametric preregistration, distance, and regularization, and calls the registration scheme `MLIR`.

This file is `E9_HNSP_MLIR__SSD_mbCurv.m`

```
%%%%%%%%%
clear, close all, help(mfilename);

% load some data, set viewer, interpolator, transformation, distance
setupHNSPData
inter('reset','inter','splineInter2D','regularizer','moments','theta',1e-2);
distance('reset','distance','SSD');
trafo('reset','trafo','affine2D');
regularizer('reset','regularizer','mbCurvature','alpha',1e1);

% run optimization
% yc =  MLIR(MLdata,'maxLevel',7);
```

9.9 FAIR Tutorials on Image Registration

FAIR contains the tutorial `BigTutorialNPIR` which summarizes a number of smaller tutorials providing examples for a variety of registration problems.

BigTutorialNPIR

`E9_Hands_NPIRmb_GN`:	NPIR, hands, elastic, matrix-based
`E9_Hands_NPIRmf_GN`:	NPIR, hands, elastic, matrix-free
`E9_Hands_NPIRmf_TR_nopre`:	NPIR, hands, trust-region, plain
`E9_Hands_NPIRmf_TR_pcg`:	NPIR, hands, trust-region, multigrid preconditioned
`E9_PETCT_MLIR_NGF_mbElas`:	MLIR, PETCT, NGF, elastic, matrix-based
`E9_HNSP_MLIR_TR`:	MLIR, HNSP, elastic, matrix-free, trust-region
`E9_Hands_MLIR_SSD_mbElas`:	MLIR, hands, elastic, matrix-based
`E9_Hands_MLIR_SSD_mfElas`:	MLIR, hands, elastic, matrix-free
`E9_Hands_MLIR_SSD_mbCurv`:	MLIR, hands, curvature, matrix-based
`E9_Hands_MLIR_SSD_mfCurv`:	MLIR, hands, curvature, matrix-free
`E9_HNSP_MLIR_SSD_mbElas`:	MLIR, HNSP, elastic, matrix-based
`E9_HNSP_MLIR_SSD_mfElas`:	MLIR, HNSP, elastic, matrix-free
`E9_HNSP_MLIR_SSD_mbCurv`:	MLIR, HNSP, curvature, matrix-based
`E9_HNSP_MLIR_SSD_mfCurv`:	MLIR, HNSP, curvature, matrix-free

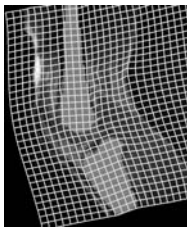
9.10 Exercises

Exercise 9.1
Repeat the multilevel registration of the hand example. Find a regularization parameter α as small as possible but such that the transformation is still visually one-to-one. Advise a method for choosing a reasonable regularization parameter.

Exercise 9.2
Compare the results obtained with the diffusion, elastic, and curvature regularizers.

Exercise 9.3
Compare the timings obtained for 2D and 3D examples using different data sizes and the matrix-based and matrix-free implementations.

Exercise 9.4
Compare and discuss the results obtained using different optimization methods.



Chapter 10
Outlook

This chapter presents a summary of the topics discussed in this book and looks at the various topics which have not been addressed.

10.1 Summary

This book provides a unified mathematical framework for image registration and presents some MATLAB tools for exploring this framework. The starting point is the variational formulation

$$\mathcal{J}[y] = \mathcal{D}[\mathcal{T}[y], \mathcal{R}] + \mathcal{S}[y - y^{\text{ref}}] = \min, \tag{10.1}$$

where the joined functional is composed of a data fitting term $\mathcal{D}$ and regularization $\mathcal{S}$; see Chapters 7 and 8, respectively.

This book provides a variety of options but also aims to provide a framework for the integration of newly designed modules.

10.1.1 Registration Modules

Special attention has been given to a modularization of building blocks and their reasonable implementations. The key building blocks are

- **image models:** Chapter 3 provides various interpolation techniques, which are used to represent images on different scales and resolutions and also to define transformed images for arbitrary transformations;
- **transformation models:** Chapter 4 provides a variety of parameterized transformations $y = y(w, x)$ (translations, rotations, rigid, affine linear, spline based), which serve as a basis for PIR; Chapters 8 and 9 provide the tools for nonparametric registration;
- **distance measures:** Chapter 5 introduces landmark-based distance measures, and volumetric distance measures are provided in Chapter 7;

- **regularization:** since registration is an ill-posed problem, regularization becomes a key component; regularization techniques are discussed in various places in this book: for interpolation, landmark registration, high-dimensional parametric registration, and nonparametric registration; note that multiscale and multilevel strategies may also be viewed as regularization techniques;
- **discretization:** this book focuses on a discretize-then-optimize approach; the idea is to create a coarse to fine discretization of the registration problem

$$J^h(y^h) = D^h(T(y^h), R^h) + S(y^h - y^{\text{ref},h}) = \min \tag{10.2}$$

  and to use the numerical solution of a coarser discretization as a starting point for a finer discretization;
- **optimization:** in this book, registration is phrased as an optimization problem; thus, numerical optimization is the final important ingredient.

10.1.2 Multiscale and Multilevel Approaches

The above ingredients are to be combined in a multilevel and/or multiscale approaches; see Sections 6.6, 9.3, and 9.4. A much deeper discussion on scale-space ideas can be found, e.g., in [106, 154, 179, 191, 192]. Note that multilevel and/or multiscale approaches can become crucial for registration, but some applications, such as microscopy images displaying clouds of cells, may not allow this representation.

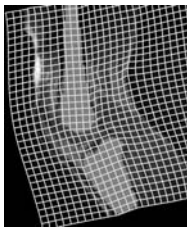
10.1.3 Optimization

As the registration problem is attacked from an optimization perspective, numerical optimization becomes a major tool. This book focuses on the discretize-then-optimize approach. The variational formulation (10.1) is consistently discretized. For a concrete discretization, numerical optimization techniques are used to compute a numerical optimizer of the discretized objective function J. Emphasis has been given to provide the analytical gradient of the discretized objective function. All modules discussed and provided in this book thus return not only a function value but also their analytic derivative. Moreover, a simple mechanism for testing derivative implementations has been provided; cf. Section 3.5.

In principle, any numerical optimization techniques could be used. However, FAIR has a bias towards quasi-Newton techniques and in particular Gauss–Newton-type optimization; see Section 9.6 for alternatives.

10.2 Topics Not Covered

This book provides a very focused and narrow view to the registration problem. Emphasis is given to the software engineering and numerical perspective. The theoretical and application perspectives have certainly been neglected. The following sections summarize some of the open problems and provide some links to the literature.



10.2.1 Theoretical Foundations

Existence of solutions for the registration problems are discussed, for example, in [206, 87, 136, 99, 139, 134, 96, 83] and the references therein. More information about splines and interpolation can be found, e.g., in [97, 75, 203, 153, 138, 149, 190].

10.2.2 Choosing the Building Blocks

The question of how to assemble the objective function from the proposed modules for a particular application is not addressed. The reason is that there seems to be no automatic way to make a proper choice, and the selection pretty much depends on the application. For the interpolation it seems to be a matter of taste, and the differences between using spline-based or linear interpolation techniques are rather small. However, already the distance measure can make a big difference. General advice is to start with a rather simple but robust approach such as sum of squared differences (SSD) or normalized cross-correlation (NCC) and, if this fails, to continue with a more complex measure such as normalized gradient field (NGF) or mutual information (MI). Note that the simple measures are designed essentially for monomodal images and are generally easier to optimize, while the more complex measures are designed for multimodal problems and generally introduce more local minima to the registration problem [171]. The problem is even more severe for the regularization. Recent work aims to also optimize with respect to the regularization. Note, however, that the so-called objective function is always subjective.

More transformation options like polyrigid, polyaffine, or piecewise affine transformation models are discussed, for example, in [198, 61]. Alternative distance measures are presented in [74, 68, 72, 141, 52, 170, 173, 135, 94, 166, 126] and alternative regularization is discussed in [151, 144]. Initialized by the pioneering work of [80, 67, 73, 86], recent trends are to improve the modeling/regularization and to integrate biomechanical models; see, e.g., [159, 100, 204, 158, 182, 168].

10.2.3 Parameter Tuning

A number of explicit and implicit parameters need to be chosen. Already the interpolation module provides a variety of options. The linear interpolation is straightforward, easy to implement, and quickly evaluated, but it is not everywhere differentiable. The spline interpolation is differentiable but introduces ringing artifacts which may yield local minima. A multiscale approach, as discussed in Chapter 3, moderates these problems but requires the choice of a regularizer (Tychonoff, Tychonoff–Phillips, moments, etc.) and a regularization parameter θ. Moreover, the evaluation requires the computation of the spline coefficients and in addition is more expensive than the linear interpolation. More information on interpolation issues can be found, for example, in [53, 199, 153, 64, 150, 190, 138, 54, 82].

Also the scale-space technique and/or coarse to fine technique require parameterization. For the multiscale techniques, this seems to be a rather stable process. The multilevel approach, however, demands some sensitivity. If the coarse representation of the problems does not provide enough details, the scheme may end up in a

local minima and may not recover. Unfortunately, there seems to be no automatic way to quantify sufficiency. Interesting aspects are discussed, e.g., in [106, 128].

Another tuning parameter is the regularization parameter α itself. Starting with a strong regularization, the optimization problem is assumed to be smoother, and computing a minimizer should be fast. Relaxing the discretization parameter can be done in a way similar to the multiscale approach. For details on this so-called continuation approach see, e.g., [55, 56, 57, 118].

10.2.4 Validation

A central, though not yet answered, question is how to validate the registration results. Some pioneering steps are presented in [60, 58, 91, 105, 130, 181, 127]. An incomplete list of validation projects is given below.

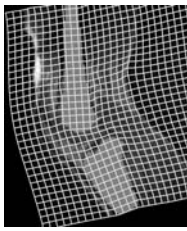
- Nonrigid Image Registration Evaluation Project (NIREP)
  http://www.nirep.org/
- Retrospective Image Registration Evaluation Project
  http://www.insight-journal.org/rire/
- ROBIN Competition
  http://robin.inrialpes.fr/overview.php
- Validation and Evaluation in Medical Imaging Processing (VMIP)
  http://idm.univ-rennes1.fr/VMIP/index.html

10.2.5 Consistency

In the optimization framework presented in this book, the quantification of a reasonable or plausible transformation is formalized using a regularizer. Dependent on the application, one may aim to archive a symmetric result when interchanging the reference and template images. An approach to force a so-called consistent transformation has been proposed in [85].

10.2.6 Diffeomorphisms

Many imaging groups consider diffeomorphisms as a natural manifold for registration problems [98, 196, 177, 82, 85, 125, 161, 84, 143, 197, 71, 178, 66, 62, 201]. A diffeomorphism is a mapping $y : \Omega \to \Sigma$ such that y and y^{-1} are differentiable. Note that the existence of y^{-1} guarantees that the transformation is one-to-one. Moreover, diffeomorphisms build a group which can be an important feature for atlas generation. However, there are medical applications, such as bending of a joint or sliding of an organ, where the transformation may not be as smooth. On the other hand, a diffeomorphism may still give an unreasonable transformation. A trivial example is $y(x) = 0.5^n x$, where n is chosen such that the size of the transformed domain has no physical meaning. More examples are illustrated in Section 10.2.9.



10.2.7 (Optical) Flow Techniques

Optical flow techniques model the image as a function in space and time, $\mathcal{T} : \Omega \times \mathbb{R} \to \mathbb{R}$, and are based on the assumption that a particle located at $x(t)$ at time t does not change its intensity. Thus, $\mathcal{T}(x(t), t) = \text{const}$, and with the velocity $v = \dot{x}$,

$$0 = dt\mathcal{T}(x(t), t) = \partial_t \mathcal{T} + \nabla_x \mathcal{T}\ v,$$

which is the optical flow constraint [140, 70]. In order to handle illumination variation, equality is replaced by minimization of a distance, and for well-posedness, regularization is added [76, 88, 81]. A typical formulation thus reads

$$\mathcal{J}^{\mathrm{OF}}[v] = 0.5\ \|\partial_t \mathcal{T} + \nabla_x \mathcal{T}\ v\|^2 + \mathcal{S}[v].$$

With the interpretations $\partial_t \mathcal{T} \approx \mathcal{T} - \mathcal{R}$ and $\dot{x} \approx dy$, the above might be viewed as linearized version of the SSD distance measure; cf. Section 7.1.

Flow techniques are also used to "drive" the deformable template towards the reference. Typical examples are the original formulation of Thirion's demons approach [193] (see also [194, 167, 102]) and Christensen's fluid approach [86]. Keeling and Ring proposed a flow approach with maximal rigidity [146].

10.2.8 Stochastical Approaches

Image registration can also be phrased as a regularized maximum likelihood problem [110, 63, 212, 107]. This involves finding parameters y for a transformation model that maximize the likelihood of the observed images I. In order to achieve this, the problem can be considered within a Bayesian context and rephrased as follows. The posterior probability of the parameters, given the images, is proportional to the likelihood of observing the data, given the parameters, times the prior probability of the parameters,

$$P(y|I) \propto P(I|y)P(y).$$

Choosing the right priors and taking the logarithm, the likelihoods can be interpreted as distance measure and regularization. Note that important concepts such as scale or multilevel representations are not easy to integrate into this statistical framework.

10.2.9 Constrained Image Registration

Another avenue on the registration map is the integration of application-relevant information. Typical examples are the integration of landmarks [143, 129, 104, 117], the maintenance of rigid structures [155, 177, 157, 187, 186, 163, 116], volume [174, 119, 118, 120, 121], and mass preservation [125, 211]. An example is given by the flip-book images in this book, which visualizes a registration of a human knee under rigidity constraints on femur and tibia.

10.2.10 Efficiency

For some applications, fast results are critical and many attempts have been made to speed up the computations [93, 79, 92, 210, 205, 131, 204, 133, 208, 148, 108]. Recent trends involve the usage of GPUs [189, 183, 164, 152, 147], sparse grid computations [114, 115], and the computation on surfaces only [142, 65, 209].

Bibliography

For a general overview on the literature see Section 1.4 (references 1–51).

[52] K. A. Abram, *Registration of images with dissimilar contrast using a hybrid method employing correlation and mutual information*, Technical Report, Dept. of Computer Science, Dartmouth College, Hannover, 2000. 167

[53] A. Aldroubi, M. Eden, and M. Unser, *Discrete spline filters for multiresolutions and wavelets of ℓ_2^**, SIAM Journal on Mathematical Analysis **25** (1994), 1412–1432. 27, 167

[54] A. Aldroubi and K. Gröchening, *Nonuniform sampling and reconstruction in shift-invariant spaces*, SIAM Review **43** (2001), 585–620. 167

[55] J. C. Alexander and J. A. Yorke, *The homotopy continuation method: Numerically implementable topological procedures*, Transactions of American Mathematical Society **242** (1978), 271–284. 168

[56] E. Allgower and K. Georg, *Numerical continuation methods*, Springer, Berlin, 1990. 168

[57] E. Allgower and K. Georg, *Numerical path following*, Handbook of Numerical Analysis, vol. 5, North–Holland, Amsterdam, 1997, pp. 3–207. 168

[58] N. M. Alpert, D. Berdichevsky, Z. Levin, E. D. Morris, and A. J. Fischman, *Improved methods for image registration*, NeuroImage **3** (1996), 10–18. 168

[59] Y. Amit, *A nonlinear variational problem for image matching*, SIAM Journal on Scientific Computing **15** (1994), 207–224. 17, 18

[60] S. Arndt, R. Rajarethinam, T. Cizadlo, D. O'Leary, J. Downhill, and N. C. Andreasen, *Landmark-based registration and measurement of magnetic resonance images: A reliability study*, Neuroimaging **67** (1996), 145–154. 168

[61] V. Arsigny, X. Pennec, and N. Ayache, *Polyrigid and polyaffine transformations: A novel geometrical tool to deal with non-rigid deformations—application to the registration of histological slices*, Medical Image Analysis **9** (2005), 507–523. 167

[62] J. Ashburner, *A fast diffeomorphic image registration algorithm*, NeuroImage **38** (2007), 95–113. 168

[63] J. Ashburner, J. Andersson, and K. J. Friston, *High-dimensional nonlinear image registration using symmetric priors*, NeuroImage **9** (1999), 619–628. 169

[64] J. Ashburner and K. J. Friston, *Nonlinear spatial normalization using basis functions*, Human Brain Mapping **7** (1999), 254–266. 167

[65] M. A. Audette, F. P. Ferrie, and T. M. Peters, *An algorithmic overview of surface registration techniques for medical imaging*, Medical Image Analysis **4** (2000), 201–217. 170

[66] B. B. Avants, P. T. Schoenemann, and J. C. Gee, *Lagrangian frame diffeomorphic image registration: Morphometric comparison of human and chimpanzee cortex*, Medical Image Analysis **10** (2006), 397–412. 168

[67] R. Bajcsy and S. Kovačič, *Multiresolution elastic matching*, Computer Vision, Graphics and Image Processing **46** (1989), 1–21. 167

[68] G. Barequet and M. Sharir, *Partial surface and volume matching in three dimensions*, IEEE Transactions on Pattern Analysis and Machine Intelligence **19** (1997), 929–948. 167

[69] R. Barrett, M. Berry, T. F. Chan, J. W. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. van der Vorst, *Templates for the solution of linear systems: Building blocks for iterative methods*, 2nd ed., SIAM, Philadelphia, 1994. 78, 134, 158

[70] J. L. Barron, D. J. Fleet, and S. S. Beauchemin, *Performance of optical flow techniques*, Technical Report no. 229, Dept. of Computer Science, University of Western Ontario, 1992. 169

[71] F. Beg, M. Miller, A. Trouvé, and L. Younes, *Computing large deformation metric mappings via geodesic flows of diffeomorphisms*, International Journal of Computer Vision **61** (2005), 139–157. 168

[72] J. L. Boes and C. R. Meyer, *Multi-variate mutual information for registration*, Proceedings of Medical Image Computing and Computer-Assisted Intervention (C. Taylor and A. Colchester, eds.), Springer, 1999, pp. 606–612. 167

[73] F. L. Bookstein, *Principal warps: Thin-plate splines and the decomposition of deformations*, IEEE Transactions on Pattern Analysis and Machine Intelligence **11** (1989), 567–585. 61, 123, 167

[74] ———, *Four metrics for image variation*, Proceedings of the 11th Int. Conf. on Information Processing in Medical Imaging Wiley, New York, 1991, pp. 227–240. 167

[75] C. De Boor, *A practical guide to splines*, Springer, New York, 1978. 167

[76] A. Borzì, K. Ito, and K. Kunisch, *Optimal control formulation for determining optical flow*, SIAM Journal on Scientific Computing **24** (2002), 818–847. 169

[77] J. W. Brewer, *Kronecker products and matrix calculus in system theory*, IEEE Transactions on Circuits and Systems **25** (1978), 772–780. 30

[78] W. L. Briggs, V. E. Henson, and S. F. McCormick, *A multigrid tutorial*, 2nd ed., SIAM, Philadelphia, 2000. 134

[79] M. Bro-Nielsen and C. Gramkow, *Fast fluid registration of medical images*, Lecture Notes in Computer Science, vol. 1131, Sprinter, Berlin, 1996, pp. 267–276. 170

[80] C. Broit, *Optimal registration of deformed images*, Ph.D. thesis, Computer and Information Science, University of Pensylvania, USA, 1981. 121, 167

[81] T. Brox, A. Bruhn, N. Papenberg, and J. Weickert, *High accuracy optical flow estimation based on a theory for warping*, Proceedings of the 8th Int. Conf. on Computer Vision—ECCV 2004, Part IV (T. Pajdla and J. Matas,

eds.), Lecture Notes in Computer Science vol. 3024, Springer, Berlin, 2004, pp. 25–36. 169

[82] V. Camion and L. Younes, *Geodesic interpolating splines*, Proceedings of EMMCVPR '01 (M. Figueiredo, J. Zerubia, and A. K. Jain, eds.), Lecture Notes in Computer Science vol. 2134, Springer, Berlin, 2001, pp. 513–527. 167, 168

[83] T. F. Chan and J. Shen, *Image processing and analysis: Variational, PDE, wavelet, and stochastic methods*, SIAM, Philadelphia, 2005. 167

[84] C. Chefd'Hotel, G. Hermosillo, and O. D. Faugeras, *Flows of diffeomorphisms for multimodal image registration*, Proceedings of ISBI, IEEE, 2002, pp. 753–756. 168

[85] G. E. Christensen and H. J. Johnson, *Consistent image registration*, IEEE Transactions on Medical Imaging **20** (2001), 568–582. 168

[86] G. E. Christensen, *Deformable shape models for anatomy*, Ph.D. thesis, Sever Institute of Technology, Washington University, USA, 1994. 167, 169

[87] U. Clarenz, M. Droske, and M. Rumpf, *Towards fast non–rigid registration*, Inverse Problems, Image Analysis and Medical Imaging, AMS Special Session Interaction of Inverse Problems and Image Analysis, vol. 313, AMS, 2002, pp. 67–84. 167

[88] U. Clarenz, S. Henn, M. Rumpf, and K. Witsch, *Relations between optimization and gradient flow methods with applications to image registration*, Proceedings of the 18th GAMM Seminar Leipzig on Multigrid and Related Methods for Optimization Problems, 2002, pp. 11–30. 169

[89] C. A. Cocosco, V. Kollokian, R. K.-S. Kwan, and A. C. Evans, *BrainWeb: Simulated Brain Database* 2006, available at http://www.bic.mni.mcgill.ca/brainweb/. 18

[90] A. Collignon, A. Vandermeulen, P. Suetens, and G. Marchal, *Automated multi-modality medical image registration based on information theory*, Computational Imaging and Vision vol. 3, 1995, pp. 263–274. 99

[91] D. L. Collins and A. C. Evans, *Animal: Validation and applications of nonlinear registration-based segmentation*, International Journal of Pattern Recognition and Artificial Intelligence **11** (1997), 1271–1294. 168

[92] S. Cotin, H. Delingette, and N. Ayache, *Real time volumetric deformable models for surgery simulation*, Visualization in Biomedical Computing (K. H. Höhne and R. Kikinis, eds.), Springer, Berlin 1996, pp. 535–540. 170

[93] R. Creutzburg, V. G. Labunets, and E. V. Labunets, *Fast spectral algorithms for invariant pattern recognition and image matching*, Proceedings of the Int. Conf. on Computer Assisted Image Processing, Academie-Verlag, Berliln, 1991, pp. 85–95. 170

[94] J. Dauguet, J. F. Mangin, T. Delzescaux, and V. Frouin, *Robust inter-slice intensity normalization using histogram scale-space analysis*, Proceedings of MICCAI, Lecture Notes in Computer Science, vol. 2879, Springer, Berlin, 2003, pp. 41–49. 167

[95] J. E. Dennis, Jr. and H. F. Walker, *Convergence theorems for least-change secant update methods*, SIAM Journal on Numerical Analysis **18** (1981), 949–987. 153

[96] M. Droske and M. Rumpf, *A variational approach to non-rigid morphological registration*, SIAM Journal on Applied Mathematics **64** (2004), 668–687. 12, 167

[97] J. Duchon, *Interpolation des fonctions de deux variables suivant le principle de la flexion des plaques minces*, RAIRO Analyse Numérique **10** (1976), 5–12. 33, 61, 123, 167

[98] P. Dupuis, U. Grenander, and M. I. Miller, *Variational problems on flows of diffeomorphisms for image matching*, Quarterly of Applied Mathematics **56** (1998), 587–600. 168

[99] O. Faugeras, G. Aubert, and P. Kornprobst, *Mathematical problems in image processing*, Springer, New York, 2002. 167

[100] M. Ferrant, S. K. Warfield, C. R. G. Guttmann, R. V. Mulkern, F. A. Jolesz, and R. Kikinis, 3*D image matching using a finite element based elastic deformation model*, Medical Image Computing and Computer Assisted Intervention (C. Taylor and A. Colchester, eds.), Springer, 1999, pp. 202–209. 167

[101] B. Fischer and J. Modersitzki, *Fast curvature based registration of MR-mammography images*, Bildverarbeitung für die Medizin (M. Meiler et al., eds.), Springer, Berlin, 2002, pp. 139–143. 122

[102] ———, *Fast diffusion registration*, AMS Contemporary Mathematics, Inverse Problems, Image Analysis, and Medical Imaging **313** (2002), 117–129. 121, 169

[103] ———, *Curvature based image registration*, Journal of Mathematical Imaging and Vision **18** (2003), 81–85. 122, 124

[104] ———, *Intensity based image registration with a guaranteed one-to-one point match*, Methods of Information in Medicine **43** (2004), 327–330. 169

[105] J. M. Fitzpatrick, J. B. West, and J. C. R. Maurer, *Predicting error in rigid-body, point-based registration*, IEEE Transactions on Medical Imaging **17** (1998), 694–702. 168

[106] L. Florack, B. Romeny, J. Koenderink, and M. Viergever, *Scale and the differential structure of images*, Image and Vision Computing **10** (1992), 376–388. 166, 168

[107] K. J. Friston, J. Ashburner, S. J. Kiebel, T. E. Nichols, and W. D. Penny (eds.), *Statistical parametric mapping: The analysis of functional brain images*, Academic Press, London, 2007. 169

[108] C. Frohn-Schauf, S. Henn, and K. Witsch, *Multigrid based total variation image registration*, Computing and Visualization in Science **11** (2008), 101–113. 170

[109] P. E. Gill, W. Murray, and M. H. Wright, *Practical optimization*, Academic Press, London, 1981. 78

[110] C. A. Glasbey and K. V. Mardia, *A review of image warping methods*, Journal of Applied Statistics **25** (1998), 155–171. 169

[111] G. Golub, M. Heath, and G. Wahba, *Generalized cross-validation as a method for choosing a good ridge parameter*, Technometrics **21** (1979), 215–223. 33

[112] G. H. Golub and C. F. van Loan, *Matrix computations*, second ed., The Johns Hopkins University Press, Baltimore, 1989. 76

[113] M. E. Gurtin, *An introduction to continuum mechanics*, Academic Press, Or-

lando, 1981. 121
[114] E. Haber, S. Heldmann, and J. Modersitzki, *An octree method for parametric image registration*, SIAM Journal on Scientific Computing **29** (2007), 2008–2023. 170
[115] ———, *Adaptive mesh refinement for nonparametric image registration*, SIAM Journal on Scientific Computing **30** (2008), 3012–3027. 170
[116] ———, *A framework for image-based constrained registration with an application to local rigidity*, LAA, **431** (2009), 459–470. 169
[117] ———, *A scale-space approach to landmark constrained image registration*, Proceedings of the Second International Conference on Scale Space Methods and Variational Methods in Computer Vision (SSVM), Lecture Notes in Computer Science, Springer, 2009, pp. 1–12. 169
[118] E. Haber and J. Modersitzki, *Numerical methods for volume preserving image registration*, Inverse Problems **20** (2004), 1621–1638. 168, 169
[119] ———, *Volume preserving image registration*, Medical Image Computing and Computer-Assisted Intervention—MICCAI 2004 (C. Barillot, D. R. Haynor, and P. Hellier, eds.), Lecture Notes in Computer Science vol. 3216, Springer, 2004, pp. 591–598. 169
[120] ———, *A scale space method for volume preserving image registration*, Proceedings of the 5th International Conference on Scale Space and PDE Methods in Computer Vision, Springer 2005, pp. 1–8. 169
[121] ———, *Image registration with a guaranteed displacement regularity*, International Journal of Computer Vision **71** (2007). 169
[122] ———, *A multilevel method for image registration*, SIAM Journal on Scientific Computing **27** (2006), 1594–1607. 128, 134
[123] ———, *Intensity gradient based registration and fusion of multi-modal images*, Methods of Information in Medicine **46** (2007), 292–299. 107
[124] J. Hadamard, *Sur les problèmes aux dérivées partielles et leur signification physique*, Princeton University Bulletin **13** (1902), 49–52. 117
[125] S. Haker and A. Tannenbaum, *Optimal transport and image warping*, MICCAI, Springer 2001, 120–127. 168, 169
[126] S. Heldmann, *Non-linear registration based on mutual information*, Ph.D. thesis, University of Lübeck, Germany, 2006. 101, 105, 167
[127] P. Hellier, C. Barillot, I. Corouge, B. Gibaud, G. L. Goualher, D. Collins, A. Evans, G. Malandain, N. Ayache, G. Christensen, and H. Johnson, *Retrospective evaluation of inter-subject brain registration*, IEEE Transactions on Medical Imaging **20**, 1120–1130. 168
[128] P. Hellier, C. Barillot, E. Mémin, and P. Pérez, *Hierarchical estimation of a dense deformation field for 3-d robust registration*, IEEE Transaction on Medical Imaging **20** (388-402), 388–402. 168
[129] P. Hellier and C. Barillot, *Coupling dense and landmark-based approaches for non rigid registration*, IEEE Transactions on Medical Imaging **22** (2003), 217–227. 169
[130] P. Hellier, C. Barillot, I. Corouge, B. Gibaud, G. Le Goualher, D. L. Collins, A. C. Evans, G. Malandain, and N. Ayache, *Retrospective evaluation of inter-subject brain registration*, MICCAI '01: Proceedings of the 4th International

Conference on Medical Image Computing and Computer-Assisted Intervention (London, UK), Springer, 2001, pp. 258–265. 168

[131] P. Hellier, C. Barillot, E. Mémin, and P. Pérez, *Medical image registration with robust multigrid techniques*, Medical Image Computing and Computer-Assisted Intervention (C. Taylor and A. Colchester, eds.), Springer, 1999, pp. 680–757. 170

[132] S. Henn, *A multigrid method for a fourth-order diffusion equation with application to image processing*, SIAM Journal on Scientific Computing **27** (2005), 831–849. 124, 130

[133] S. Henn and K. Witsch, *A multigrid approach for minimizing a nonlinear functional for digital image matching*, Computing **64** (2000), 339–348. 128, 170

[134] ———, *Multimodal image registration using a variational approach*, SIAM Journal on Scientific Computing **25** (2003), 1429–1447. 167

[135] G. Hermosillo, *Variational methods for multimodal image matching*, Ph.D. thesis, Université de Nice, France, 2002. 101, 167

[136] G. Hermosillo, C. Chefd'Hotel, and O. Faugeras, *Variational methods for multimodal image matching*, International Journal of Computer Vision **50** (2002), 329–343. 167

[137] M. R. Hestenes and E. Stiefel, *Methods of conjugate gradients for solving linear systems*, Journal of Research of the National Bureau of Standards **49** (1952), 409–436. 78, 158

[138] W. Hinterberger and O. Scherzer, *Models for image interpolation based on the optical flow*, Computing **66** (2000), 1–20. 167

[139] W. Hinterberger, O. Scherzer, C. Schnörr, and J. Weickert, *Analysis of optical flow models in the framework of calculus of variations*, Numerical Functional Analysis and Optimization **23** (2002), 69–82. 167

[140] B. K. P. Horn and B. G. Schunck, *Determining optical flow*, Artificial Intelligence **17** (1981), 185–204. 121, 169

[141] H.-W. Hseu, A. Bhalerao, and R. Wilson, *Image matching based on the co-occurence matrix*, Technical Report CS, RR-358, Department of Computer Science, University of Warwick, Coventry, UK 1999. 167

[142] A. Johnson and M. Hebert, *Surface registration by matching oriented points*, International Conference on Recent Advances in 3-D Digital Imaging and Modeling, 1997, pp. 121–128. 170

[143] H. J. Johnson and G. E. Christensen, *Consistent landmark and intensity-based image registration*, IEEE Transactions on Medical Imaging **21** (2002), 450–461. 168, 169

[144] S. Kabus, A. Franz, and B. Fischer, *Variational image registration with local properties*, Biomedical Image Registration: Third International Workshop, WBIR 2006 (F. A. Gerritsen J. P. W. Pluim, B. Likar, ed.), Lecture Notes in Computer Science, Springer, 2006, pp. 92–100. 167

[145] B. Kaltenbacher, *On Broyden's method for regularization of nonlinear ill-posed problems*, Numerical Functional Analysis and Optimization **19** (1998), 807–833. 153

[146] S. L. Keeling and W. Ring, *Medical image registration and interpolation by op-*

tical flow with maximal rigidity, Journal of Mathematical Imaging and Vision **23** (2005), 47–65. 169

[147] A. Köhn, J. Drexl, F. Ritter, M. König, and H. O. Peitgen, *GPU accelerated image registration in two and three dimensions*, Bildverarbeitung für die Medizin (T. Tolxdoff et al., eds.), Springer, 2008, pp. 261–265. 170

[148] H. Köstler, M. Stürmer, and U. Rüde, *A fast full multigrid solver for applications in image processing*, Numerical Linear Algebra with Applications **15** (2008), 187–200. 170

[149] J. Kybic and M. Unser, *Multidimensional elastic registration of images using splines*, Proceedings of ICIP, Vancouver, vol. II, IEEE 2000, pp. 455–458. 167

[150] T. Lehmann, C. Gönner, and K. Spitzer, *Survey: Interpolation methods in medical image processing*, IEEE Transactions on Medical Imaging **18** (1999), 1049–1075. 167

[151] H. Lester, S. R. Arridge, and K. M. Jansons, *Local deformation metrics and nonlinear registration using a fluid model with variable viscosity*, Proceedings of Medical Image Understanding and Analysis (MIUA '98), 1998. 167

[152] B. Li, A. A. Young, and B. R. Cowan, *GPU accelerated non-rigid registration for the evaluation of cardiac function*, in Metaxas et al. [160], pp. 880–887. 170

[153] W. A. Light, *Variational methods for interpolation, particularly by radial basis functions*, Numerical Analysis (London 1995), (D. F. Griffiths and G. A. Watson, eds.), Longmans, 1996, pp. 94–106. 167

[154] T. Lindeberg, *Scale-space theory in computer vision*, Kluwer Academic Publishers, Dordrecht, 1994. 166

[155] J. A. Little, D. L. G. Hill, and D. J. Hawkes, *Deformations incorporating rigid structures*, Computer Vision and Image Understanding **66** (1997), 223–232. 169

[156] D. C. Liu and J. Nocedal, *On the limited memory BFGS method for large scale optimization*, Mathematical Programming **45** (1989), 503–528. 154

[157] D. Loeckx, F. Maes, D. Vandermeulen, and P. Suetens, *Nonrigid image registration using free-form deformations with a local rigidity constraint*, Medical Image Computing and Computer-Assisted Intervention—MICCAI, Lecture Notes in Computer Science, vol. 3216, Springer, 2004, pp. 639–646. 169

[158] F. M, Nabavi A, Macq B, Jolesz FA, Kikinis R, and Warfield SK, *Registration of 3-D intraoperative MR images of the brain using a finite-element biomechanical model*, IEEE Transactions on Medical Imaging **20** (2001), 1384–1397. 167

[159] G. Malandain, S. Fernández-Vidal, and J.-M. Rocchisani, *Improving registration of 3-D images using a mechanical based method*, Computer Vision—ECCV Lecture Notes in Computer Science, vol. 801, Springer, 1994, pp. 131–136. 167

[160] D. N. Metaxas, L. Axel, G. Fichtinger, and G. Székely (eds.), *Medical Image Computing and Computer-Assisted Intervention—MICCAI 2008, 11th International Conference, New York, 2008, Proceedings, Part II*, Lecture Notes in Computer Science, vol. 5242, Springer, 2008. 177, 181

[161] M. I. Miller and L. Younes, *Group actions, homeomorphisms, and matching:*

A general framework, International Journal on Computer Vision **41** (2001), 61–84. 168

[162] J. Modersitzki, *Numerical methods for image registration*, Oxford University Press, New York, 2004. 12, 27, 33, 35, 58, 61, 62, 71, 96, 121, 123

[163] ———, *FLIRT with rigidity—image registration with a local non-rigidity penalty*, International Journal on Computer Vision **76** (2008), 153–163. 169

[164] P. Muyan-Ozcelik, J. D. Owens, J. Xia, and S. S. Samant, *Fast deformable registration on the GPU: A CUDA implementation of demons*, Proceedings of Int. Conf. on Computational Science and Its Applications IEEE, 2008, pp. 223–233. 170

[165] J. Nocedal and S. J. Wright, *Numerical optimization*, Springer, New York, 1999. 67, 77, 78, 139, 153, 154, 158

[166] H. Park, P. H. Bland, K. K. Brock, and C. R. Meyer, *Adaptive registration using local information measures*, Medical Image Analysis **8** (2004), 465–473. 167

[167] X. Pennec, P. Cachier, and N. Ayache, *Understanding the "demon's algorithm" 3D non-rigid registration by gradient descent*, Medical Image Computing and Computer-Assisted Intervention (C. Taylor and A. Colchester, eds.), Springer, 1999, pp. 597–605. 169

[168] X. Pennec, P. Fillard, and N. Ayache, *A Riemannian framework for tensor computing*, International Journal on Computer Vision **66** (2006), 41–66. 167

[169] L. Piegl and W. Tiller, *The* NURBS *book*, 2nd ed., Springer, New York, 1997. 27

[170] J. P. W. Pluim, J. B. A. Maintz, and M. A. Viergever, *Image registration by maximization of combined mutual information and gradient information*, IEEE Transactions on Medical Imaging **19** (2000), 809–814. 101, 167

[171] J. P. W. Pluim, J. B. A. Maintz, and M. A. Viergever, *Interpolation artefacts in mutual information based image registration*, Proceedings of the SPIE 2004, Medical Imaging, 1999 (K. M. Hanson, ed.), vol. 3661, SPIE, 1999, pp. 56–65. 101, 167

[172] ———, *Mutual-information-based registration of medical images: A survey*, IEEE Transactions on Medical Imaging **22** (1999), 986–1004. 101

[173] A. Roche, *Recalage d'images médicales par inférence statistique*, Ph.D. thesis, Université de Nice, Sophia-Antipolis, France, 2001. 101, 167

[174] T. Rohlfing, C. R. Maurer, Jr., D. A. Bluemke, and M. A. Jacobs, *Volume-preserving nonrigid registration of MR breast images using free-form deformation with an incompressibility constraint*, IEEE Transaction on Medical Imaging **22** (2003), 730–741. 169

[175] K. Rohr, *Landmark-based image analysis*, Computational Imaging and Vision, Kluwer Academic Publishers, Dordrecht, 2001. 57, 61, 123

[176] D. Rueckert, C. Hayes, C. Studholme, P. Summers, M. Leach, and D. J. Hawkes, *Non-rigid registration of breast MR images using mutual information*, Medical Image Computing and Computer-Assisted Intervention—MICCAI '98, (A. C. F. Colchester, S. L. Delp, and W. M. Wells III, eds.), Springer, 1998, pp. 1144–1152. 47

[177] D. Rueckert, L. Sonoda, C. Hayes, D. Hill, M. Leach, and D. Hawkes, *Non-*

rigid registration using free-form deformations, IEEE Transactions on Medical Imaging **18** (1999), 712–721. 168, 169

[178] D. Rueckert, P. Aljabar, R. A. Heckemann, J. V. Hajnal, and A. Hammers, *Diffeomorphic registration using B-splines*, 9th International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI 2006), 2006, pp. 702–709. 168

[179] A. H. Salden, B. M. Ter Haar Romeny, and M. A. Viergever, *Linear scale-space theory from physical principles*, Journal of Mathematical Imaging and Vision **9** (1998), 103–139. 166

[180] O. Schmitt, *Die multimodale Architektonik des menschlichen Gehirns*, Habilitation, Insitute of Anatomy, Medical University of Lübeck, Germany, 2001. 18

[181] J. A. Schnabel, C. Tanner, A. D. C. Smith, A. Degenhard, M. O. Leach, D. R. Hose, D. L. G. Hill, and D. J. Hawkes, *Validation of non-rigid image registration using finite element methods: Application to breast MR images*, IEEE Transactions on Medical Imaging **22** (2003), 238–247. 168

[182] M. Sermesant, C. Forest, X. Pennec, H. Delingette, and N. Ayache, *Deformable biomechanical models: Application to 4D cardiac image analysis*, Medical Image Analysis **7** (2003), 475–488. 167

[183] G. C. Sharp, N. Kandasamy, H. Singh, and M. Folkert, *GPU-based streaming architectures for fast cone-beam CT image reconstruction and demons deformable registration*, Physics in Medicine and Biology **52** (2007), 5771–5783. 170

[184] R. Shekhar, V. Walimbe, S. Raja, V. Zagrodsky, M. Kanvinde, G. Wu, and B. Bybel, *Automated 3-dimensional elastic registration of whole-body PET and CT from separate or combined scanners*, Journal of Nuclear Medicine **46** (2005), 1488–1496. 18, 113, 114

[185] B. W. Silverman, *Density estimation for statistics and data analysis*, Chapman and Hall, London, New York, 1986. 102

[186] M. Staring, S. Klein, and J. P. W. Pluim, *A rigidity penalty term for nonrigid registration*, Medical Physics **34** (2007), 4098–4108. 169

[187] M. Staring, S. Klein, and J. P. W. Pluim, *Nonrigid registration using a rigidity constraint*, Proceedings of the SPIE 2006, Medical Imaging, 2006 (J. M. Reinhardt and J. P. W. Pluim, eds.), SPIE-6144, 2006, pp. 1–10. 169

[188] T. Steihaug, *The conjugate gradient method and trust regions in large scale optimization*, SIAM Journal on Numerical Analysis **20** (1983), 626–637. 158

[189] R. Strzodka, M. Droske, and M. Rumpf, *Image registration by a regularized gradient flow—a streaming implementation in DX9 graphics hardware*, Computing **73** (2004), 373–389. 170

[190] P. Thévenaz, T. Blu, and M. Unser, *Image interpolation and resampling*, Handbook of Medical Imaging, Processing and Analysis (I. N. Bankman, ed.), Academic Press, San Diego 2000, pp. 393–420. 27, 167

[191] P. Thévenaz, U. E. Ruttimann, and M. Unser, *A pyramid approach to subpixel registration based on intensity*, IEEE Transactions on Image Processing **7** (1998), 27–41. 166

[192] P. Thévenaz and M. Unser, *Optimization of mutual information for multires-*

olution image registration, IEEE Transactions on Image Processing **9** (2000), 2083–2099. 166

[193] J.-P. Thirion, *Non-rigid matching using demons*, Proceedings of Computer Vision and Pattern Recognition—CVPR '96, San Francisco, IEEE, 1996. 169

[194] ———, *Image matching as a diffusion process: An analogy with Maxwell's demons*, Medical Image Analysis **2** (1998), 243–260. 121, 169

[195] U. Trottenberg, C. Oosterlee, and A. Schüller, *Multigrid*, Academic Press, London, 2001. 134

[196] A. Trouvé, *Diffeomorphisms groups and pattern matching in image analysis*, International Journal on Computer Vision **28** (1998), 213–221. 64, 168

[197] C. J. Twining and S. Marsland, *Constructing diffeomorphic representations for the groupwise analysis of nonrigid registrations of medical images*, IEEE Transactions on Medical Imaging **23** (2004), 1006–1020. 64, 168

[198] S. Uchida and H. Sakoe, *Piecewise linear two-dimensional warping*, Proceedings of the 15th International Conference on Pattern Recognition, Barcelona, Spain, vol. 3, 2000, pp. 538–541. 167

[199] M. Unser, P. Thévenaz, and L. Yaroslavsky, *Convolution-based interpolation for fast, high-quality rotation of images*, IEEE Transactions on Image Processing **4** (1995), 1371–1381. 167

[200] P. A. van den Elsen, E.-J. D. Pol, and M. A. Viergever, *Medical image matching—a review with classification*, IEEE Engineering in Medicine and Biology **12** (1993), 26–38. 47

[201] T. Vercauteren, X. Pennec, A. Perchant, and N. Ayache, *Non-parametric diffeomorphic image registration with the demons algorithm*, Medical Image Computing and Computer-Assisted Intervention—MICCAI, Brisbane, 2007, pp. 319–326. 168

[202] P. A. Viola, *Alignment by maximization of mutual information*, Ph.D. thesis, Massachusetts Institute of Technology, 1995. 100

[203] G. Wahba, *Spline models for observational data*, SIAM, Philadelphia, 1990. 33, 167

[204] S. K. Warfield, M. Ferrant, X. Gallez, A. Nabavi, and F. A. Jolesz, *Real-time biomechanical simulation of volumetric brain deformation for image guided neurosurgery*, Proceedings of the 2000 ACM/IEEE Conference on Supercomputing (CDROM), IEEE Computer Society, 2000, p. 23. 167, 170

[205] S. K. Warfield, F. A. Jolesz, and R. Kikinis, *A high performance computing approach to the registration of medical imaging data*, Parallel Computing **24** (1998), 1345–1368. 170

[206] J. Weickert and C. Schnörr, *A theoretical framework for convex regularizers in PDE-based computation of image motion*, International Journal on Computer Vision **45** (2001), 245–264. 167

[207] P. Wesseling, *An introduction to multigrid methods*, John Wiley & Sons, Chichester, 1992. 134

[208] G. Wollny and F. Kruggel, *Computational cost of nonrigid registration algorithms based on fluid dynamics*, IEEE Transactions on Medical Imaging **21** (2002), 946–952. 170

[209] B. Yeo, M. Sabuncu, T. Vercauteren, N. Ayache, B. Fischl, and P. Golland,

Spherical demons: Fast surface registration, in Metaxas et al. [160], pp. 745–753. 170

[210] J. You, W. P. Zhu, E. Pissaloux, and H. A. Cohen, *Parallel image matching on a distributed system*, Proceedings of ICAPP, vol. 2, IEEE, 1995, pp. 870–873. 170

[211] L. Zhu, S. Haker, and A. Tannenbaum, *Area preserving mappings for the visulization of medical structures*, Medical Image Computing and Computer-Assisted Intervention—MICCAI, Springer, 2003, pp. 277–284. 169

[212] L. Zollei, J. Fisher, and W. Wells, *A unified statistical and information theoretic framework for multi-modal image registration*, Information Processing in Medical Imaging (IPMI 2003), Lecture Notes in Computer Science, vol. 2732, Springer, 2003, pp. 366–377. 169

Symbols, Acronyms, Index

Symbols

$\langle \cdot, \cdot \rangle$, *inner product*
$\odot$, *Hadamard product*
$\otimes$, *Kronecker product, see* FAIR *2 (p.17)*
$\| \cdot \|$, *L_2-norm and ℓ_2-norm*
$| \cdot |$, *Euclidian norm in $\mathbb{R}^d$*
α, *regularization parameter (Section 6.3)*
β, *stability parameter for Hessian (Section 6.3)*
η, *edge parameter (Section 7.4)*
Ω, *spatial domain, see* FAIR *1 (p.11)*
ρ, *radial basis function (Section 5.3.1) or joint density (Section 7.3)*
σ, *kernel width (Section 7.3)*
θ, *scale-space parameter (Section 3.6)*
d, *spatial dimension, see* FAIR *1 (p.11)*
H, `H`, *(approximative) Hessian of J; entropy, cf. eq. (7.6); coarse grid*
h, *cell size, discretization width (Chapter 3), fine grid*
$I = I_n$, *see matrix, identity*
`M`, *regularization matrix (Section 6.3.2) or approximation to d^2D (Section 9.1)*
m, *discretization size*
n, *number of discretization points*
$n[\mathcal{T}]$, *normalized image gradient (Section 7.4)*
$\mathcal{O}$, *see Landau symbol*
`omega`, *MATLAB representation of spatial domain Ω*
`P`, *see matrix, interpolation or prolongation operator*
$\mathcal{Q}$, Q, *see basic function (Chapter 4)*
r_j, *landmark in reference image (Chapter 5)*
t_j, *landmark in template image (Chapter 5)*
u, *displacement $u = y - y^{\text{ref}}$*
`uc`, *current discretized displacement*
W, *weighting matrix for spline regularization (Chapter 3)*
w, `wc`, *coefficient vector for transformations (Chapter 4)*
`wRef`, *inhomogeneity for parameter regularization (Section 6.3.2)*
`xc`, *current grid (Chapter 3)*
y, *transformation, see* FAIR *1 (p.11)*
y^h, `yc`, *discretized transformation*
y^{ref}, `yRef`, *see regularization inhomogeneity (Section 8.2)*
derivative
  dD, `dD`, *of distance D*
  d^2D, `d2D`, *approx. 2nd of D*
  dJ, `dJ`, *of objective J*
  dS, `dS`, *of regularizer S*
  d^2S, `d2S`, *2nd of regularizer*
  $\mathrm{d}T$, `dT`, *of template (Chapter 3)*
  dy, `dy`, *of parametric transformation*
differential operator
  Δ, *Laplacian*
  $\nabla \cdot$, *divergence*

Acronyms

Index